UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE GUAYAQUIL

CARRERA DE INGENIERÍA ELECTRÓNICA

TRABAJO DE TITULACIÓN PREVIO A LA

OBTENCIÓN DEL TÍTULO DE INGENIERO ELECTRÓNICO

PROYECTO TÉCNICO:

"DISEÑO E IMPLEMENTACIÓN DE UN MÓDULO DE ENTRENAMIENTO PARA MICROCONTROLADOR 18F4550 CON SOFTWARE DIDÁCTICO PARA LA MATERIA DE MICROCONTROLADORES I Y II"

AUTORES:

JOSÉ EDUARDO LÓPEZ GUZMÁN
LISSETTE CAROLINA REINOSO FIGUEROA

TUTOR:

ING. BREMNEN MARINO VELIZ NOBOA PhD.

GUAYAQUIL – ECUADOR
2021

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA

Nosotros, José Eduardo López Guzmán y Lissette Carolina Reinoso Figueroa autorizamos a la **Universidad Politécnica Salesiana** la publicación total o parcial de este trabajo de titulación y su reproducción sin fines de lucro.

Además, se declara que los conceptos y análisis desarrollados y conclusiones del presente trabajo son de exclusiva responsabilidad del autor.

José Eduardo López Guzmán
C.I

Lissette Carolina Reinoso Figueroa
C.I.: 0923881155

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR

Nosotros, **José Eduardo López Guzmán**, con documento de identificación N° 0927228890 y **Lissette Carolina Reinoso Figueroa**, con documento de identificación N° 0923881155,manifestamos nuestra voluntad y ceder a la **UNIVERSIDAD POLITÉCNICA SALESIANA** la titularidad sobre los derechos patrimoniales en virtud de que somos autores del trabajo de grado titulado: **"DISEÑO E IMPLEMENTACIÓN DE UN MÓDULO DE ENTRENAMIENTO PARA MICROCONTROLADOR 18F4550 CON SOFTWARE DIDÁCTICO PARA LA MATERIA DE MICROCONTROLADORES I Y II"** mismo que 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 antes cedidos.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en condición de autor me reservo los derechos morales de la obra antes citada. En concordancia, suscrito este documento en el momento que se realiza la entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Politécnica Salesiana.

---------- ----------

José Eduardo López Guzmán
C.I.:

Lissette Carolina Reinoso Figueroa
C.I.: 0923881155

CERTIFICADO DE DIRECCIÓN DE TRABAJO DE TITULACIÓN

Yo declaro que bajo mi dirección y asesoría fue desarrollado el trabajo de titulación "**DISEÑO E IMPLEMENTACIÓN DE UN MÓDULO DE ENTRENAMIENTO PARA MICROCONTROLADOR 18F4550 CON SOFTWARE DIDÁCTICO PARA LA MATERIA DE MICROCONTROLADORES I Y II**" con resolución de aprobación de Consejo de Carrera N.º RESOLUCION realizado por los estudiantes **José Eduardo López Guzmán**, con documento de identificación N° 0927228890 y **Lissette Carolina Reinoso Figueroa**, con documento de identificación N°0923881155, obteniendo un producto que cumple con los objetivos del diseño de aprobación, informe final y demás requisitos estipulados por la Universidad Politécnica Salesiana, para ser considerados como trabajo final de titulación.

Guayaquil, agosto del 2021

ING. BREMNEN MARINO VELIZ NOBOA PhD.

Docente

DEDICATORIA

Este trabajo está dedicado a mi madre Patricia Guzmán, a mi ángel que donde quiera que me esté viendo espero que estés orgullosa de mi tía Anita, mi tío Francisco Guzmán y mi abuelita… gracias a todos ellos por inculcarme responsabilidad, esfuerzo, valentía y determinación en todo lo que me he propuesto hacer en mi vida, por nunca dejar de creer en mí y siempre apoyándome para lograr mis metas, esto se los dedico a ustedes.

A mi prometida Briggitte Moreno por siempre ser mi apoyo incondicional, mi compañera de vida, quien siempre me alienta a seguir adelante, y es el impulso que le da a mi vida para nunca dejar de levantarme, esto también se lo dedico a usted.

José Eduardo López Guzmán

DEDICATORIA

El presente trabajo de titulación se lo dedico a mis padres, Mercy y William, por ser las fuerzas que siempre necesite para culminar esta carrera.

A mi hermana Eva, mis tíos Simón Figueroa y Mario Figueroa que siempre estuvieron presentes con su apoyo incondicional.

A mi esposo por ser el mejor ejemplo para seguir dentro de esta carrera que hoy culmino y a mis hijos que son mi mayor motivación para cumplir mis metas, los amo infinitamente.

A mis angelitos Evica y Don Simón, por ser mi bendición día con día, por sembrarme valores desde muy pequeña y cuidar de mí siempre.

Lissette Carolina Reinoso Figueroa

AGRADECIMIENTO

Agradezco a Dios, por darme esta oportunidad de vivir para poder cumplir mi sueño.

A mi madre quien con todo su amor y responsabilidad me ayudo a enfocarme únicamente en mis estudios y así poder darle el mejor orgullo que se merece.

A los docentes los cuales me enseñaron y me formaron en mi vida profesional.

A mi compañera de tesis, que me ayudo bastante con este trabajo de titulación.

José Eduardo López Guzmán

AGRADECIMIENTO

Agradezco a Dios, primeramente, porque sus tiempos son perfectos y me permiten compartir este logro con las personas que más amo.

A Christopher Henríquez, el principal motor de mi vida cuando inicie esta carrera por ser mi compañía siempre, gracias por ser la razón que me permitió continuar.

A los docentes con los que compartí a lo largo de estos años por su calidad de enseñanza y formación en mi vida profesional.

A la Lcda. Mariela Quisphe por ser mi guía en todo momento desde que inicie la carrera hasta hoy en día que la culmino.

A mi compañero de tesis, con el cual trabaje de manera eficiente.

Y finalmente a mi Esposo por su ayuda incondicional a que culmine este proyecto, te amo.

Lissette Reinoso de Lara

Lissette Carolina Reinoso Figueroa

RESUMEN

AÑO	ALUMNOS	DIRECTOR DE PROYECTO TÉCNICO	TEMA DE PROYECTO TÉCNICO
2021	José Eduardo López Guzmán Lissette Carolina Reinoso Figueroa	ING. BREMNEN MARINO VELIZ NOBOA MSc.	Diseño E Implementación De Un Módulo De Entrenamiento Para Microcontrolador 18f4550 Con Software Didáctico Para La Materia De Microcontroladores I Y II

El presente trabajo surgió como necesidad de optimización y mejora de los recursos para el aumento en el nivel de aprendizaje que está 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 , pantalla graficas de 128x64, conjunto de display de 7 segmentos 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, para que los alumnos de Ingeniería Electrónica y Automatización puedan ejecutar prácticas y desarrollar un entendimiento en aplicaciones basadas en microsistemas de automatización mediante controladores y experiencias que puedan aplicar en su vida profesional.

Mediante la integración del lenguaje de programación C se tendrá una integración con diferentes elementos mediante el uso de librerías y lograr integrar nuevos sistemas inteligentes basados en microcontroladores de 8 bits

enfocados en controles de lazo cerrado y análisis en controles de lazo abierto dando la posibilidad de usarse en controles neuronales, difuso e integrándose con otros programas como los de instrumentación virtual Labview y matemáticos como Matlab.

Se logro implementar ocho módulos, que ayudan a la metodología teórica y experimental impartidas por los docentes ayudando a las cátedras impartidas teniendo una metodología de carácter descriptiva y de campo mediante el diseño de una tarjeta de circuito impreso diseñada en el programa Eagle de la familia de autodesk.

Este trabajo aporta a los estudiantes una herramienta de aprendizaje práctico ya que permite la creación de aplicaciones mediante el uso del entrenador, complementando con el uso del simulador que facilita las actividades didácticas para los estudiantes de la Carrera de Ingeniería Electrónica de la Universidad Politécnica Salesiana.

PALABRAS CLAVES: 18f4550, microcontroladores, entrenadores, electrónica, automatización, PIC.

ABSTRACT

YEAR	STUDENTS	DIRECTOR OF TECHNICAL PROJECT	TECHNICAL PROJECT THEME
2020	• José Eduardo López Guzmán • Lissette Carolina Reinoso Figueroa	ING. BREMNEN MARINO VELIZ NOBOA MSc.	Design and Implementation of a Training Module for Microcontroller 18f4550 With Didactic Software for The Matter of Microcontrollers I and II

The present work arose as a need for optimization and improvement of resources to increase the level of learning that is focused on a trainer module for applications with 18F4550 microcontrollers of the microchip family, integrating different peripherals such as matrix keyboard, 16x20 pixel liquid screens , 128x64 graphic screen, 7 segment pixel display set, high pulse and low pulse keypads, outputs through LEDs, an output block for connections with the controller with male spikes, led matrices with their sent records of movements having as means of connecting jumper-type cables for the different sections of the board that allows the PIC C Compiler program as C programming language, so that students of Electronic Engineering and Automation can carry out practices and develop an understanding in applications based on automation microsystems through drivers and experiences that can be applied in your professional life.

Through the integration of the C programming language, there will be an integration with different elements using libraries and achieve the integration of new intelligent systems based on 8-bit microcontrollers focused on closed-loop controls and analysis on open-loop controls, giving the possibilities for

use in neural controls, fuzzy and integrating with other programs such as Labview virtual instrumentation and mathematics such as MATLAB.

Eight modules were implemented, which help the theoretical and experimental methodology taught by the teachers, helping the chairs taught by having a descriptive and field methodology the design of a printed circuit board designed in the Eagle program of the autodesk family.

This work provides students with a practical learning tool that allows the creation of applications using the trainer, complementing the use of the simulator that facilitates didactic activities for students of the Electronic Engineering Career of the Salesian Polytechnic University.

KEYWORDS: 18f4550, microcontrollers, trainers, electronics, automation, pic.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El proyecto está basado en la implementación de ocho módulos de entrenamiento para controladores PIC18F4550 de la familia microchip para la creación de rutinas de aprendizajes de los diferentes protocolos en los sistemas microcontrolador como SPI, i2c, can, rs232 y en el diseño de aplicaciones con sistemas de sensores y salidas digitales como analógicos.

El proyecto surgió como necesidad de actualizar los elementos en el laboratorio y repotenciar con el uso de los módulos para las PRÁCTICA de la materia de microcontroladores I y II con los estudiantes de la carrera de Ingeniería en Automatización Industrial de la Universidad Politécnica Salesiana Sede Guayaquil.

El objetivo principal de este proyecto es implementar una guía para el de desarrollo de las diferentes PRÁCTICAS didáctica y apoyar el aprendizaje estudiantil, de esta manera aumentar las destrezas y conocimientos adquiridos para su formación como ingeniero.

1. EL PROBLEMA

1.1. Antecedentes

Debido a la creciente innovación tecnológica respecto a la creación de placas electrónicas, diseños de PCBs y programación de microprocesadores, es cada vez más visible la creación de aparatos modernos y de un tamaño reducido es necesario un ambiente controlado donde el estudiante pueda realizar rutinas de aprendizaje enfocadas en microcontroladores .

Actualmente la materia de Microcontroladores I y II no cuenta con estos módulos de entrenamiento para la cantidad de alumnos, enfocándose en esta problemática se propone la creación de ocho placas de entrenamiento para la enseñanza plena y didáctica del estudiante de acuerdo con el plan analítico de la materia teniendo un complemento como una guía detallada de las PRÁCTICAS base a realizar y con un diseño capaz de implementar nuevas prácticas con la integración de nuevos elementos.

1.2. Importancia y Alcances

El módulo didáctico consta de los periféricos necesarios a utilizar en la materia microcontroladores evitando que el estudiante realice compras de elementos extras para el aprendizaje, teniendo como ventaja un equipo completo, permitiendo el desarrollo de aplicaciones con diferentes protocolos como UART,SPI,I2C,CAN, USB, ayudando al estudiante a tener conocimientos prácticos y teóricos respecto al uso de sistemas inteligentes basado en microcontroladores siendo una herramienta de enseñanza para los docentes de la Universidad Politécnica Salesiana Sede Guayaquil de la carrera Ingeniería electrónica.

Actualmente la materia de Microcontroladores I y II no cuenta con estos módulos de entrenamiento, sin embargo, se quiere garantizar que el estudiante cumpla con los proyectos, optimizando a la vez el desarrollo de sus

prácticas académicas de laboratorio, por esta razón la integración de esta herramienta pedagógica no solo fomentará el aprendizaje activo del estudiante sino también se convierte en una gran innovación para la materia y para la Universidad.

1.3. Delimitación

1.3.1. Temporal

La implementación de este proyecto se realizó en un intervalo de seis meses a partir de la aprobación de este.

1.3.2. Espacial

El proyecto se desarrolla en un ambiente controlado fuera de la universidad, evitando la estática que dañe los periféricos del módulo con la finalidad de entregarle a la universidad en las instalaciones del Laboratorio de digitales de la carrera Ingeniería Electrónica de la Universidad Politécnica Salesiana de la Sede Guayaquil. Figura 1.

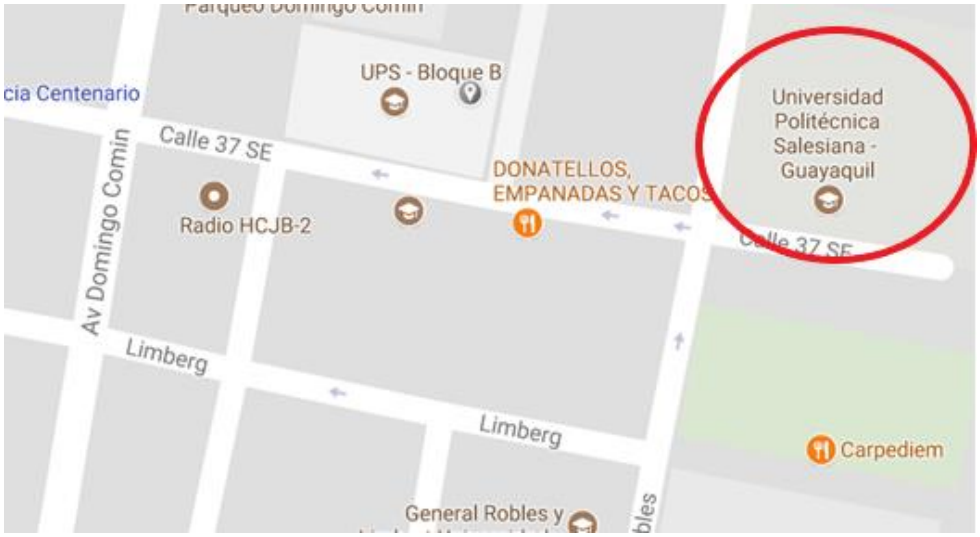


Figura 1: Delimitación espacial del módulo del Proyecto Técnico

1.3.3. Académica

Este módulo que se desarrolla e implementa para uso total de la Universidad Politécnica Salesiana - Sede Guayaquil, en la materia de Microcontroladores I y II. Para la inscripción de la carrera y para esta materia siempre hay una alta demanda de estudiantes alrededor de cuarenta por Semestre.

1.4. Innovación

El estudiante tendrá un módulo en el cual poder desarrollar las aplicaciones sin la necesidad de comprar elementos, también se pueden adaptar con el estudio de sistemas inteligentes basados en microcontroladores.

1.5. Objetivos

1.5.1. Objetivo general

- Diseñar e implementar el módulo de entrenamiento para microcontrolador 18F4550 con software didáctico para la materia de Microcontroladores I y II.

1.5.2. Objetivos específicos

- Analizar los fundamentos teóricos de los microcontroladores y sistemas para módulos de entrenamiento.
- Elaborar el diseño esquemático utilizando el dispositivo PIC 18F4550.
- Ejecutar la implementación y ensamblaje de los componentes para el módulo de entrenamiento.
- Elaborar la implementación del programa en PIC Compiler que incluirá el proceso para la creación de la práctica.
- Elaborar escenario de prácticas que contengan el sistema de microcontroladores de acuerdo con las asignaturas de microcontroladores I y II.

2. FUNDAMENTOS TEÓRICOS

2.1. Microcontroladores PIC

Todos sistemas basados en una arquitectura de 8 bits pertenecientes a la familia de microcontroladores PIC utilizan una arquitectura Harvard, lo que quiere que constan con una memoria de programa conectada a la CPU por más de 8 líneas. Existen diferentes tipos de arquitecturas basadas en microcontroladores de 12, 14 y 16 bits, dependiendo del tamaño del bus. La figura 2 muestra las características principales de estas tres categorías (Microchips, 2017).

Familia	ROM [Kbytes]	RAM [bytes]	Pines	Frecuencia de reloj. [MHz]	Entradas A/D	Resolución del convertidor A/D	Comparadores	Temporizadores de 8/16 bits	Comunicación serial	Salidas PWM	Otros
Arquitectura de la gama baja de 8 bits, palabra de instrucción de 12 bits											
PIC10FXXX	0.375 - 0.75	16 - 24	6 - 8	4 - 8	0 - 2	8	0 - 1	1 x 8	-	-	-
PIC12FXXX	0.75 - 1.5	25 - 38	8	4 - 8	0 - 3	8	0 - 1	1 x 8	-	-	EEPROM
PIC16FXXX	0.75 - 3	25 - 134	14 - 44	20	0 - 3	8	0 - 2	1 x 8	-	-	EEPROM
PIC16HVXXX	1.5	25	18 - 20	20	-	-	-	1 x 8	-	-	Vdd = 15V
Arquitectura de la gama media de 8 bits, palabra de instrucción de 14 bits											
PIC12FXXX	1.75 - 3.5	64 - 128	8	20	0 - 4	10	1	1 - 2 x 8 1 x 16	-	0 - 1	EEPROM
PIC12HVXXX	1.75	64	8	20	0 - 4	10	1	1 - 2 x 8 1 x 16	-	0 - 1	-
PIC16FXXX	1.75 - 14	64 - 368	14 - 64	20	0 - 13	8 or 10	0 - 2	1 - 2 x 8 1 x 16	USART I2C SPI	0 - 3	-
PIC16HVXXX	1.75 - 3.5	64 - 128	14 - 20	20	0 - 12	10	2	2 x 8 1 x 16	USART I2C SPI	-	-
Arquitectura de la gama alta de 8 bits, palabra de instrucción de 16 bits											
PIC18FXXX	4 - 128	256 - 3936	18 - 80	32 - 48	4 - 16	10 or 12	0 - 3	0 - 2 x 8 2 - 3 x 16	USB2.0 CAN2.0 USART I2C SPI	0 - 5	-
PIC18FXXJXX	8 - 128	1024 - 3936	28 - 100	40 - 48	10 - 16	10	2	0 - 2 x 8 2 - 3 x 16	USB2.0 USART Ethernet I2C SPI	2 - 5	-
PIC18FXXKXX	8 - 64	768 - 3936	28 - 44	64	10 - 13	10	2	1 x 8 3 x 16	USART I2C SPI	2	-

Figura 2: Características principales de 12, 14 y 16 bits (Microchips, 2017).

2.2. Arquitectura de los microcontroladores PIC de 8 bits

La arquitectura aplicada a los microcontroladores es la Harvard, que permite la conexión de la CPU de forma independiente y con buses distintos con la memoria de instrucciones y con la de datos y así permitir un acceso simultaneo (Villarán, 2005).

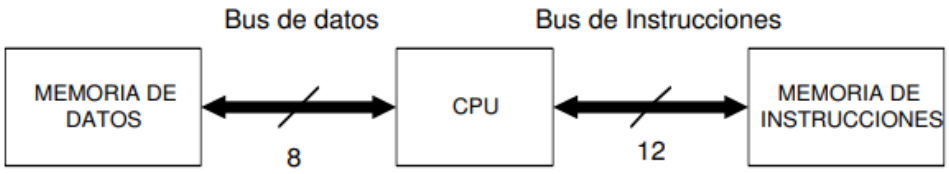


Figura 3: Arquitectura de microcontroladores (Villarán, 2005).

2.3. Funcionamiento de la arquitectura Harvard

La arquitectura Harvard tiene áreas diferentes de direcciones de memoria para el programa y para los datos.

Esto trae como consecuencia la capacidad de diseñar un circuito de tal manera que se pueda usar un bus y un circuito de control para manejar el flujo de información desde la memoria del programa y otro separado para manejar el flujo de información hacia la memoria de datos (Corvo, 2021)..

El uso de buses separados significa que es posible que la recuperación y ejecución de un programa se realice sin que haya ninguna interrupción por alguna transferencia ocasional de datos a la memoria de datos (Corvo, 2021)..

Por ejemplo, en una versión simple de esta arquitectura, la unidad de recuperación del programa podría estar ocupada recuperando la siguiente instrucción en la secuencia del programa y en paralelo realizar una operación de transferencia de datos que pudieron haber sido parte de la anterior instrucción del programa (Corvo, 2021)..

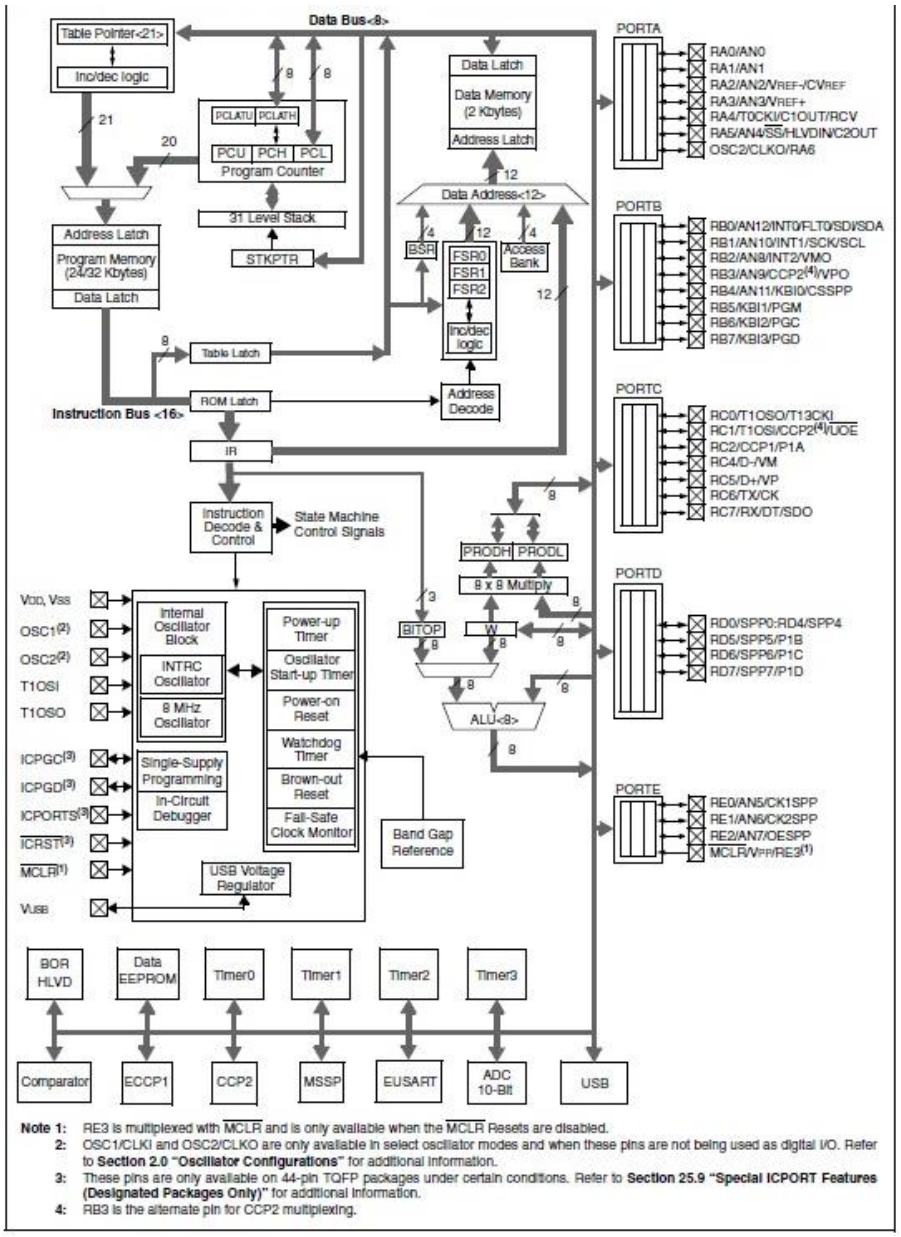
En este nivel la arquitectura Harvard tiene una limitación, ya que generalmente no es posible colocar el código del programa en la memoria de datos y ejecutarlo desde allí (Corvo, 2021).

2.4. PIC 18F4550

El Pic 18F4550 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 (TodoElectrodo, 2013).



Note 1: RE3 is multiplexed with MCLR and is only available when the MCLR Resets are disabled.
2: OSC1/CLKI and OSC2/CLKO are only available in select oscillator modes and when these pins are not being used as digital I/O. Refer to **Section 2.0 "Oscillator Configurations"** for additional information.
3: These pins are only available on 44-pin TQFP packages under certain conditions. Refer to **Section 25.9 "Special ICPORT Features (Designated Packages Only)"** for additional information.
4: RB3 is the alternate pin for CCP2 multiplexing.

Figura 4: Arquitectura interna del PIC 18F4550 (TodoElectrodo, 2013).

2.5. Memoria de programa del 18f4550

El Pic 18F4550 cuenta con una memoria de programa de 32K (32768 bytes). Es una memoria tipo Flash. Esta memoria es la que se encarga de almacenar las instrucciones, constantes y datos. La podemos escribir o leer con un programador externo o en ejecución.

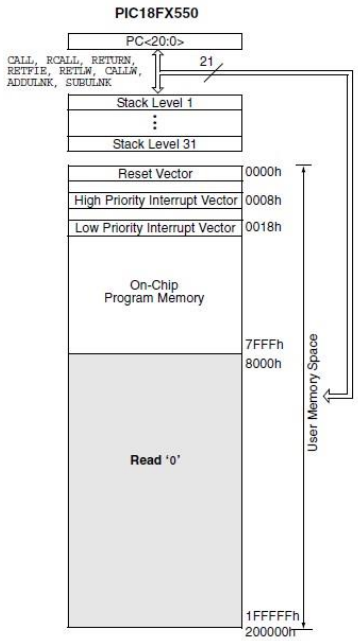


Figura 5: Memoria de programa del 18f4550 (TodoElectrodo, 2013).

2.6. Distribución de pines del 18f4550

El Pic 18F4550 cuenta con:

- Microcontroladores con modulo USB 2.0. Soporta Low speed 1.5Mb/s y full speed 12Mb/s.
- 1kB de memoria de doble acceso vía USB
- 35 pines I/O disponibles
- Memoria de programa flash de 32 kB
- RAM de 2048 Bytes
- EEPROM de datos de 256 Bytes

- Velocidad de la CPU 12 MIPS
- Oscilador externo de dos modos hasta 48 MHz
- Oscilador interno seleccionable entre 8 frecuencias desde 31kHz hasta 8MHz
- Oscilador secundario con Timer 1 de hasta 32kHz
- Opciones de oscilador dual permiten que la velocidad de la CPU y del módulo USB sean diferentes
- ADC de 10 bits y 13 canales
- Tecnología nano Watt que brinda características y funciones de bajo consumo y ahorro de energía
- Voltaje de operación 4.2V a 5.5V
- 4 Timer (desde Timer0 a Timer3). Uno de 8 bits y 3 de 16 bits
- 2 módulos de captura/comparación/PWM
- EUSART, SPP, SPI, I^2C.
- 20 fuentes de interrupciones (3 externas)
- Resistencias de pull-ups en el puerto B programables
- Función del pin MCLR opcional
- Brown-out Reset de valor programable.
- Power-on Reset
- Power-up Timer y Oscillator Start-up Timer
- Soporta 100,000 ciclos de borrado/escritura en memoria flash
- Soporta 1,000,000 ciclos de borrado/escritura en memoria EEPROM
- Retención de datos mayor a 40 años
- Protección de código y datos programable
- Encapsulado DIP de 40 pines

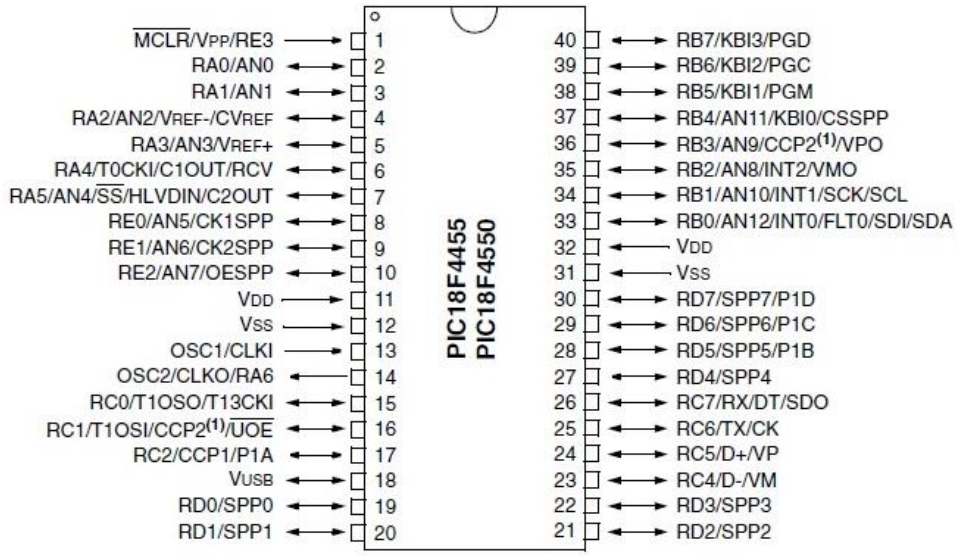


Figura 6: Distribución de pines del 18f4550 (Microchips, 2017)

2.7. Puertos de entrada y salida

Los pines de comunicación de los microcontroladores se agrupan en conjuntos llamados puertos porque dejan entrar y salir la información al procesador. Algunos pines de entrada salida están multiplexados con una función alternativa de algún periférico. Cuantos más periféricos posea el modelo, más líneas de comunicación con más multiplexado de señales exige.

En general, cuando un periférico está activado, el pin correspondiente no puede ser usado como pin de propósito general. En los 18FXX2 existen 5 puertos, y cada uno consta de 8 pines (excepto el puerto A que tiene 7 y el puerto E que tan solo tiene tres). En total el dispositivo tiene 40 patas, que sirven para dar entrada y salida a los convertidores analógico digital, a los comparadores, a las comunicaciones paralelo, serie, a los temporizadores... además de servir como pines de propósito general (Villarán, 2005).

Cada puerto tiene tres registros para su manejo, los cuales son:

- TRIS, registro de dirección de los datos.
- PORT, lee el nivel en los pines del dispositivo.
- LAT, Hatch de salida.

2.7.1. Puerto A

Se trata de un puerto bidireccional de 7 bits. Dependiendo del valor en TRISA se comportarán como entrada o salidas. Al leer el registro PORTA se obtiene el estado de los pines. El pin RA4 es multiplexado con la entrada de reloj del TIMER 0, el resto de los pines del puerto tiene niveles de entrada TTL y manejadores de salida CMOS. Estos pines, además, están multiplexados con entradas analógicas y las entradas de referencia Vref + y Vref -. El modo de operación de cada puerto es seleccionado mediante la puesta a '0' o a '1' del registro ADCON1 (Villarán, 2005)..

2.7.2. Puerto B

Registro bidireccional de anchura 8 bits. TRISB y PORTB tienen el mismo cometido que en el puerto A. Hay que anotar que tras un reset estos pines son configurados como entradas digitales.

Cuatro de los pines del puerto B, RB4:RB7, pueden ser fuente de interrupción cuando cambia el valor que tienen a la entrada. Sólo si están configurados como entradas es posible que actúen de esta forma. RB0:RB2 pueden ser (si se configuran de tal modo) fuente de interrupción externa.

El pin RB3 puede ser configurado mediante el pin de configuración CCP2MX como pin periférico alternativo para el módulo de comparación CCP2 (Villarán, 2005).

2.7.3. Puerto C

Es un puerto bidireccional de 8 bits, cada patita actúa como entrada o salida digital, según la programación de TRISC. Además, también puede actuar como entrada o salida Sistema de desarrollo PIC18F452 Capítulo 3 de diversos periféricos internos, debiéndose tener cuidado en la configuración del registro TRISC en este caso (Villarán, 2005).

2.7.4. Puerto D

Se trata de un registro de 8 bits de anchura, con los correspondientes registros TRISD y PORTD. Además de cómo puertos de entrada salida digitales, también puede funcionar como Puerto Paralelo Esclavo para soportar la interconexión directa con el bus de datos de 8 bits de otro microprocesador (Villarán, 2005).

En este caso los buffers de entrada son TTL. Para funcionar en este modo hay que poner a 1 el bit 4 (PSMODE) de TRISE. En tal caso, las líneas RE del Puerto E pasan a soportar las señales de control CS, WR y RD, entre el Puerto D y el bus del microprocesador (Villarán, 2005)..

Cada vez que el microprocesador realiza un ciclo de lectura o escritura sobre el Puerto Del bit 7 (PSPIF) del registro PIR1 se pone a 1. 3.10.5 Puerto E Este puerto, que sólo dispone de tres patitas, se configura como entradas digitales tras un RESET. Las 3 patitas pueden funcionar como E/S digitales, según la programación de los tres bits menos significativos del registro TRISE. También pueden actuar como señales de control (RD, WR y CS) para el flujo de datos entre un microprocesador y el Puerto D, cuando está programada en el modo Esclavo. Deben programarse como entradas en este caso. Finalmente, también pueden realizar estos tres pines la función de canal de entrada analógica para el Conversor A/D, según la programación del registro ADCON1 (Villarán, 2005).

2.8. Protocolo de comunicación SPI

La interfaz periférica en serie (SPI) es un bus de interfaz comúnmente utilizado para enviar datos entre microcontroladores y pequeños periféricos, como registros de desplazamiento, sensores y tarjetas SD. Utiliza líneas separadas de reloj y datos, junto con una línea de selección para elegir el dispositivo con el que desea hablar (Sparkfun, 2018).

En SPI, solo un lado genera la señal de reloj (generalmente llamada CLK o SCK para Serial ClocK). El lado que genera el reloj se llama "maestro" y el otro lado se llama "esclavo". Siempre hay un solo maestro, pero puede haber varios esclavos (Sparkfun, 2018).

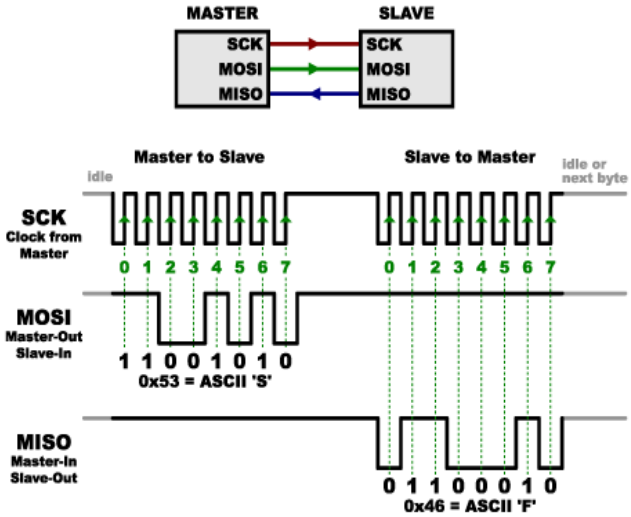


Figura 7: Protocolo Spi de comunicación (Sparkfun, 2018).

2.9. Protocolo de comunicación I2C

I2C significa Circuito integrado (Por sus siglas en Inglés Inter-Integrated Circuito) es un protocolo de comunicación serial desarrollado por Phillips Semiconductoras allá por la década de los 80s. Básicamente se creó para poder comunicar varios chips al mismo tiempo dentro de los televisores (Teslabem, 2017).

El I2C toma e integra lo mejor de los protocolos SPI y UART. Con el I2C podemos tener a varios maestros controlando uno o múltiples esclavos. Esto puede ser de gran ayuda cuando se van a utilizar varios microcontroladores para almacenar un registro de datos hacia una sola memoria o cuando se va a mostrar información en una sola pantalla (Teslabem, 2017).

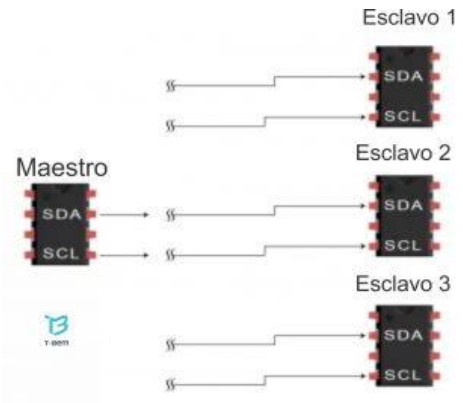


Figura 8: Protocolo i2c (Teslabem, 2017).

2.10. Protocolo de comunicación UART

El corazón del sistema de comunicaciones serie es la UART, acrónimo de Universal Asíncronos Receiver-Transmitter. Es un chip cuya misión principal es convertir los datos recibidos del bus del PC en formato paralelo, a un formato serie que será utilizado en la transmisión hacia el exterior. También realiza el proceso contrario: transformar los datos serie recibidos del exterior en un formato paralelo entendible por el bus (Zator, 2016).

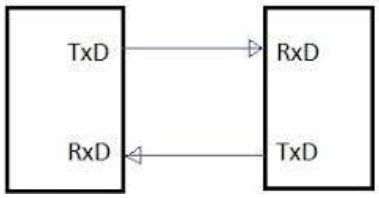


Figura 9: Comunicación UART (Zator, 2016).

2.11. Conversión Analógica a digital

El ADC realiza una muestra de la tensión analógica en un instante, y luego determina cuál sería el valor en binario en el lado de salida del ADC, el MAX1118EKA+T de Maxim Integrated, este dispositivo tiene una velocidad de muestreo de 100 kHz para que pueda probar la tensión analógica en su lado de entrada 100.000 veces por segundo al ser capaz de tomar muchas muestras en un segundo, es posible registrar con precisión cómo se vería la tensión analógica mediante una interpretación binaria (Digikey, 2017).

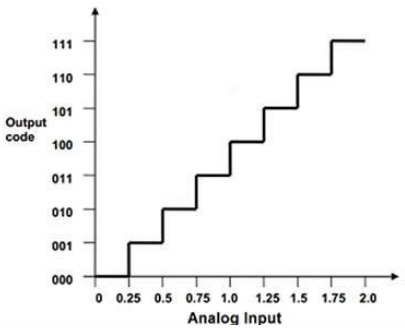


Figura 10: Conversión de analógico a la digital (Digikey, 2017).

2.12. Tarjetas de circuito impreso

Las tarjetas o placas de circuito impreso, también conocidas por sus siglas PCB (Printed Circut Board) son esenciales para conectar diversos componentes de tipo electrónico. Estas delicadas placas poseen caminos de cobre, la base de la tarjeta se realiza en diversos materiales como el plástico, fibra de vidrio, cerámica, o teflón. Pueden ser rígidas o flexibles y su diseño o diagramación se realiza en un software de PCB. La creación y producción de esta puede ser automatizada o manual (González, 2018).

Pues una PCB básicamente es **un soporte físico en donde se instalan componentes electrónicos** y eléctricos y se interconectan entre ellos. Estos componentes pueden ser, chips, condensadores, diodos, resistencias, conectores (González, 2018).

2.13. Componentes de una Pcb

Los componentes de una tarjeta de circuito impreso o PCB son los siguientes:

- Capa de cobre
- Soldermask o máscara de soldado
- Serigrafía o silkscreen
- Pads
- Caminos de cobre
- Perforaciones metálicas o drill

2.13.1. Capa de cobre

Las estructuras o capas de una PCB llamadas así porque tiene **una única capa exterior de cobre** (LLAMAS, 2020). Si obviamos la capa de pintura superficial (silkscreen) la estructura interna de una PCB de una capa se muestra en la figura 11.

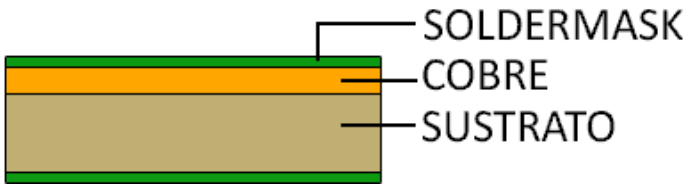


Figura 11: Estructura de una capa de una Pcb (LLAMAS, 2020)

2.13.2. Soldermask o máscara de soldado

Para montar los componentes electrónicos en los circuitos impresos se requiere de un proceso de ensamblado, que puede ser manual o mediante maquinaria especializada. Los procesos de ensamblado requieren la utilización de soldadura para poder fijar los componentes a la placa. Para evitar que la soldadura pueda cortocircuitar accidentalmente dos tracks pertenecientes a nodos distintos se utiliza una máscara de soldado, o soldermask en inglés (Cohen, 2010).

Esta máscara de soldado es un barniz que se aplica a los circuitos impresos en la etapa de fabricación y puede ser de variados colores. El color que se utiliza más frecuente es el verde seguidos del rojo y azul (Cohen, 2010).

2.13.3. Serigrafía o silkscreen

La serigrafía es el proceso en donde se imprime sobre la máscara de soldado información conducente a facilitar la labor del ensamblado y de posterior verificación. Generalmente se imprime para indicar puntos de prueba como también la posición, orientación y referencia de las componentes que conforman el circuito. También puede utilizarse para cualquier propósito que el diseñador requiera, como por ejemplo para el nombre del producto,

compañía, instrucciones de configuración, etc. La serigrafía puede ir en ambas capas externas o caras del circuito impreso. En inglés se conoce como silkscreen u overlay. En la figura 2 se puede apreciar la serigrafía, que corresponde a todo lo impreso en color blanco (Cohen, 2010)..

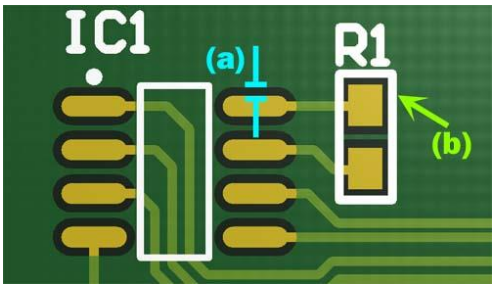


Figura 12: Soldermask o máscara de soldado (LLAMAS, 2020)

2.13.4. Pads

Un pad es una superficie de cobre en un circuito impreso o PCB que permite soldar o fijar la componente a la placa. Existen dos tipos de pads; los thru-hole y los SMD (montaje de superficie). (Cohen, 2010)..

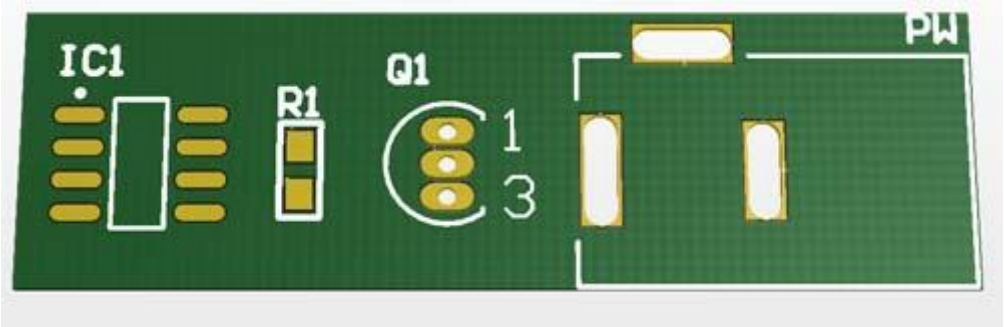


Figura 13: Pads (LLAMAS, 2020)

2.13.5. Caminos de cobres (pistas o tracks)

Un track es un camino conductor de cobre que sirve para conectar un pad (donde descansa el pin o terminal de un componente) a otro. Los tracks pueden ser de distinto ancho dependiendo de las corrientes que fluyen a través de ellos. (Cohen, 2010)..

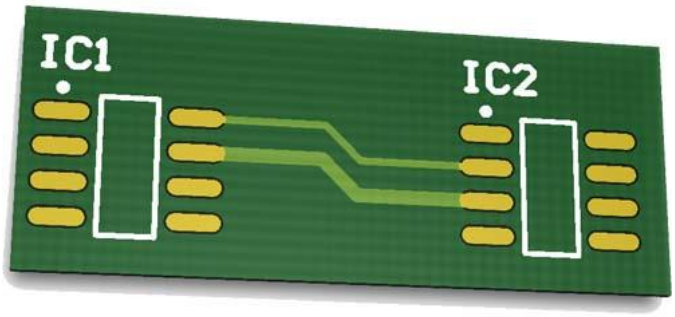


Figura 14: Caminos de cobres (LLAMAS, 2020)

2.13.6. Perforaciones Metalizadas a través de orificio (Thru-hole Vias o Full Stack Vias)

Cuando se debe realizar una conexión de un componente que se encuentra en la capa superior de la PCB con otro de la capa inferior, se utiliza una via. Una via es una perforación metalizada (en inglés, plated via) que permite que la conducción eléctrica no se interrumpa cuando se pasa de una superficie a otra. En la figura 15 puede apreciarse como salen 2 tracks desde los pads de un chip que se encuentra en la capa superior de la PCB, que luego de pasar por 2 vías, se conectan a los pads del chip que se encuentra en la capa inferior (Cohen, 2010)..

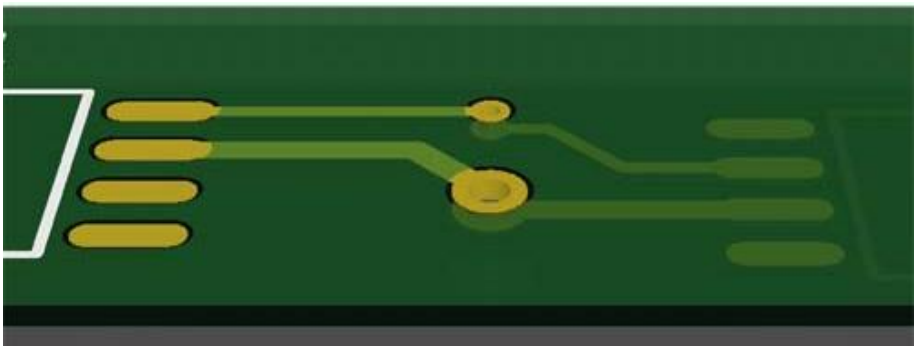


Figura 15: Perforaciones Metalizadas a través de orificio (LLAMAS, 2020)

2.14. Eagle - Autodesk

EAGLE (siglas de **E**asily **A**pplicable **G**raphical **L**ayout **E**ditor) es un programa de diseño de diagramas y PCBs con autoenrutador famoso alrededor del mundo de los proyectos electrónicos DiY, debido a que muchas versiones de

este programa tienen una licencia Freeware y gran cantidad de bibliotecas de componentes alrededor de la red (CADSOFT EAGLE, 2016).

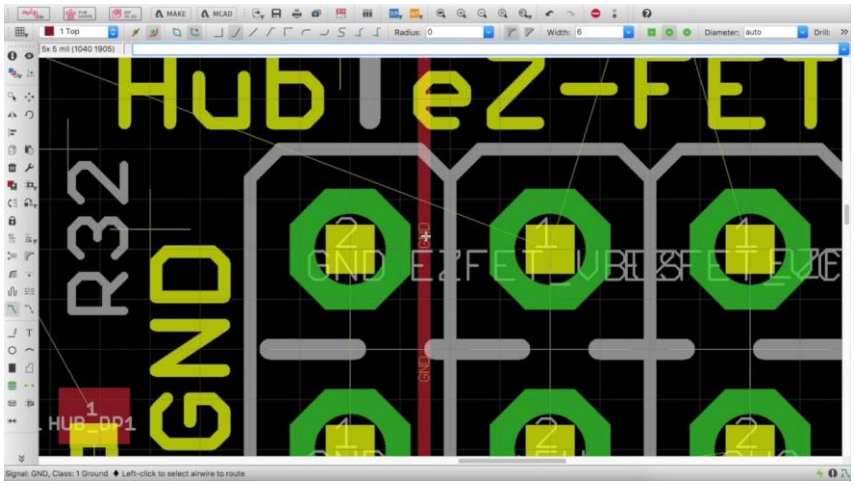


Figura 16: Autodesk Eagle (CADSOFT EAGLE, 2016).

2.15. Pantalla Grafica Glcd -St7920

Una pantalla gráfica de cristal líquido o GLCD (acrónimo del inglés Graphic Liquid Crystal Display) es una pantalla plana formada por una matriz de píxeles 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 energía eléctrica, hay versiones de pantallas con diferentes controladores embebidos, como el Samsung KS0107, Samsung KS0108 o el Toshiba T6963 (Franco Reina Rafael Christian, 2014).

Figura 17: Pantalla Grafica Glcd -St7920 (Franco Reina Rafael Christian, 2014)

La distribución de los pines se muestra en la figura 17.

Pin Num	Pin Name	Description
1	VSS	GND
2	VDD	Power Supply for Logic
3	V0	Operating Voltage for LCD
4	RS	H: data L: Instruction code
5	R/W	H: read L: write
6	E	Enable Signal
7	DB0	Data Bus Line
8	DB1	
9	DB2	
10	DB3	
11	DB4	
12	DB5	
13	DB6	
14	DB7	
15	PSB	H: Parallel Mode L: Serial Mode
16	NC	No Connection
17	/RST	Reset Signal, Active "L"
18	Vout	Negative Voltage Output
19	LEDA	Backlight Anode (+5V)
20	LEDK	Backlight Cathode (0V)

Figura 18: Pantalla Grafica configuración de pines (Franco Reina Rafael Christian, 2014)

2.16. 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 (Santiago, 2017).

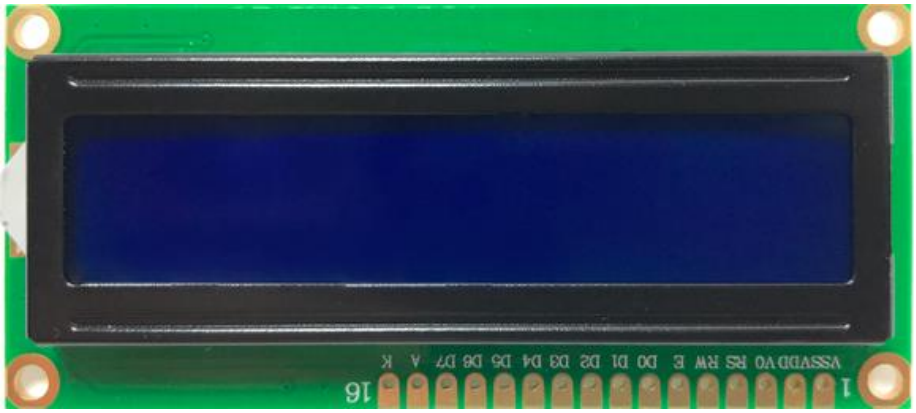


Figura 19: Pantalla LCD 20x4 (Santiago, 2017).

2.17. Teclado matricial 4x4

El Teclado matricial de botones plásticos formado por 4 filas y 4 columnas para un total de 16 teclas permite agregar una entrada de usuario a tus proyectos. El teclado es de tipo membrana, por lo que entre sus ventajas se encuentra el poco espacio que requiere para ser instalado. Posee una cubierta adhesiva y un cable flexible de conexión (Naylampmechatronics, 2016).

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 (Naylampmechatronics, 2016)..

La conexión del teclado matricial 4x4 con microcontroladores es simple: se necesitan 8 pines digitales en total. Puede trabajar con microcontroladores de 3.3V o 5V sin problema. Es necesario colocar resistencias pull-up entre los pines de las columnas y VCC. (Naylampmechatronics, 2016)..

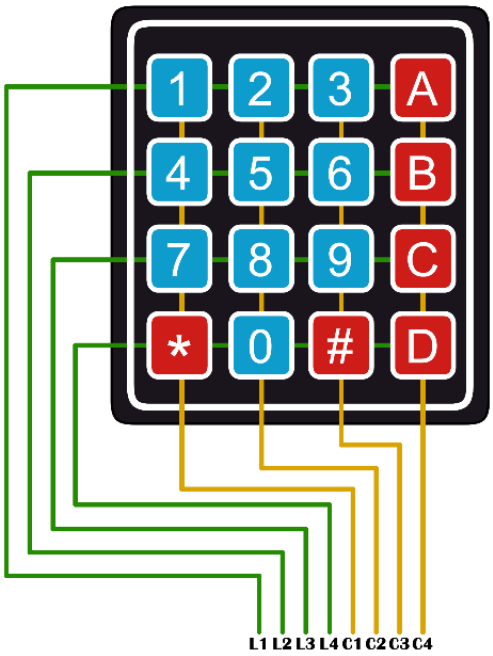


Figura 20: Teclado Matricial 4x4 (Naylampmechatronics, 2016).

2.18. Resistencia variable

Un potenciómetro es un resistor eléctrico con un valor de resistencia variable y generalmente ajustable manualmente. Los potenciómetros utilizan tres terminales y se suelen utilizar en circuitos de poca corriente, para circuitos de mayor corriente se utilizan los reóstatos. En muchos dispositivos eléctricos los potenciómetros son los que establecen el nivel de salida. Por ejemplo, en un altavoz el potenciómetro ajusta el volumen; en un televisor o un monitor de ordenador se puede utilizar para controlar el brillo (Mecafenix, 2017).

El valor de un potenciómetro viene expresado en ohmios (símbolo Ω) como las resistencias, y el valor del potenciómetro siempre es la resistencia máxima que puede llegar a tener. El mínimo lógicamente es cero. Por ejemplo, un potenciómetro de 10KΩ puede tener una resistencia variable con valores entre 0Ω y 10.000Ω (Mecafenix, 2017).

Figura 21: Resistencia variable (Mecafenix, 2017).

2.19. Pulsadores

Un pulsador eléctrico o botón pulsador es un componente eléctrico que permite o impide el paso de la corriente eléctrica cuando se aprieta o pulsa, y que al soltarlo vuelve a su posición inicial, para que el pulsador funcione, debe tener un resorte o muelle, que hace que vuelva a la posición anterior después de presionarlo (Granda, 2015).

Figura 22: Pulsadores abierto y cerrado (Granda, 2015).

2.20. Resistencias Pull up

Como su nombre indica esta resistencia tiene la función de “jalar” hacia “arriba”, lo que significa que polariza el voltaje hacia el voltaje de fuente (VDD) que puede ser +5V o +3.3V. De esta forma cuando el pulsador está abierto o en reposo, el voltaje en la entrada del Arduino siempre será de +5V. Las entradas del Arduino son de alta impedancia lo que significa que la corriente que circulará por esa línea sea mínima en el orden de los microamperios, por lo que el voltaje que “cae” en la resistencia pull-up es mínimo y tenemos casi el mismo voltaje de fuente en la entrada del microcontrolador (naylampmechatronics, 2016).

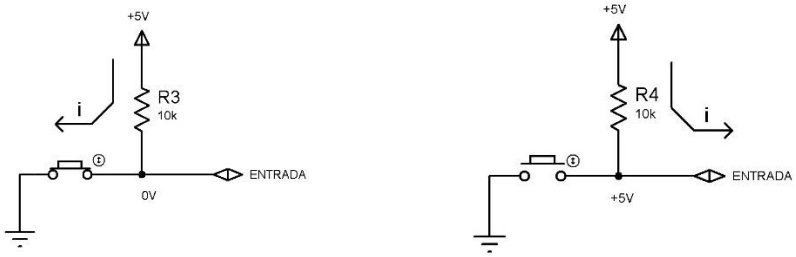


Figura 23: Resistencia’s Pull up (naylampmechatronics, 2016).

2.21. Resistencias Pull Down

De forma similar la resistencia pull-Down “jala” el voltaje hacia “abajo” o “0V”. Cuando el pulsador está en reposo, el voltaje en la entrada del Arduino será 0V. Cuando presionamos el pulsador la corriente fluye de +5V por el pulsador

hacia la resistencia y termina en 0V, de esa forma tenemos +5V en la entrada del microcontrolador (naylampmechatronics, 2016)..

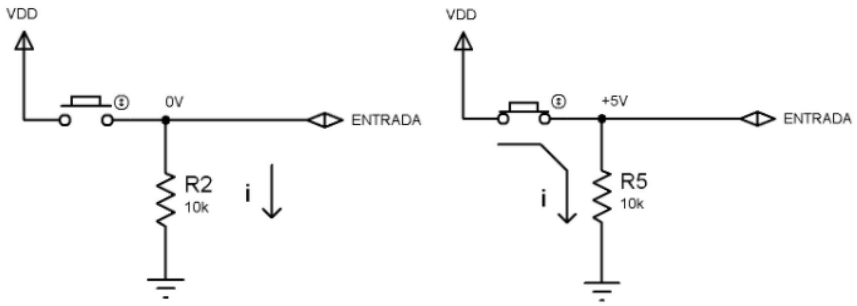


Figura 24: Resistencia Pull down (naylampmechatronics, 2016).

2.22. Display de 7 segmentos

El Display 7 Segmentos es un dispositivo optoelectrónico que permite visualizar números del 0 al 9. Existen dos tipos de Display, de cátodo común y de ánodo común. Este tipo de elemento de salida digital o display, se utilizaba en los primeros dispositivos electrónicos de la década de los 70's y 80's. Hoy en día es muy utilizados en proyectos educativos o en sistemas vintage. También debido a su facilidad de uso, mantenimiento y costo, son utilizados en relojes gigantes o incluso como marcadores en algunos tipos de canchas deportivas (Hetpro, 2015).

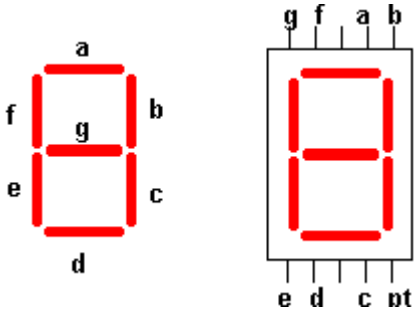


Figura 25: Display de 7 segmentos (Hetpro, 2015).

2.23. Conversor de binario a bcd

Los decodificadores son circuitos integrados que convierten una entrada de código binario a código BCD correspondiente para encender los segmentos

de un display para que se forme el valor en decimal que responde al número binario. Existen múltiples referencias, en este artículo hablaremos sobre los más usados, el 7447 y el 7448, los cuales son tecnología TTL, es decir, que su voltaje de alimentación es de 5 voltios (Li, 2012).

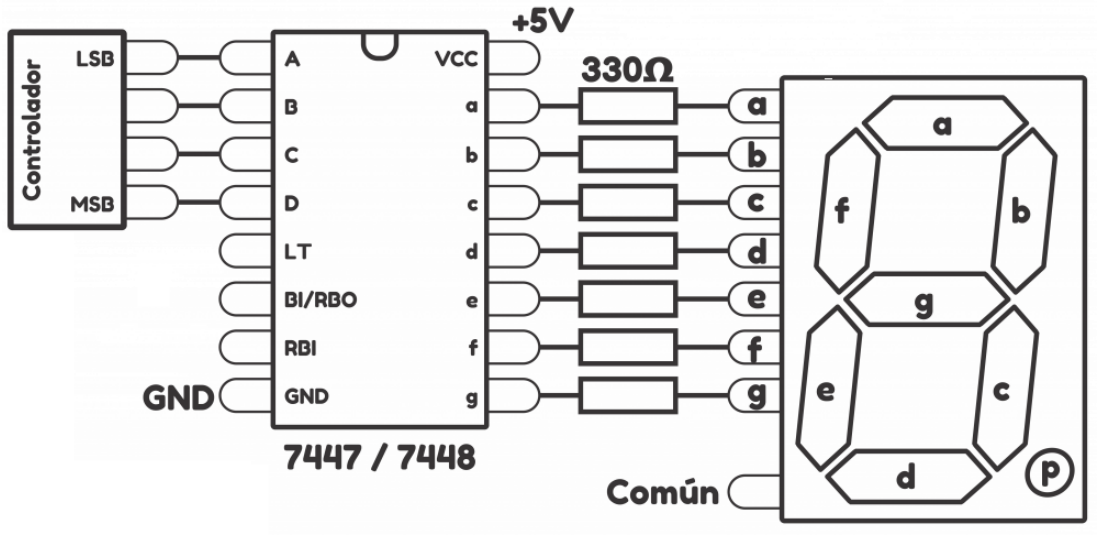


Figura 26: Conversor de binario a bcd (Li, 2012).

2.24. Registro de desplazamiento 4094

n registro de desplazamiento es un circuito digital secuencial (es decir, que los valores de sus salidas dependen de sus entradas y de los valores anteriores) consistente en una serie de biestables, generalmente de tipo D, conectados en cascada de forma sincrónica con la misma señal de reloj. Según las conexiones entre los biestables, se tiene un desplazamiento a la izquierda o a la derecha de la información almacenada. Es de señalar que un desplazamiento a la izquierda de un conjunto de bits multiplica por 2, mientras que uno a la derecha, divide entre 2. Existen registros de desplazamiento bidireccionales, que pueden funcionar en ambos sentidos. Los registros universales, además de bidireccionales permiten la carga en paralelo (Palazzesi, 2011).

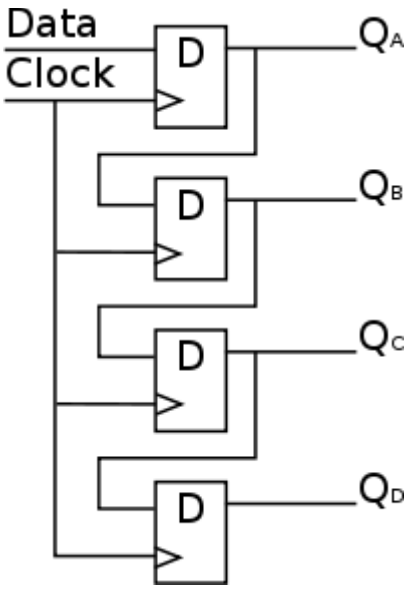


Figura 27: Registro de desplazamiento de 4 bits (Palazzesi, 2011).

2.25. Matrices LED 8x8

Una matriz LED es un display formado por múltiples LED en distribución rectangular. Existen distintos tamaños, siendo el más habitual los cuadrados de 8x8 LED, se puede combinar varios módulos para formar un display mucho mayor. En estos display podemos mostrar textos, dibujos o animaciones, como desplazar un texto, está formado por diferentes LED, **cableados de forma conjunta por filas y columnas**. Podemos encender un LED determinado de la matriz aplicando correctamente los valores HIGH y LOW a su respectiva fila y columna (Lui, 2016).

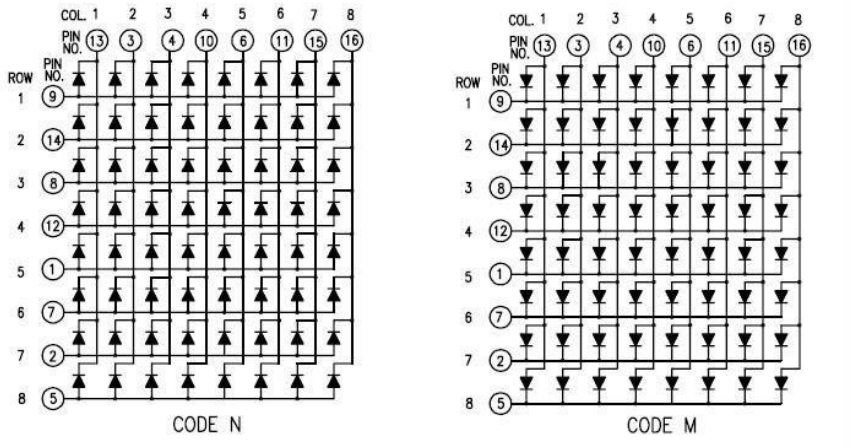


Figura 28: Matrices LED 8x8 (Lui, 2016).

3. MARCO METODOLÓGICO

3.1. Diseño del entrenador

Para el diseño del entrenador se optó el software Eagle por su viabilidad al momento de la creación de tarjetas de circuitos impreso en la figura 29 se muestra la placa en el software y en la figura 30 se aprecia la placa después de la fabricación.

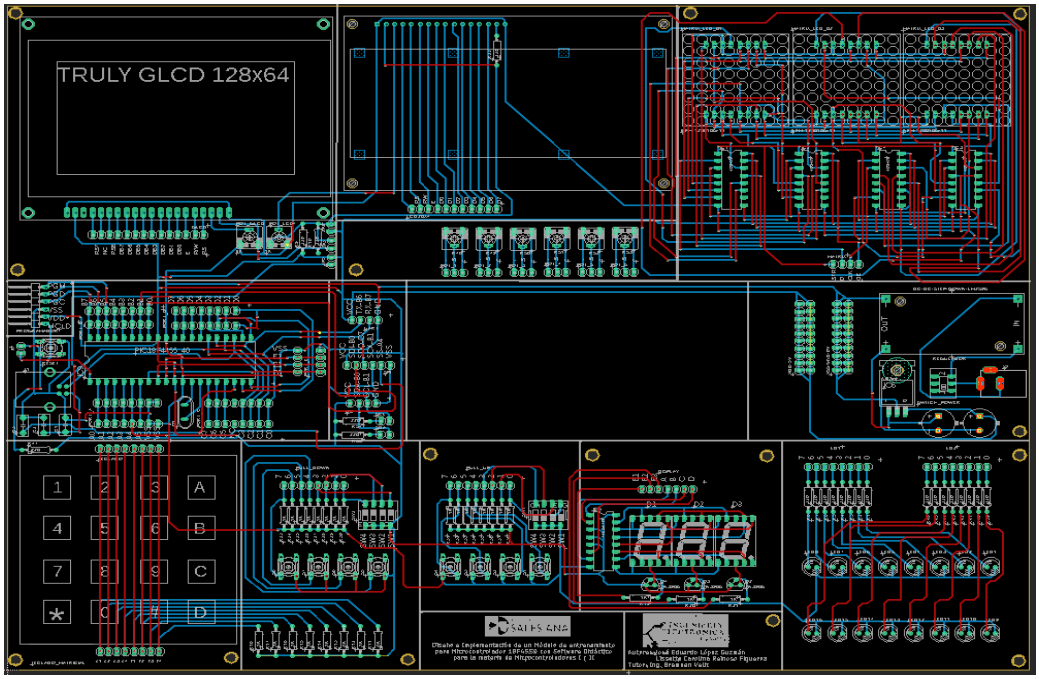


Figura 29. Montaje mediante arreglo modular.

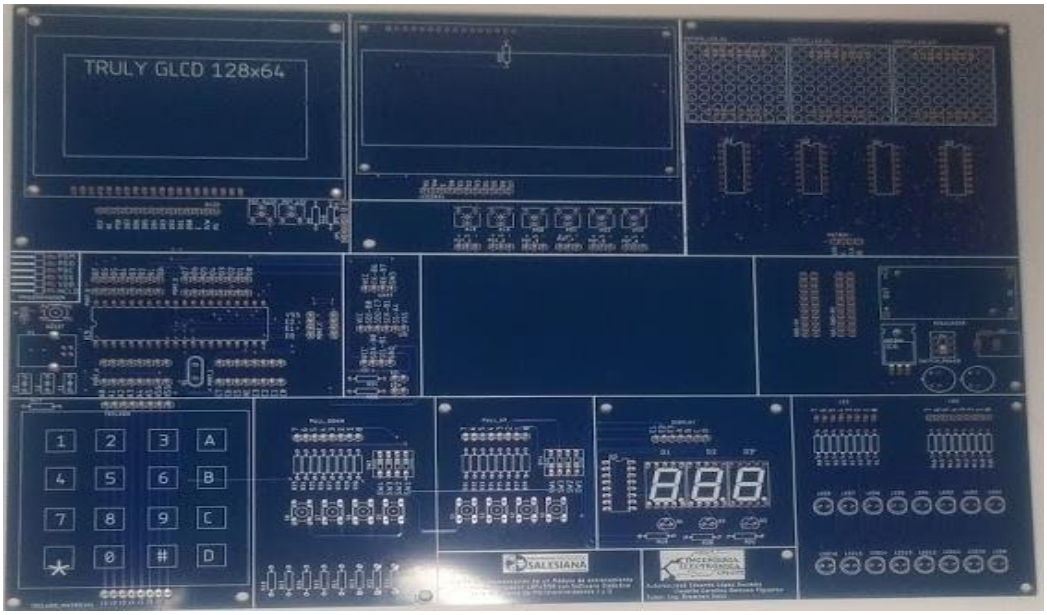


Figura 30. Montaje mediante arreglo modular.

3.2. Entrenador para microcontrolador 18f4550 esquemático

El presente proyecto tiene un diseño el cual cumpla con las necesidades del estudiante al momento de recibir las catedra de las materias de microcontroladores I y II, contando con diferentes secciones que se detallan en la sección 3.2. hasta 3.18.

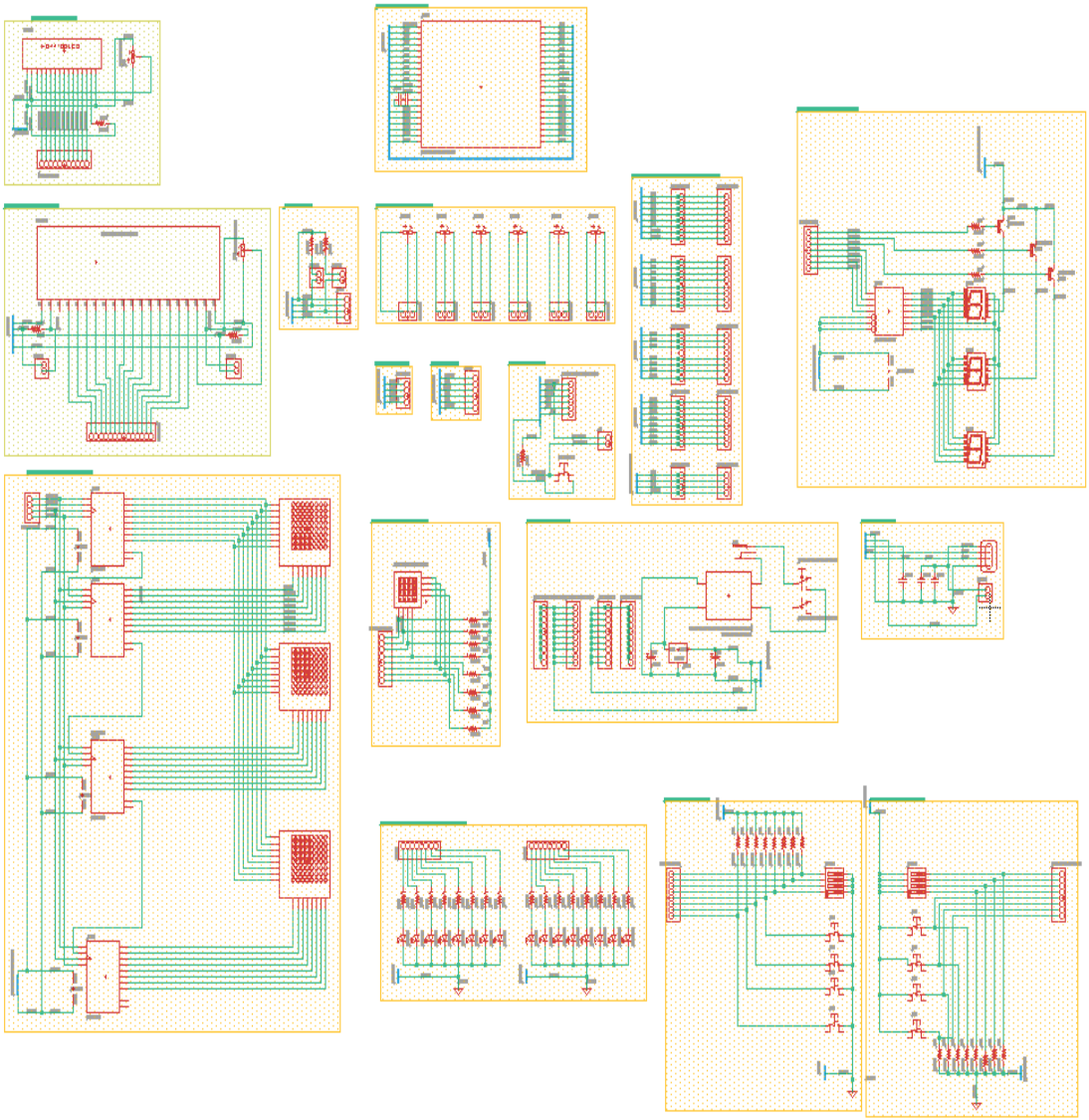


Figura 31. Circuito esquemático en Eagle Autodesk.

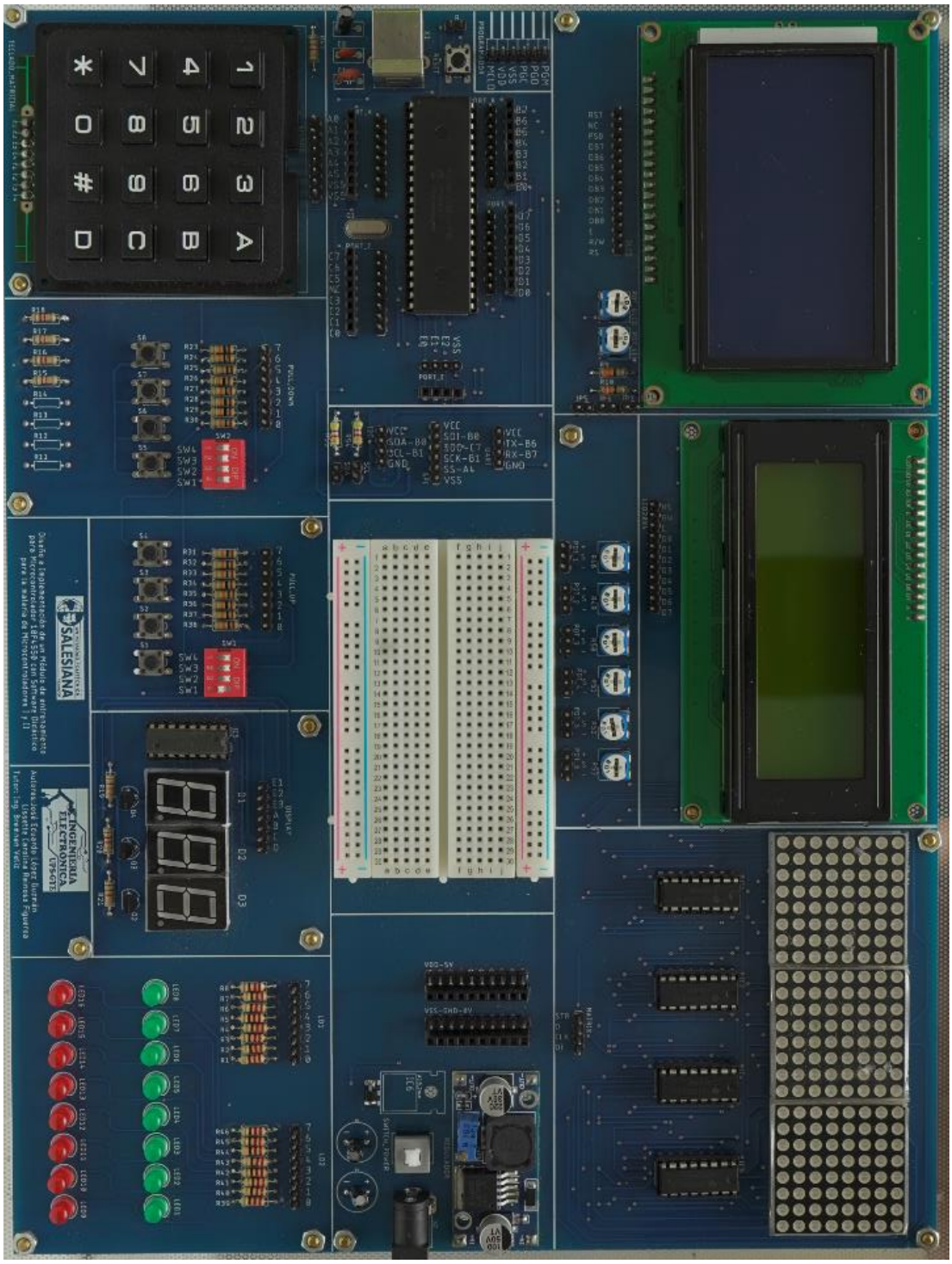


Figura 32. Entrenador PIC ensamblado.

3.3. Bloque del microcontrolador

El módulo consta de un núcleo que es un microcontrolador PIC 18F4550 como se muestra en la figura 33 en el dónde se programa mediante un periférico pickit 3. (programador para el PIC), se programa cada PRÁCTICA

individualmente para la función establecida en las PRÁCTICAS detalladas en el Capítulo 4 Manual de PRÁCTICAS

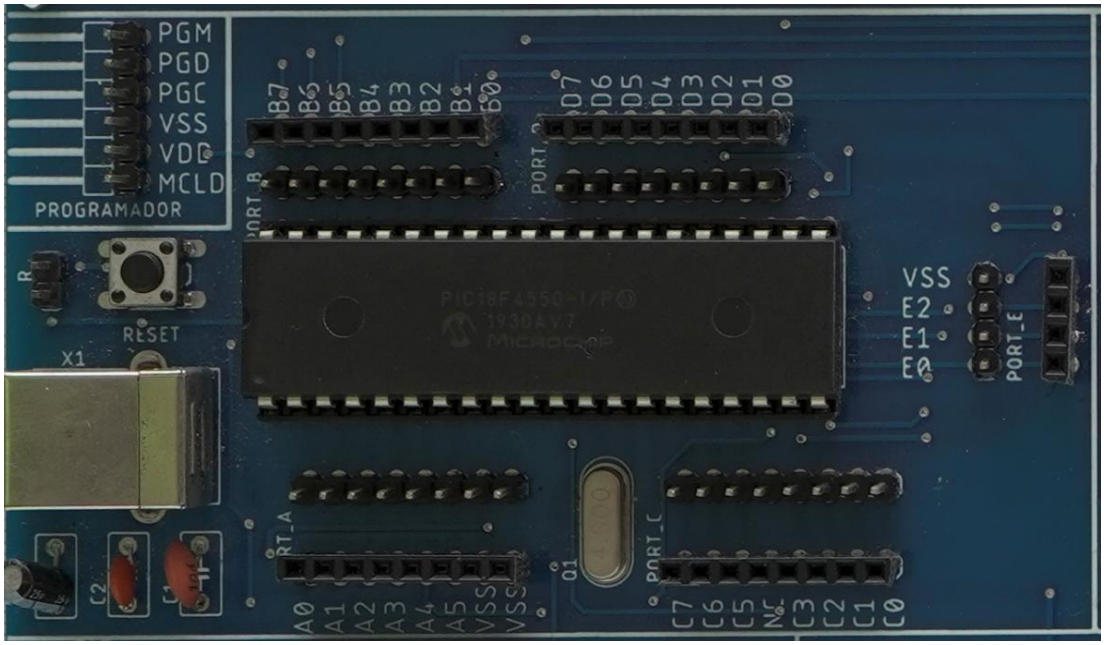


Figura 33. *Microcontrolador PIC 18F4550.*

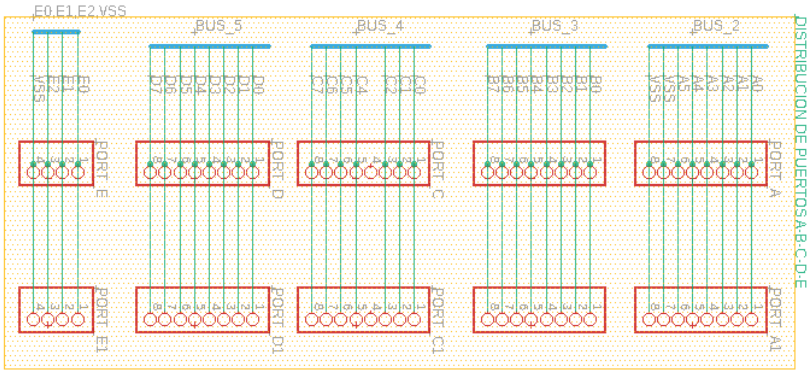


Figura 34. *Microcontrolador PIC 18F4550 - puertos*

3.4. Bloque del teclado

En el bloque de teclado consta de un arreglo de pulsadores de membrana los cuales estarán conectados al microcontrolador de acuerdo a la práctica que se realice o como el estudiante le necesite para futuras pruebas como se muestra en la figura 35.

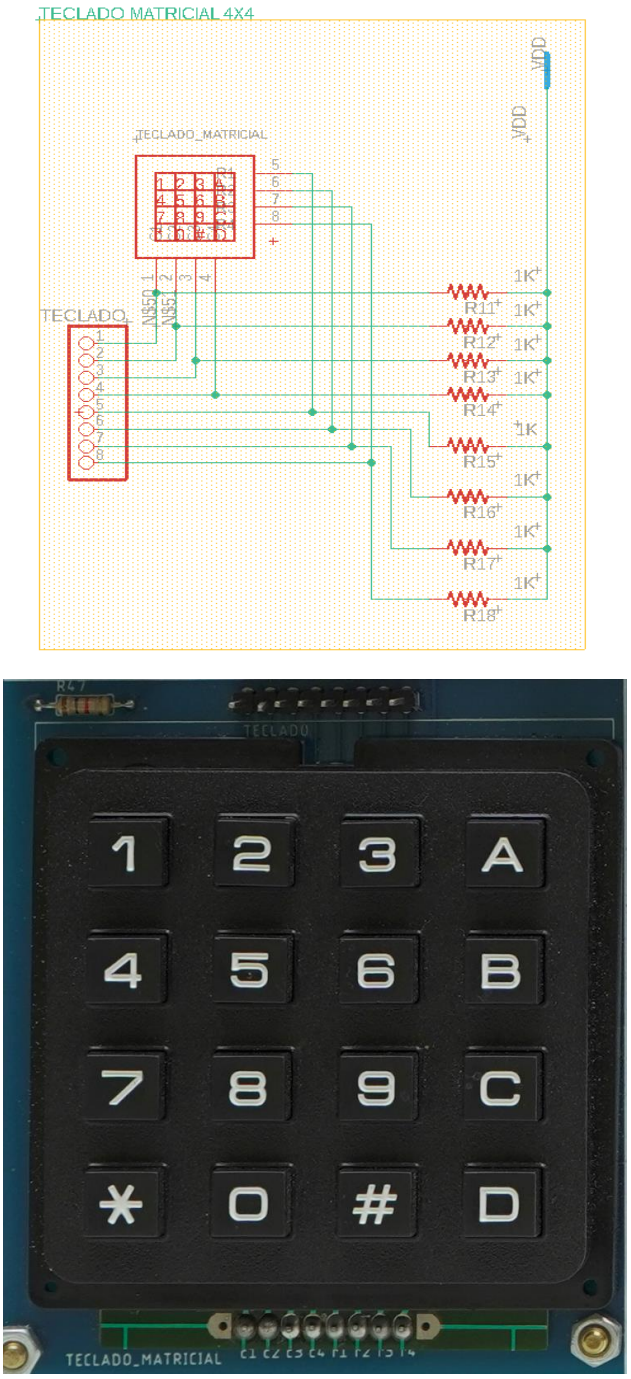
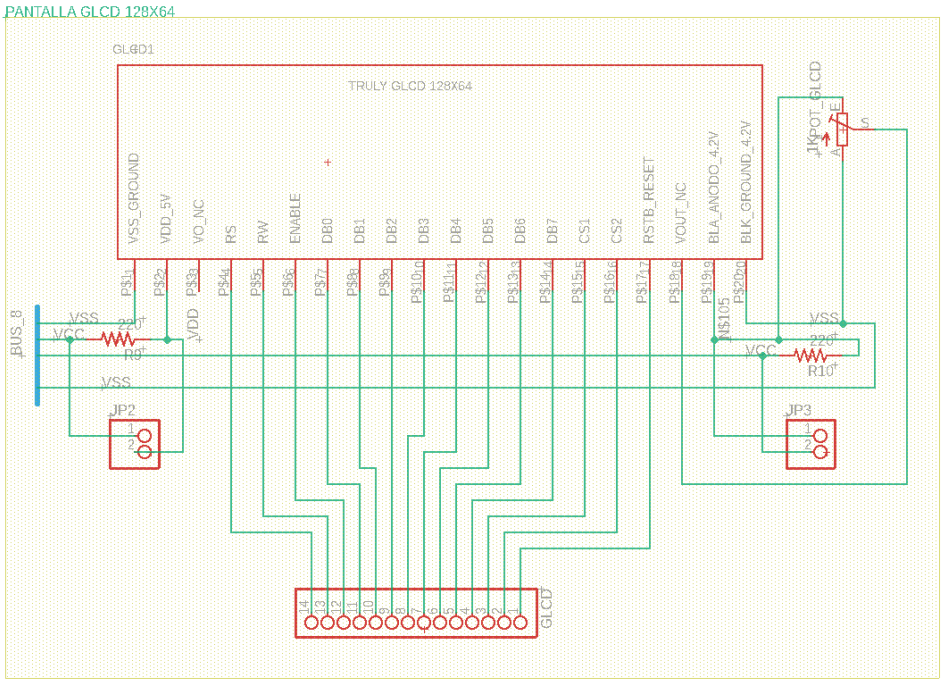


Figura 35. *Bloque del teclado.*

3.5. Bloque del GLCD

El bloque Glcd o pantalla grafica está constituido de 128 pixeles en el eje x y 64 pixeles en el eje y, permitiendo plotear o dibujar barras, figuras de acuerdo a la tarea a realizar en la guía de práctica, se muestra el esquemático y la Pcb con el periférico en la figura 35.



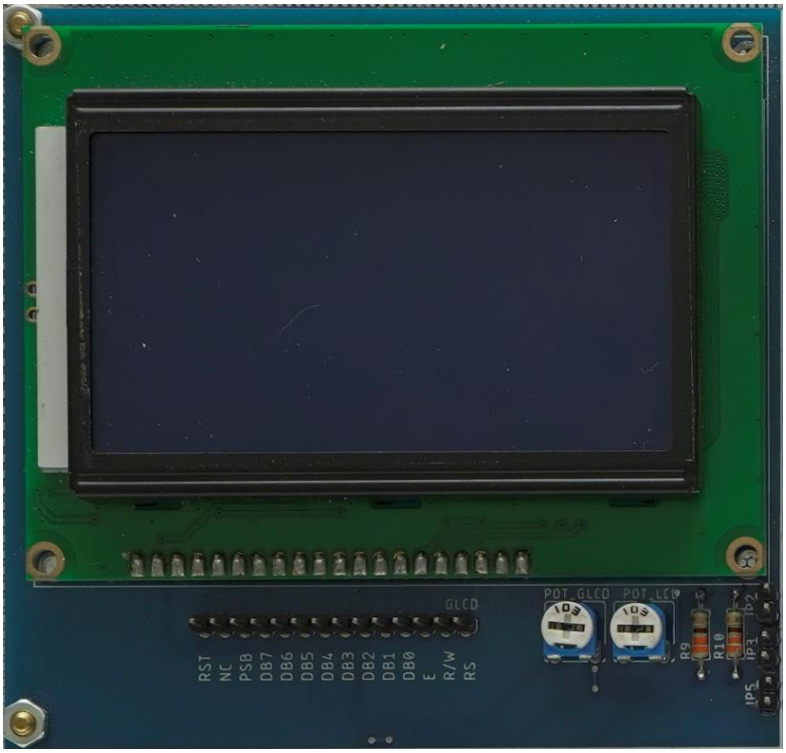


Figura 36. *Estructura metálica del módulo.*

3.6. Bloque del LCD

El bloque Lcd o pantalla monocromática está constituido de 20 pixeles en el eje x y 4 pixeles en el eje y, permitiendo escribir mensajes alfanuméricos de acuerdo con la tarea a realizar en la guía de práctica, se muestra el esquemático y la Pcb con el periférico en la figura 36.

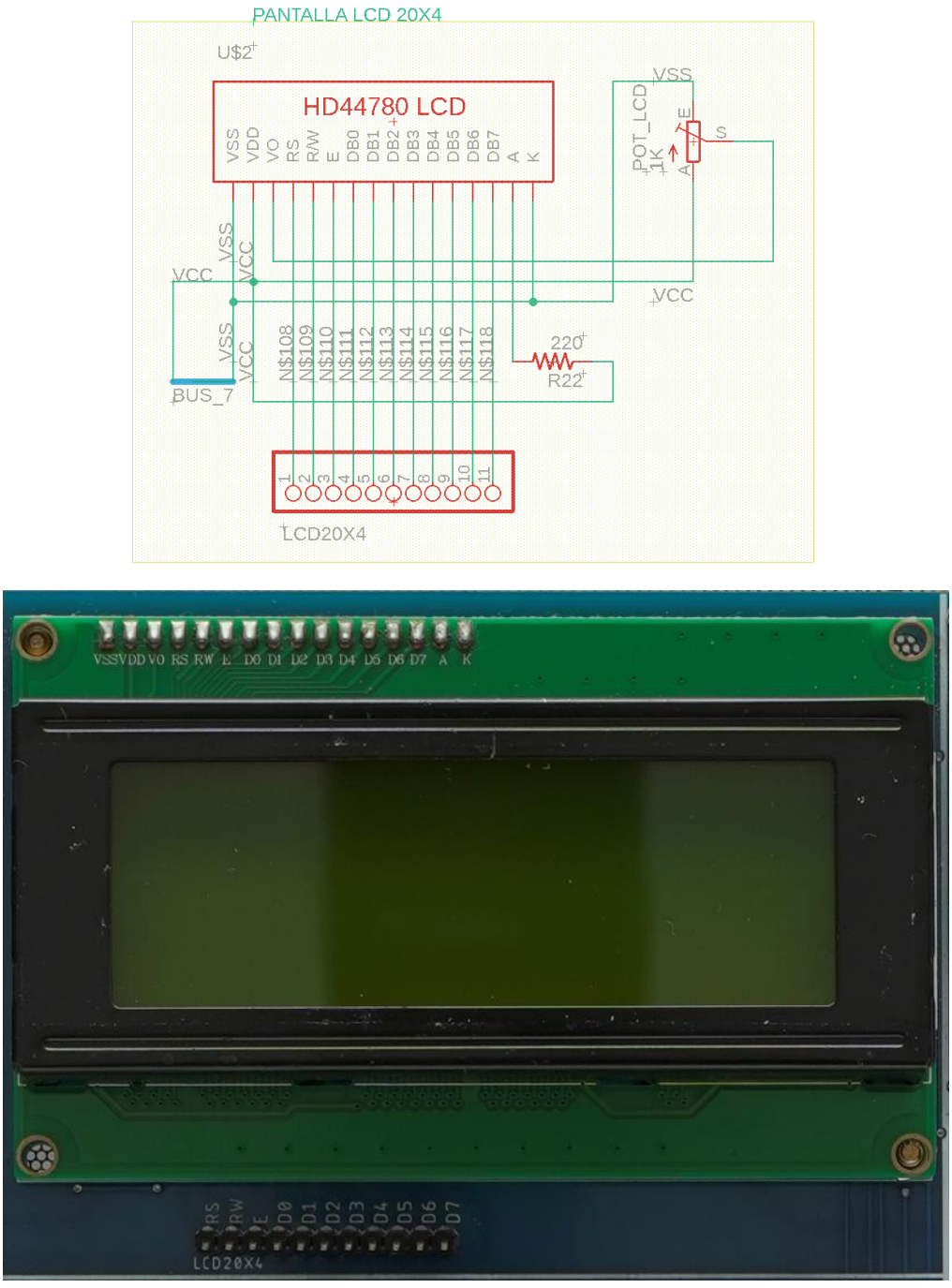


Figura 37. *Bloque del LCD.*

3.7. Bloques del pull up

El bloque de pull up está diseñado que cuando configuramos una entrada digital con una resistencia, estamos asegurando, que en todo momento vamos a tener una señal ALTA hasta el momento en el que se produzca una pulsación del interruptor, en este momento la entrada digital quedará conectada directamente a masa, o lo que es lo mismo, a un nivel de tensión 0 como se muestra en la figura 22.

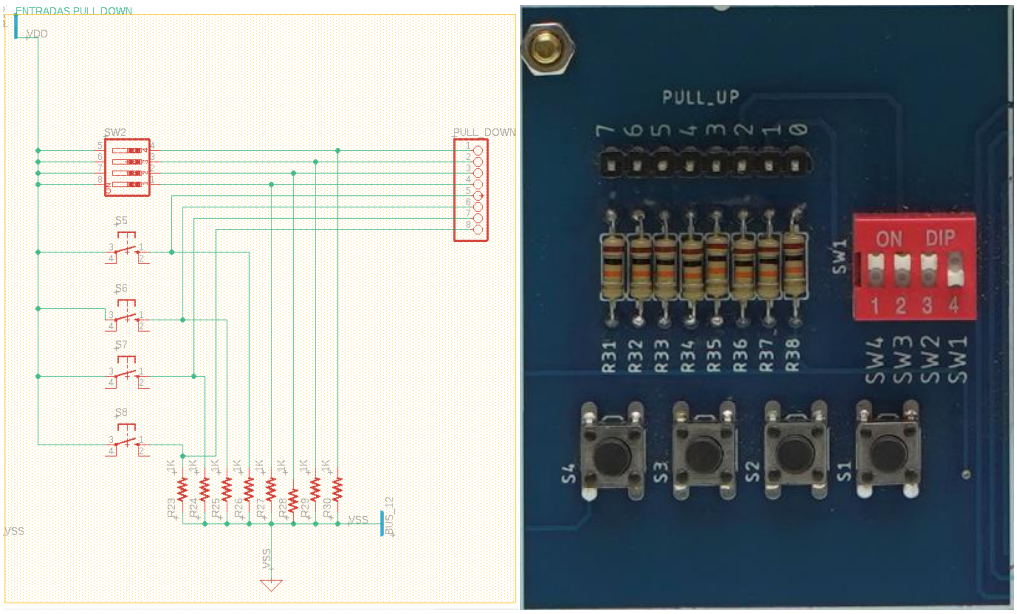


Figura 38. *Bloques del pull up.*

3.8. Bloques del pull down

El bloque de pull Down está diseñado que cuando configuramos una entrada digital con una resistencia, estamos asegurando, que en todo momento vamos a tener una señal baja hasta el momento en el que se produzca una pulsación del interruptor, en este momento la entrada digital quedará conectada directamente a masa, o lo que es lo mismo, a un nivel de tensión 1 como se muestra en la figura 39.

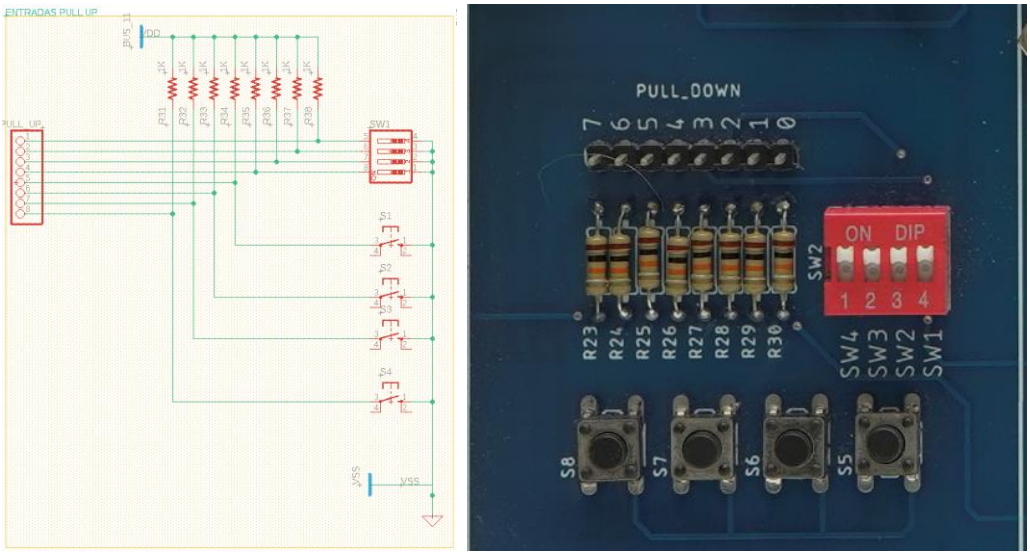


Figura 39. *Bloques del pull Down.*

3.9. Bloque de indicadores led

El bloque de indicadores sirve como visualización del estado en alto y bajo mediante led de 3.3 V como se muestra en la figura 40.

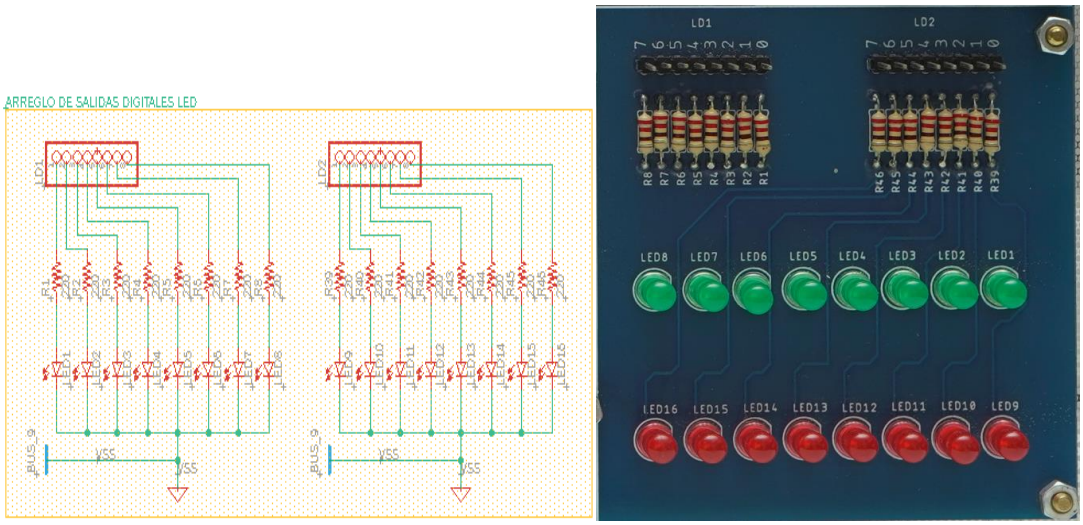


Figura 40*. Bloque de indicadores.*

3.10. Bloque de alimentación

El bloque de alimentación es el encargado de distribuir el voltaje a los demás periféricos tiene una protección a corto mediante un módulo step Down como se muestra en la figura 41.

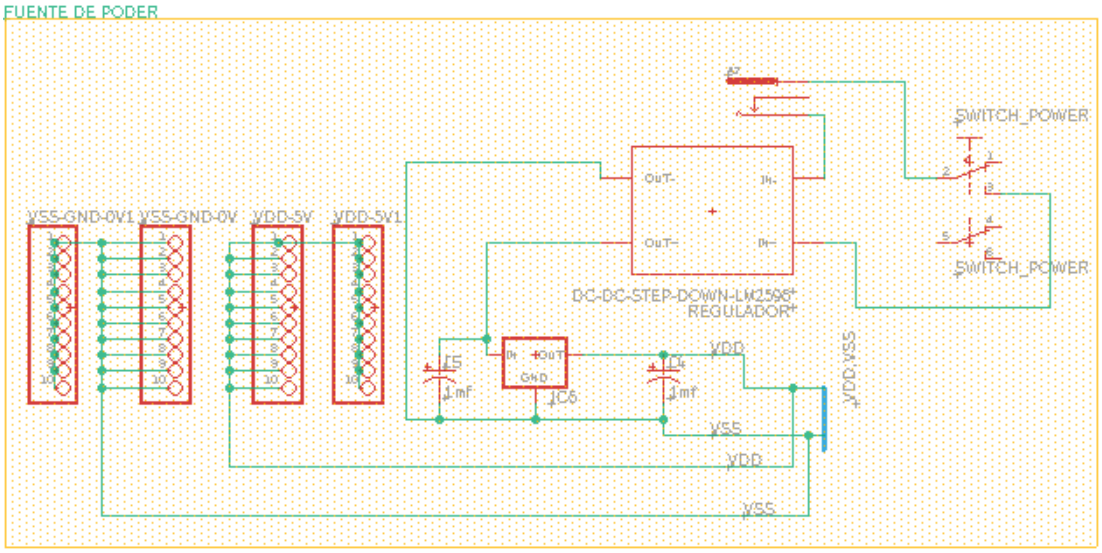


Figura 41. *Bloque de alimentación.*

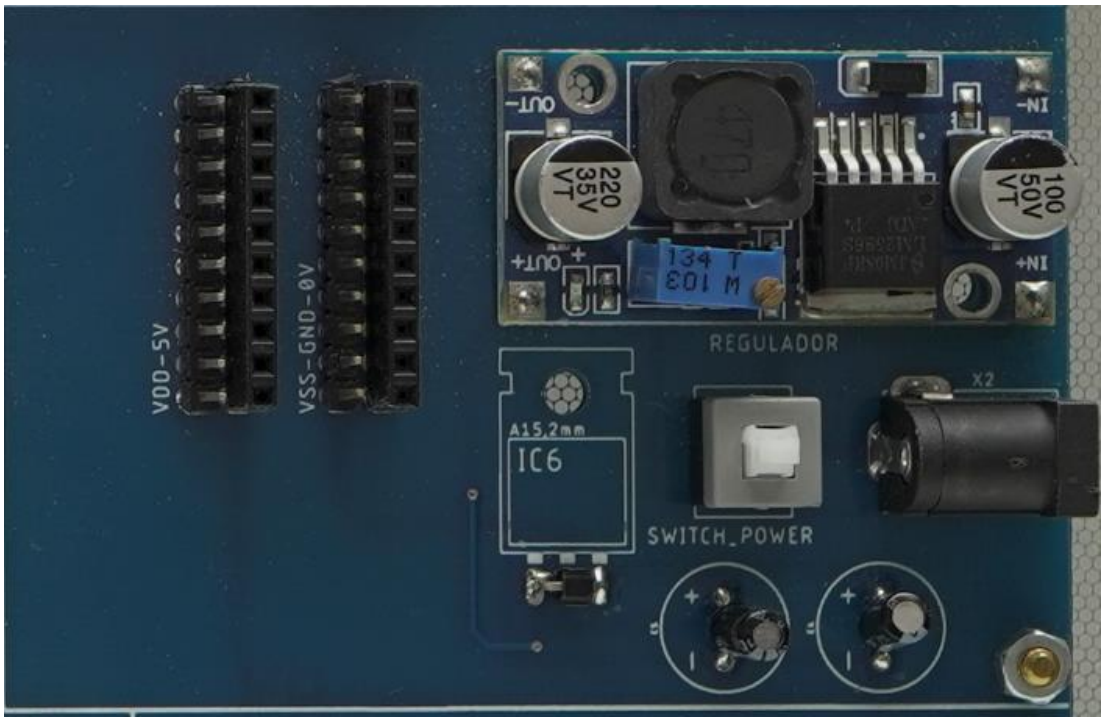


Figura 42. *Bloque de alimentación.*

3.11. Bloque del bootloader

El bloque del bootloader es un firmware para permitir la rápida descarga de programas en los microcontroladores. En el caso de los PIC, el bootloader permite descargar programas directamente desde el PC sin necesidad de utilizar ningún tipo de grabador como se muestra en la figura 43.

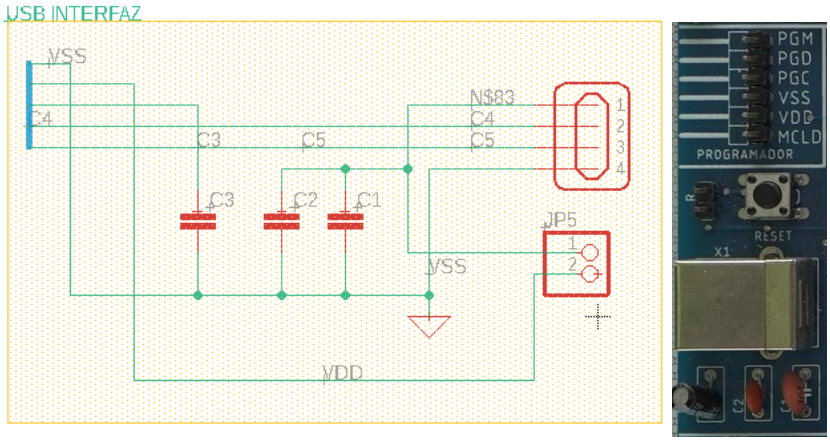


Figura 43. *Bloque del bootloader.*

3.12. Bloque de matrices led 8x8

El bloque de display matrices led está compuesto de tres arreglos de leds que sirven para la visualización de palabras, números o figuras mediante la multiplexación por un registro de desplazamiento 4094 teniendo como señal los datos de un microcontrolador como se muestra en la figura 44.

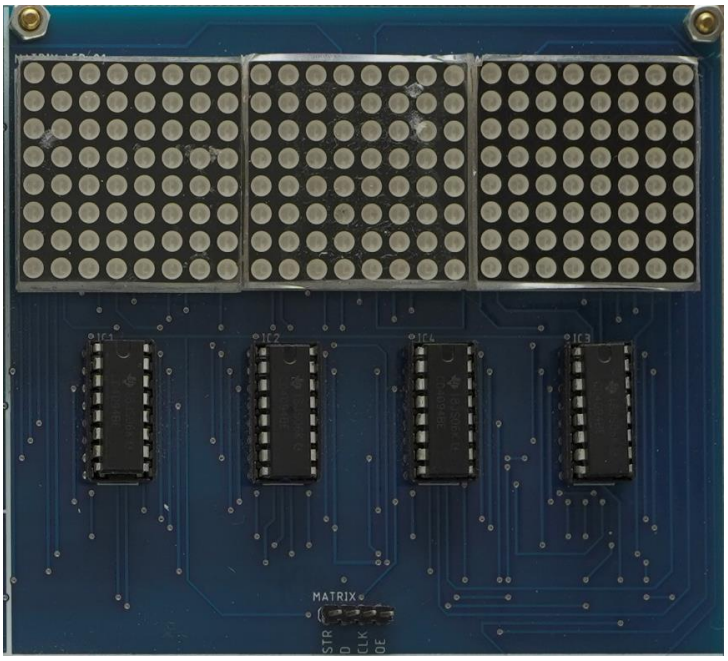


Figura 44. *Bloque de matrices led 8x8.*

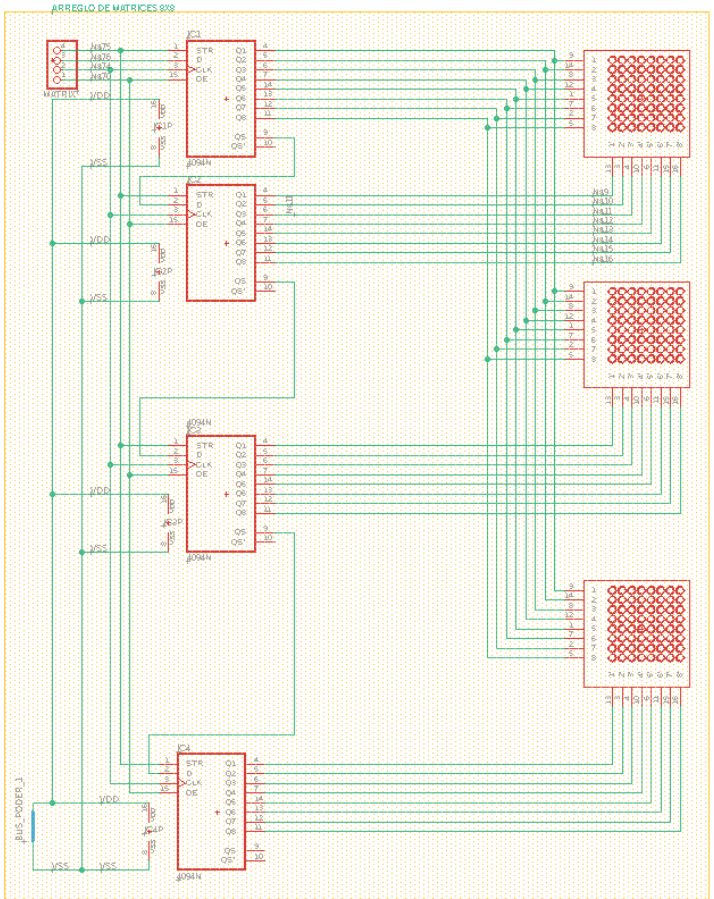


Figura 45. *Bloque de matrices led 8x8.*

3.13. Bloque del display de 7 segmentos

El bloque de display de 7 segmentos este compuesto de tres indicadores numéricos conectados a un conversor de binario a código bcd, teniendo cada display una conexión por el cátodo a un integrado 2n3906 para realizar la multiplexación de los valores mediante un controlador como se muestra en la figura 44.

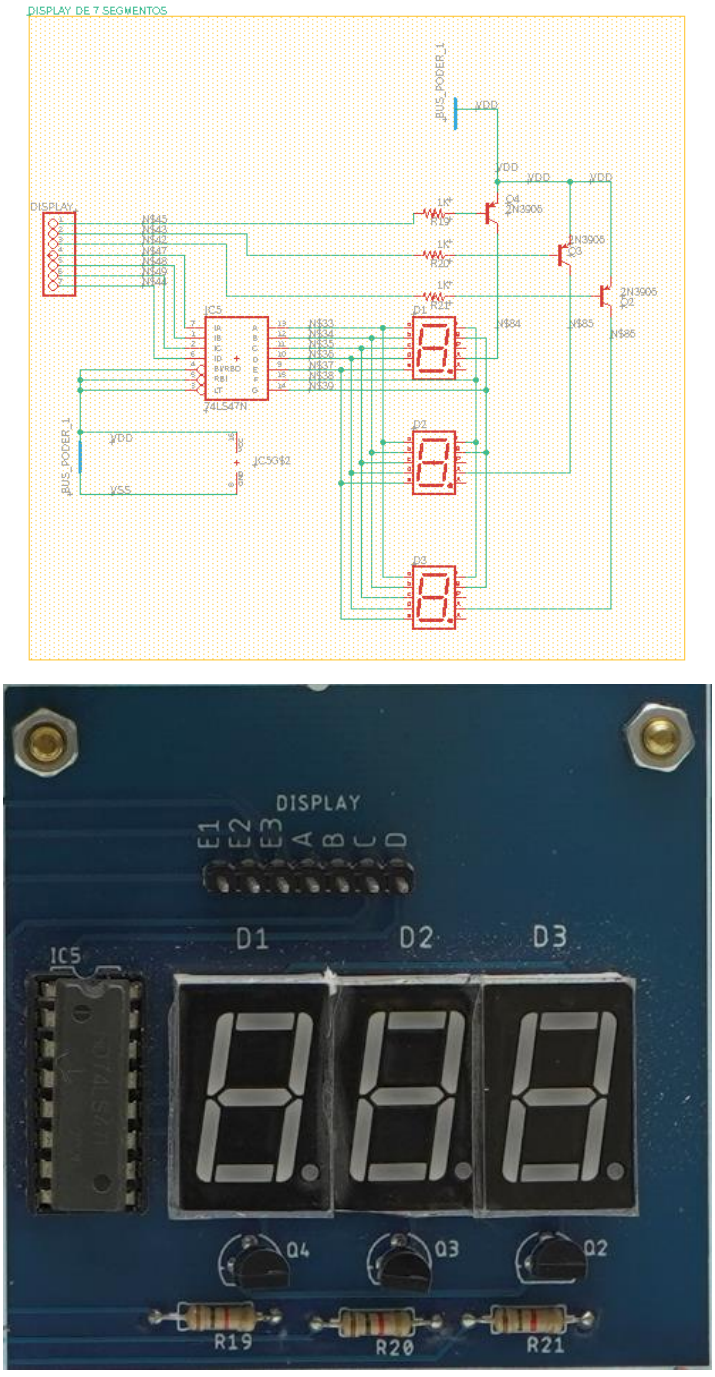


Figura 46. *Bloque del display de 7 segmentos.*

3.14. Bloque de entradas analógicas

El bloque de entradas analógicas consta de 6 resistencias variables para los requerimientos necesarios de cada PRÁCTICA y posibles tareas a futuro como se muestra en la figura 47.

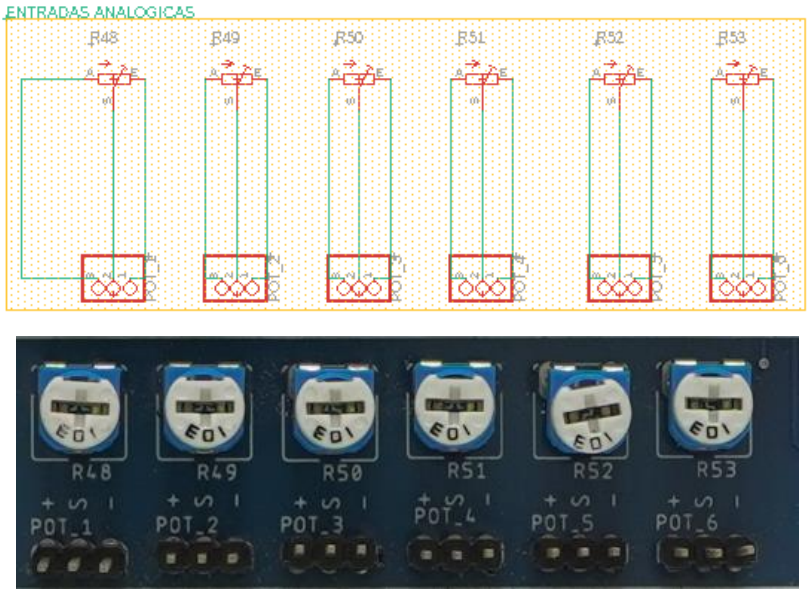


Figura 47. *Bloque entradas analógicas.*

3.15. Bloque de comunicación

El bloque de comunicación cuenta con tres protocolos los cuales tienen sus pines en el microcontrolador para las diversas prácticas.

En la figura 48 se muestra el protocolo i2c con sus respectivas resistencias de pull up.

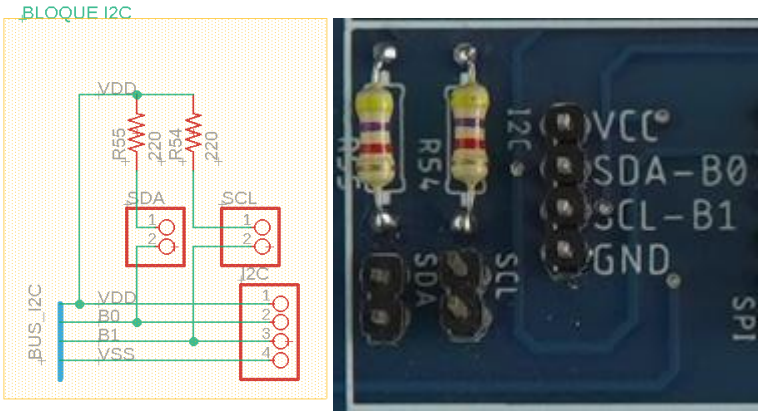


Figura 48. *Protocolo i2c.*

En la figura 48 se muestra el protocolo Spi con sus respectivas resistencias de pull up.

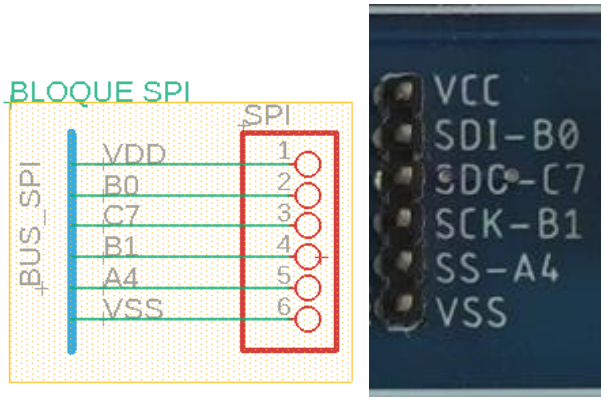


Figura 49. *Protocolo SPI.*

En la figura 49 se muestra el protocolo UART con sus respectivas resistencias de pull up.

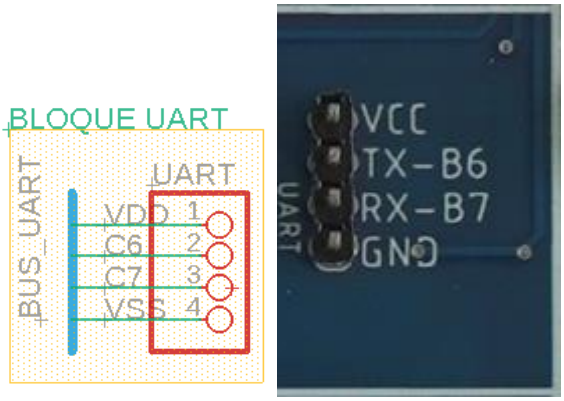


Figura 50. *Protocolo UART.*

4. MANUAL DE PRÁCTICAS

Práctica#1: Control del encendido y apagado de Leds mediante botones utilizando el PIC18F4550

<table>
<tr><td colspan="4">GUÍA DE PRÁCTICA DE LABORATORIO</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td colspan="2">ASIGNATURA: Microcontroladores</td></tr>
<tr><td>No.:</td><td>1</td><td colspan="2">TÍTULO: Control del encendido y apagado de Leds mediante botones utilizando el PIC18F4550</td></tr>
<tr><td colspan="4">OBJETIVOS:
Realizar las conexiones del módulo didáctico entre los bloques de microcontroladores con los de entradas y salidas digitales.
Utilizar el comando tris para la asignación de entrada o salidas en un puerto.
Utilizar el comando port para declarar un bit o puerto con un estado en alto o bajo.
Control del encendido y apagado de Leds mediante botones utilizando el PIC18F4550</td></tr>
<tr><td colspan="2" rowspan="3">INSTRUCCIONES:</td><td colspan="2">1. Analizar la solución propuesta de la práctica#1 ubicada en los anexos de la memoria técnica.</td></tr>
<tr><td colspan="2">2. Conectar el módulo a una fuente de alimentación de 5-12 VDC</td></tr>
<tr><td colspan="2">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="4">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="4">1. Definir variables y realizar el direccionamiento correcto.</td></tr>
<tr><td colspan="4">2. Elaborar la programación para la verificación de entradas y salidas analógicas.</td></tr>
<tr><td colspan="4">3. Realizar el cableado respectivo de entradas y salidas utilizando pulsadores y luces pilotos.</td></tr>
<tr><td colspan="4">RESULTADOS:
Desarrollo del programa en el compilador en lenguaje C para el desarrollo de la aplicación planteada.
Comprender el manejo de puertos y sus respectivas configuraciones.</td></tr>
<tr><td colspan="4">CONCLUSIONES:
Al implementar un bit como entrada o salida digital es necesario el uso del comando tris.
Para la escritura en un puerto determinado se puede usar el comando port.</td></tr>
<tr><td colspan="4">RECOMENDACIONES:
Revisar las conexiones de los bloques de entrada con el microcontrolador.
Considerar la configuración del puerto asignado como entrada o salida en el módulo didáctico.</td></tr>
</table>

***Docente*: ING. BREMNEN MARINO VELIZ NOBOA Msc.**

Firma: **Resolución CS N° 076-04-2016-04-20**

Práctica#2: Diseño de una calculadora grafica utilizando el LCD, Teclado Matricial y PIC18F4550

<table>
<tr><td colspan="4">UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR
GUÍA DE PRÁCTICA DE LABORATORIO</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td colspan="2">ASIGNATURA: Microcontroladores</td></tr>
<tr><td>No.:</td><td>2</td><td colspan="2">TÍTULO: Diseño de una calculadora grafica utilizando el LCD, Teclado Matricial y PIC18F4550</td></tr>
<tr><td colspan="4">OBJETIVOS:
Realizar las conexiones del módulo didáctico entre los bloques de microcontroladores con el bloque de LCD y TECLADO.
Utilizar librería para el manejo de una pantalla Lcd de 20 x 4.
Utilizar librería para el manejo de un teclado matricial de 4x4.
Diseñar una calculadora grafica utilizando el LCD, Teclado Matricial y PIC18F4550</td></tr>
<tr><td colspan="2" rowspan="3">INSTRUCCIONES:</td><td colspan="2">1. Analizar la solución propuesta de la práctica#2 ubicada en los anexos de la memoria técnica.</td></tr>
<tr><td colspan="2">2. Conectar el módulo a una fuente de alimentación de 5 VDC</td></tr>
<tr><td colspan="2">3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 2.</td></tr>
<tr><td colspan="4">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="4">1. Definir variables y realizar el direccionamiento correcto.</td></tr>
<tr><td colspan="4">2. Elaborar la programación para la verificación de entradas y salidas analógicas.</td></tr>
<tr><td colspan="4">3. Realizar el cableado respectivo de entradas y salidas utilizando pulsadores y luces pilotos.</td></tr>
<tr><td colspan="4">RESULTADOS:
Desarrollo del programa en el compilador en lenguaje C para el desarrollo de la aplicación planteada.
Aplicación para el entendimiento del correcto uso de librerías y subclases</td></tr>
<tr><td colspan="4">CONCLUSIONES:
Mediante el uso de librerías se logra obtener un resultado más simplificado y menos código ahorrando memoria en el microcontrolador.
Al momento de usar subclases o sub-métodos ayuda a un programa más amigable.</td></tr>
<tr><td colspan="4">RECOMENDACIONES:
Al utilizar el módulo es necesario revisar el esquemático de la práctica para su correcta conexión entre los bloques del controlador, Lcd y teclado.
Considerar un correcto suministro en la fuente de alimentación desde 5 VDC a 12 VDC</td></tr>
</table>

***Docente*: ING. BREMNEN MARINO VELIZ NOBOA Msc.**

Firma: **Resolución CS N° 076-04-2016-04-20**

Práctica#3: Contador ascendente y descendente utilizando los display de 7 segmentos y botones.

<table>
<tr><td colspan="4">UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR
GUÍA DE PRÁCTICA DE LABORATORIO</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td colspan="2">ASIGNATURA: Microcontroladores</td></tr>
<tr><td>No.:</td><td>3</td><td colspan="2">TÍTULO: Contador ascendente y descendente utilizando los display de 7 segmentos y botones.</td></tr>
<tr><td colspan="4">OBJETIVOS:
• Implementar un aplicativo para la lectura de dos entradas digitales.
• - Controlar el estado de un Display de 7 segmentos mediante una variable en el código.</td></tr>
<tr><td colspan="2" rowspan="3">INSTRUCCIONES:</td><td colspan="2">1. Analizar la solución propuesta de la práctica#3 ubicada en los anexos de la memoria técnica.</td></tr>
<tr><td colspan="2">2. Conectar el módulo a una fuente de alimentación de 5 VDC</td></tr>
<tr><td colspan="2">3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 3.</td></tr>
<tr><td colspan="4">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="4">1. Definir variables y declarar las librerías necesarias para el aplicativo.</td></tr>
<tr><td colspan="4">2. Elaborar la programación para el control de un contador ascendente y descendente de entradas y salidas digitales.</td></tr>
<tr><td colspan="4">3. Realizar el cableado respectivo de entradas y salidas utilizando pulsadores y un display de 7 segmentos.</td></tr>
<tr><td colspan="4">RESULTADOS:
• Desarrollo del programa en el compilador en lenguaje C para el desarrollo de la aplicación planteada.
• - Aplicación para el incremento y decremento de un valor y visualización en un display de 7 segmentos.</td></tr>
<tr><td colspan="4">CONCLUSIONES:
• Utilizar un conversor de código bcd para controlar el display de 7 segmentos ayuda a realizar un código más optimizado.
• - Al momento de usar subclases o sub-métodos ayuda a un programa más amigable.</td></tr>
<tr><td colspan="4">RECOMENDACIONES:
• Al utilizar el módulo es necesario revisar el esquemático de la práctica para su correcta conexión entre los bloques del controlador, bloque de pulsadores y el display de 7 segmentos
• Considerar un correcto suministro en la fuente de alimentación desde 5 VDC a 12 VDC</td></tr>
</table>

***Docente*: ING. BREMNEN MARINO VELIZ NOBOA Msc.**

Firma:

Resolución CS N° 076-04-2016-04-20

Práctica#4: Detector de distancia mediante el Sensor Ultrasónico visualizando los datos en un LCD.

<table>
<tr><td colspan="3">GUÍA DE PRÁCTICA DE LABORATORIO</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td>ASIGNATURA: Microcontroladores</td></tr>
<tr><td>No.:</td><td>4</td><td>TÍTULO: Detector de distancia mediante el Sensor Ultrasónico visualizando los datos en un LCD.</td></tr>
<tr><td colspan="3">OBJETIVOS:
Implementar un aplicativo para el acondicionamiento de un sensor de distancia.
Diseñar una interfaz visual donde se muestre el valor de distancia.</td></tr>
<tr><td rowspan="3">INSTRUCCIONES:</td><td colspan="2">1. Analizar la solución propuesta de la práctica#4 ubicada en los anexos de la memoria técnica.</td></tr>
<tr><td colspan="2">2. Conectar el módulo a una fuente de alimentación de 5 VDC</td></tr>
<tr><td colspan="2">3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 4.</td></tr>
<tr><td colspan="3">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="3">1. Definir variables y realizar el direccionamiento correcto.</td></tr>
<tr><td colspan="3">2. Elaborar la programación para el acondicionamiento de un sensor ultrasónico.</td></tr>
<tr><td colspan="3">3. Realizar el cableado respectivo de entradas y salidas utilizando pulsadores y luces pilotos.</td></tr>
<tr><td colspan="3">RESULTADOS:
Desarrollo del programa en el compilador en lenguaje C para el desarrollo de la aplicación planteada.
Acondicionamiento de una señal de ancho pulso de un sensor ultrasónico a un valor coma flotante.</td></tr>
<tr><td colspan="3">CONCLUSIONES:
Para el acondicionamiento de un sensor ultrasónico es necesario la utilización de la librería math.
Al momento de usar subclases o sub-métodos ayuda a un programa más amigable.</td></tr>
<tr><td colspan="3">RECOMENDACIONES:
Utilizar función math para el cálculo del valor de distancia del sensor ultrasónico.
Considerar un correcto suministro en la fuente de alimentación desde 5 VDC a 12 VDC</td></tr>
</table>

***Docente*: ING. BREMNEN MARINO VELIZ NOBOA Msc.**

Firma: **Resolución CS N° 076-04-2016-04-20**

Práctica#5: Conexiones entre 2 PIC 18F4550

<table>
<tr><td colspan="3">GUÍA DE PRÁCTICA DE LABORATORIO</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td>ASIGNATURA: Microcontroladores</td></tr>
<tr><td>No.:</td><td>5</td><td>TÍTULO: Conexiones entre 2 PIC 18F4550</td></tr>
<tr><td colspan="3">OBJETIVOS:
Implementar un aplicativo para el envío y recepción de datos entre dos microcontroladores 18f4550.</td></tr>
<tr><td rowspan="3">INSTRUCCIONES:</td><td colspan="2">1. Analizar la solución propuesta de la práctica#6 ubicada en los anexos de la memoria técnica.</td></tr>
<tr><td colspan="2">2. Conectar el módulo a una fuente de alimentación de 5 VDC</td></tr>
<tr><td colspan="2">3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 6.</td></tr>
<tr><td colspan="3">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="3">1. Definir variables para él envió de datos desde un microcontrolador a otro</td></tr>
<tr><td colspan="3">2. Elaborar el programa de recepción de datos mediante protocolo UART</td></tr>
<tr><td colspan="3">3. Realizar el cableado respectivo entre los dos módulos didácticos</td></tr>
<tr><td colspan="3">RESULTADOS:
Desarrollo del programa en el compilador en lenguaje C para el desarrollo de la aplicación planteada.
Desarrollo de aplicación para la comunicación entre dos microcontroladores.</td></tr>
<tr><td colspan="3">CONCLUSIONES:
Mediante el uso del puerto de comunicación UART es necesario declarar los pines a usar con su respectiva velocidad.
Al momento de usar subclases o sub-métodos ayuda a un programa más amigable.</td></tr>
<tr><td colspan="3">RECOMENDACIONES:
Revisar el cableado entre los dos microcontroladores sea cruzado receptor con emisor y emisor con receptor
Revisar que ambos circuitos o módulos didácticos estén alimentados a la fuente de poder.</td></tr>
</table>

***Docente*: ING. BREMNEN MARINO VELIZ NOBOA Msc.**

Firma:

Resolución CS N° 076-04-2016-04-20

Práctica#6: Conexión mediante el PIC18f4550 y la PC.

<table>
<tr><td colspan="3">GUÍA DE PRÁCTICA DE LABORATORIO</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td>ASIGNATURA: Microcontroladores</td></tr>
<tr><td>No.:</td><td>6</td><td>TÍTULO: Conexión mediante el PIC18f4550 y la PC.</td></tr>
<tr><td colspan="3">OBJETIVOS:
Implementar una conexión entre el Pic 18f4550 y la PC
Utilizar la comunicación USB del Pic 18f4550.</td></tr>
<tr><td rowspan="3">INSTRUCCIONES:</td><td colspan="2">1. Analizar la solución propuesta de la práctica#6 ubicada en los anexos de la memoria técnica.</td></tr>
<tr><td colspan="2">2. Conectar el módulo a una fuente de alimentación de 5 VDC</td></tr>
<tr><td colspan="2">3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 6.</td></tr>
<tr><td colspan="3">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="3">1. Definir el correcto uso de las librerías necesarias para el uso del USB del picf4550.</td></tr>
<tr><td colspan="3">2. Lograr la comunicación entre el pic18f4550 y el computador mediante un puerto serial.</td></tr>
<tr><td colspan="3">3. Realizar el cableado con el módulo didáctico.</td></tr>
<tr><td colspan="3">RESULTADOS:
Realizar la conexión mediante el computador con el Pic 18f4550 como un puerto serial.
Enviar datos del pic18f4550 y el pc</td></tr>
<tr><td colspan="3">CONCLUSIONES:
Al utilizar el Pic 18f4550 con comunicación USB es necesario usar las librerías "ex_usb_common.h" incluidas el Pic Compiler
Para él envió de datos es necesario verificar la velocidad de transmisión de los datos en el buffer.</td></tr>
<tr><td colspan="3">RECOMENDACIONES:
Revisar el estado del cable de datos del puerto de comunicación del pic18f4550
Revisar que el pin de habilitación del controlador este en bajo para que el controlador establezca comunicación con el pc</td></tr>
</table>

***Docente*: ING. BREMNEN MARINO VELIZ NOBOA Msc.**

Firma:

Práctica#7: Interrupciones utilizando botones y señales analógicas mediante el PIC18F4550.

<table>
<tr><td colspan="4">GUÍA DE PRÁCTICA DE LABORATORIO</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td colspan="2">ASIGNATURA: Microcontroladores</td></tr>
<tr><td>No.:</td><td>7</td><td colspan="2">TÍTULO: Interrupciones utilizando botones y señales analógicas mediante el PIC18F4550</td></tr>
<tr><td colspan="4">OBJETIVOS:
Implementar un código en lenguaje C utilizando la interrupción externa RB0
Realizar la conversión analógica y escritura del valor en un puerto del microcontrolador cuando este activa la interrupción.</td></tr>
<tr><td colspan="2" rowspan="3">INSTRUCCIONES:</td><td colspan="2">1. Analizar la solución propuesta de la práctica#7 ubicada en los anexos de la memoria técnica.</td></tr>
<tr><td colspan="2">2. Conectar el módulo a una fuente de alimentación de 5 VDC</td></tr>
<tr><td colspan="2">3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 7</td></tr>
<tr><td colspan="4">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="4">1. Declarar los métodos para utilizar interrupciones externas en el 18f4550.</td></tr>
<tr><td colspan="4">2. Elaborar la programación que permita la lectura de un valor analógico mediante conversión analógica a digital.</td></tr>
<tr><td colspan="4">3. Realizar el cableado respectivo entre los bloques de pull up, bloques de led y el microcontrolador</td></tr>
<tr><td colspan="4">RESULTADOS:
Conversión analógica a digital y visualización en un puerto del microcontrolador
Activar el método o función declarado para la interrupción externa en RB0</td></tr>
<tr><td colspan="4">CONCLUSIONES:
Al utilizar una interrupción declarar el tipo de interrupción y la transición para la activación si es de un bit en alto o bajo.
Para la utilización del conversor se utiliza un tamaño de bit que tendrá el conversor si es de 8 bit es 256 datos o 10 bits 1024 datos.</td></tr>
<tr><td colspan="4">RECOMENDACIONES:
Declara correctamente la velocidad del cristal de Quarzo para la lectura de la interrupción
Conectar correctamente la entrada analógica en el puerto A y declarar el número de bits a utilizar</td></tr>
</table>

***Docente*: ING. BREMNEN MARINO VELIZ NOBOA Msc.**

Firma:

Resolución CS N° 076-04-2016-04-20

Práctica#8: Conexiones entre 2 PIC 18F4550 mediante comunicación I2C.

<table>
<tr><td colspan="3">GUÍA DE PRÁCTICA DE LABORATORIO</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td>ASIGNATURA: Microcontroladores</td></tr>
<tr><td>No.:</td><td>8</td><td>TÍTULO: Conexiones entre 2 PIC 18F4550 mediante comunicación I2C</td></tr>
<tr><td colspan="3">OBJETIVOS:
Implementar un aplicativo para el envío y recepción de datos entre dos microcontroladores 18f4550 por protocolo i2c
Programar un microcontrolador como esclavo que recepta los valores del maestro.
Elaborar el programa de transmisión de datos de un microcontrolador a otro mediante protocolo i2c
Realizar el cableado respectivo entre los dos módulos didácticos.</td></tr>
<tr><td colspan="2" rowspan="3">INSTRUCCIONES:</td><td>1. Analizar la solución propuesta de la práctica#7 ubicada en los anexos de la memoria técnica.</td></tr>
<tr><td>2. Conectar el módulo a una fuente de alimentación de 5 VDC</td></tr>
<tr><td>3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 7.</td></tr>
<tr><td colspan="3">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="3">1. Elaborar un programa para la transmisión de valores por I2C como maestro.</td></tr>
<tr><td colspan="3">2. Elaborar un programa para la recepción de valores por I2C como esclavo</td></tr>
<tr><td colspan="3">3. Realizar el cableado respectivo de entradas y salidas utilizando pulsadores y luces pilotos.</td></tr>
<tr><td colspan="3">RESULTADOS:
Comunicación entre dos microcontroladores por I2C
recepción de datos por i2c y posterior escritura en un puerto como salidas digitales.</td></tr>
<tr><td colspan="3">CONCLUSIONES:
Al implementar una aplicación con comunicación i2c es necesario declarar la misma velocidad de transmisión en ambos puntos.
Para el uso de la aplicación se declara cual sirve como maestro y esclavo antes del método principal.</td></tr>
<tr><td colspan="3">RECOMENDACIONES:
Revisar la correcta conexión entre emisor y receptor, de manera paralela.
Configurar los pines de transmisión de acuerdo con la aplicación.</td></tr>
</table>

***Docente*: ING. BREMNEN MARINO VELIZ NOBOA Msc.**

Firma:

Práctica#9: Envío y Recepción de datos utilizando LCD 20x4, Teclado Matricial mediante el PIC18F4550.

<table>
<tr><td colspan="3">GUÍA DE PRÁCTICA DE LABORATORIO</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td>ASIGNATURA: Microcontroladores</td></tr>
<tr><td>No.:</td><td>9</td><td>TÍTULO: Envío y Recibo de datos utilizando LCD 20x4, Teclado Matricial mediante el PIC18F4550</td></tr>
<tr><td colspan="3">OBJETIVOS:
Implementar un aplicativo para el envío y recepción de datos entre dos microcontroladores 18f4550 por protocolo UART
Programar un microcontrolador que reciba valores de un teclado matricial posteriormente envíe el dato por serial.
Programar un microcontrolador que reciba valores por el uart posteriormente se visualicen en un Lcd.
Realizar el cableado respectivo entre los dos módulos didácticos.</td></tr>
<tr><td colspan="2" rowspan="3">INSTRUCCIONES:</td><td>1. Analizar la solución propuesta de la práctica#9 ubicada en los anexos de la memoria técnica.</td></tr>
<tr><td>2. Conectar el módulo a una fuente de alimentación de 5 VDC</td></tr>
<tr><td>3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 9.</td></tr>
<tr><td colspan="3">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="3">1. Elaborar un programa para la transmisión de valores por UART.</td></tr>
<tr><td colspan="3">2. Elaborar un programa para la recepción de valores y mostrar en una pantalla Lcd 4x20.</td></tr>
<tr><td colspan="3">3. Realizar el cableado respectivo de entradas y salidas utilizando pulsadores y luces pilotos.</td></tr>
<tr><td colspan="3">RESULTADOS:
Comunicación entre dos microcontroladores por UART.
Recepción de datos por uart y visualización en un Lcd 20x4</td></tr>
<tr><td colspan="3">CONCLUSIONES:
Al implementar una aplicación con comunicación UART se declara la misma velocidad en el receptor y emisor
Para el uso de la aplicación se configura el UART en los pines del puerto C6-C7</td></tr>
<tr><td colspan="3">RECOMENDACIONES:
Revisar la correcta conexión entre emisor y receptor, de manera cruzada.
Configurar los pines de transmisión de acuerdo con la aplicación.</td></tr>
</table>

***Docente*: ING. BREMNEN MARINO VELIZ NOBOA Msc.**

Firma:

Práctica#10: Envío y Recibo de datos utilizando LCD 20x4, Teclado Matricial mediante el PIC18F4550 y las interrupciones.

<table>
<tr><td colspan="3">UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR
GUÍA DE PRÁCTICA DE LABORATORIO</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td>ASIGNATURA: Microcontroladores</td></tr>
<tr><td>No.:</td><td>10</td><td>TÍTULO: Envío y Recibo de datos utilizando LCD 20x4, Teclado Matricial mediante el PIC18F4550 y las interrupciones</td></tr>
<tr><td colspan="3">OBJETIVOS:
Implementar un aplicativo para el envío y recepción de datos entre dos microcontroladores 18f4550 por protocolo UART.
Programar un microcontrolador que envié valores por serial cada vez que se ejecute una interrupción por un tiempo determinado.
Realizar un programa que reciba valores del puerto serial por una interrupción de comunicación UART.
Realizar el cableado respectivo entre los dos módulos didácticos.</td></tr>
<tr><td colspan="2" rowspan="3">INSTRUCCIONES:</td><td>1. Analizar la solución propuesta de la práctica#10 ubicada en los anexos de la memoria técnica.</td></tr>
<tr><td>2. Conectar el módulo a una fuente de alimentación de 5 VDC</td></tr>
<tr><td>3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 10.</td></tr>
<tr><td colspan="3">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="3">1. Elaborar un programa para la transmisión de valores por UART con la interrupción Timer.</td></tr>
<tr><td colspan="3">2. Elaborar un programa en el receptor mediante la interrupción del puerto UART con la interrupción de comunicación.</td></tr>
<tr><td colspan="3">3. Realizar el cableado respectivo de entradas y salidas utilizando pulsadores y luces pilotos.</td></tr>
<tr><td colspan="3">RESULTADOS:
Comunicación entre dos microcontroladores por UART.
Recepción de datos mediante interrupción por uart y visualización en un Lcd 20x4</td></tr>
<tr><td colspan="3">CONCLUSIONES:
Al implementar una aplicación con comunicación UART se declara la misma velocidad en el receptor y emisor
Para el uso de interrupciones de comunicación se configura el pre escalar para que en el puerto de recepción guarde el dato, cuando tenga valores en el buffer.</td></tr>
<tr><td colspan="3">RECOMENDACIONES:
Revisar la correcta conexión entre emisor y receptor, de manera cruzada.
Configurar los pines de transmisión de acuerdo con la aplicación.</td></tr>
</table>

***Docente*: ING. BREMNEN MARINO VELIZ NOBOA Msc.**

Firma: **Resolución CS N° 076-04-2016-04-20**

Práctica#11: Graficas mediante GLCD utilizando el PIC18F4550.

<table>
<tr><td colspan="3">GUÍA DE PRÁCTICA DE LABORATORIO</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td>ASIGNATURA: Microcontroladores</td></tr>
<tr><td>No.:</td><td>11</td><td>TÍTULO: Graficas mediante GLCD utilizando el PIC18F4550</td></tr>
<tr><td colspan="3">OBJETIVOS:
Implementar una guía para el uso de un GLDC de 128x64 para realizar grafica con un microcontrolador 18f4550.
Implementar la visualización de graficas o figuras geométricas básicas en un GLCD.
Realizar el cableado respectivo entre los dos módulos didácticos.</td></tr>
<tr><td rowspan="3">INSTRUCCIONES:</td><td colspan="2">1. Analizar la solución propuesta de la práctica#11 ubicada en los anexos de la memoria técnica.</td></tr>
<tr><td colspan="2">2. Conectar el módulo a una fuente de alimentación de 5 VDC</td></tr>
<tr><td colspan="2">3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 11.</td></tr>
<tr><td colspan="3">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="3">1. Elaborar un programa para el funcionamiento de un GLCD con sus diferentes librerías.</td></tr>
<tr><td colspan="3">2. Elaborar un programa que permita la visualización de figuras geométricas, texto en un GLCD</td></tr>
<tr><td colspan="3">3. Realizar el cableado respectivo de entradas y salidas entre el GLCD y el pic</td></tr>
<tr><td colspan="3">RESULTADOS:
Visualización de en el Lcd grafico de figuras geométricas.
Conexión entre un Glcd y un pic 18f4550</td></tr>
<tr><td colspan="3">CONCLUSIONES:
Al implementar una aplicación con GLCD se tiene que declarar la librería de acuerdo con el controlador ST7920(revisar anexo de librerías)
Para el uso del GLCD se debe tener en cuenta la cantidad de bits a enviar por el puerto al GLCD.</td></tr>
<tr><td colspan="3">RECOMENDACIONES:
Revisar la correcta conexión entre el GLCD y el pic.
Revisar la declaración de las librerías sea correcta</td></tr>
</table>

***Docente*: ING. BREMNEN MARINO VELIZ NOBOA Msc.**

Firma:

Práctica#12: Graficas mediante GLCD y Potenciómetro utilizando el PIC18F4550.

<table>
<tr><td colspan="4">GUÍA DE PRÁCTICA DE LABORATORIO</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td colspan="2">ASIGNATURA: Microcontroladores</td></tr>
<tr><td>No.:</td><td>12</td><td colspan="2">TÍTULO: Graficas mediante GLCD y Potenciómetro utilizando el PIC18F4550.</td></tr>
<tr><td colspan="4">OBJETIVOS:
Implementar una guía para el uso de un GLDC de 128x64 para realizar grafica con un microcontrolador 18f4550.
Implementar la visualización de una figura geométrica circular controlando el radio mediante un valor obtenido por conversión análoga a digital
Convertir un valor numérico a string utilizando el comando itoa.
Realizar el cableado respectivo entre los dos módulos didácticos.</td></tr>
<tr><td colspan="2" rowspan="3">INSTRUCCIONES:</td><td colspan="2">1. Analizar la solución propuesta de la práctica#12 ubicada en los anexos de la memoria técnica.</td></tr>
<tr><td colspan="2">2. Conectar el módulo a una fuente de alimentación de 5 VDC</td></tr>
<tr><td colspan="2">3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 12.</td></tr>
<tr><td colspan="4">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="4">1. Implementar la visualización de una figura geométrica circular controlando el radio mediante un valor obtenido por conversión análoga a digital</td></tr>
<tr><td colspan="4">2. Elaborar una conversión analógica a digital y mostrar el valor en el GLCD.</td></tr>
<tr><td colspan="4">3. Realizar el cableado respectivo de entradas y salidas entre el GLCD y el pic</td></tr>
<tr><td colspan="4">RESULTADOS:
Visualización de en el Lcd grafico controlado por una señal analógica.
conversión de un dato numérico a texto.
Conexión entre un Glcd y un pic 18f4550</td></tr>
<tr><td colspan="4">CONCLUSIONES:
Al convertir un valor numérico a texto es necesario el uso de librería string o stliod.
Al realizar una conversión análoga a digital para publicar el valor en el GLCD es necesario la conversión de datos y el uso de la librería Graphics.</td></tr>
<tr><td colspan="4">RECOMENDACIONES:
Revisar la correcta conexión entre el GLCD y el pic.
Revisar la declaración de las librerías sea correcta</td></tr>
</table>

***Docente*: ING. BREMNEN MARINO VELIZ NOBOA Msc.**

Firma:

Práctica#13: Sistema de caja fuerte utilizando GLCD, Teclado Matricial, Buzzer y PIC18f4550.

<table>
<tr><td colspan="4">UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR
GUÍA DE PRÁCTICA DE LABORATORIO</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td colspan="2">ASIGNATURA: Microcontroladores</td></tr>
<tr><td>No.:</td><td>13</td><td colspan="2">TÍTULO: Sistema de caja fuerte utilizando GLCD, Teclado Matricial, Buzzer y PIC18f4550</td></tr>
<tr><td colspan="4">OBJETIVOS:
Implementar: un aplicativo en el módulo didáctico que simule un sistema de acceso a una caja fuerte.
Diseñar un programa que mediante el uso de librerías logre enviar los datos al GLCD y al teclado separando el nivel de cómputo.</td></tr>
<tr><td colspan="2" rowspan="3">INSTRUCCIONES:</td><td colspan="2">1. Analizar la solución propuesta de la práctica#13 ubicada en los anexos de la memoria técnica.</td></tr>
<tr><td colspan="2">2. Conectar el módulo a una fuente de alimentación de 5 VDC</td></tr>
<tr><td colspan="2">3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 13.</td></tr>
<tr><td colspan="4">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="4">1. Definir variables y realizar el direccionamiento correcto.</td></tr>
<tr><td colspan="4">2. Utilizar la librería del GLCD con el controlador st7920 para enviar datos desde el microcontrolador.</td></tr>
<tr><td colspan="4">3. Realizar el cableado entre los periféricos como GLCD, teclado matricial y el pic 18f4550</td></tr>
<tr><td colspan="4">RESULTADOS:
Implementación de un sistema de seguridad básico de una caja fuerte con una visualización en una pantalla grafica GLCD.
Integración entre los periféricos con el pic18f4550</td></tr>
<tr><td colspan="4">CONCLUSIONES:
Para el uso del periférico GLCD y el teclado matricial con el pic, se declara la distribución correcta entre los pines y que no interfieran con el código entre ellos.
Declarar la velocidad de 20 MHz, ya que el nivel de cómputo es alto al usar el GLCD.</td></tr>
<tr><td colspan="4">RECOMENDACIONES:
Revisar el correcto estado de los cables para la conexión entre los módulos
Direccionar correctamente los pines de las librerías de cada periférico</td></tr>
</table>

***Docente*: ING. BREMNEN MARINO VELIZ NOBOA Msc.**

Firma:

Práctica#14: Matrices LED creando palabras, números mediante el PIC18f4550.

<table>
<tr><td colspan="4">UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR
GUÍA DE PRÁCTICA DE LABORATORIO</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td colspan="2">ASIGNATURA: Microcontroladores</td></tr>
<tr><td>No.:</td><td>14</td><td colspan="2">TÍTULO: Matrices LED creando palabras, números mediante el PIC18f4550</td></tr>
<tr><td colspan="4">OBJETIVOS:
Realizar un programa en el cual permita graficar palabras en la matriz led
Crear letras mediante arreglos con el microcontrolador.
Aprender el funcionamiento de la herramienta 8x8pixel.</td></tr>
<tr><td colspan="2" rowspan="3">INSTRUCCIONES:</td><td colspan="2">1. Analizar la solución propuesta de la práctica#14 ubicada en los anexos de la memoria técnica.</td></tr>
<tr><td colspan="2">2. Conectar el módulo a una fuente de alimentación de 5 VDC</td></tr>
<tr><td colspan="2">3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 14.</td></tr>
<tr><td colspan="4">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="4">1. Realizar un arreglo que sirva para la creación de palabras y números en la matriz led</td></tr>
<tr><td colspan="4">2. Elaborar la programación para él envió de valores mediante comunicación SPI a las matrices led.</td></tr>
<tr><td colspan="4">3. Realizar el cableado respectivo entre el microcontrolador y la matriz led</td></tr>
<tr><td colspan="4">RESULTADOS:
Entender el proceso de creación de caracteres para la matriz led mediante el programa en línea xlr8.
Envió de datos del microcontrolador a la matriz led mediante el registro de desplazamiento 4094.</td></tr>
<tr><td colspan="4">CONCLUSIONES:
El uso de herramientas como el xlr8, facilita el manejo de la matriz led, permitiendo programar de manera más rápida.
El uso de registro de desplazamientos evita saturar los pines del microcontrolador.</td></tr>
<tr><td colspan="4">RECOMENDACIONES:
Al utilizar la herramienta xlr8 tomar en cuenta la posición de la matriz led para su correcta escritura
Revisar el estado de la fuente de alimentación al módulo y sus correctas conexiones.</td></tr>
</table>

***Docente*: ING. BREMNEN MARINO VELIZ NOBOA Msc.**

Firma:

Práctica#15: Registro de desplazamiento utilizando Matrices LED mediante el PIC18F4550.

<table>
<tr><td colspan="4">UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR
GUÍA DE PRÁCTICA DE LABORATORIO</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td colspan="2">ASIGNATURA: Microcontroladores</td></tr>
<tr><td>No.:</td><td>15</td><td colspan="2">TÍTULO: Registro de desplazamiento utilizando Matrices LED mediante el PIC18F4550.</td></tr>
<tr><td colspan="4">OBJETIVOS:
Realizar un programa en el cual permita graficar palabras en la matriz led
Crear letras mediante arreglos con el microcontrolador.
Realizar una conversión análoga a digital y mostrar el valor en la matriz led</td></tr>
<tr><td colspan="2" rowspan="3">INSTRUCCIONES:</td><td colspan="2">1. Analizar la solución propuesta de la práctica#15 ubicada en los anexos de la memoria técnica.</td></tr>
<tr><td colspan="2">2. Conectar el módulo a una fuente de alimentación de 5 VDC</td></tr>
<tr><td colspan="2">3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 15.</td></tr>
<tr><td colspan="4">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="4">1. Realizar un arreglo que sirva para la creación de palabras y números en la matriz led</td></tr>
<tr><td colspan="4">2. Elaborar la programación para él envió de valores mediante comunicación SPI a las matrices led.</td></tr>
<tr><td colspan="4">3. Realizar el cableado respectivo entre el microcontrolador y la matriz led</td></tr>
<tr><td colspan="4">RESULTADOS:
Entender el proceso de creación de caracteres para la matriz led mediante el programa en línea xlr8.
Envió de datos del microcontrolador a la matriz led mediante el registro de desplazamiento 4094.
envió de valores de la conversión análoga a digital de un microcontrolador con visualización en matriz led.</td></tr>
<tr><td colspan="4">CONCLUSIONES:
El uso de herramientas como el xlr8, facilita el manejo de la matriz led, permitiendo programar de manera más rápida.
El uso de registro de desplazamientos evita saturar los pines del microcontrolador.</td></tr>
<tr><td colspan="4">RECOMENDACIONES:
Al utilizar la herramienta xlr8 tomar en cuenta la posición de la matriz led para su correcta escritura
Revisar el estado de la fuente de alimentación al módulo y sus correctas conexiones.</td></tr>
</table>

***Docente*: ING. BREMNEN MARINO VELIZ NOBOA Msc.**

Firma: **Resolución CS N° 076-04-2016-04-20**

5. RESULTADOS

5.1 Diseño y elaboración de los elementos del módulo didáctico para entrenamiento de microcontroladores PIC 18F4550

Para la respectiva realización del módulo se efectuó diversos procedimientos como la elaboración de la tarjeta Pcb en el software Eagle

En la Figura 50 se puede observar el módulo.

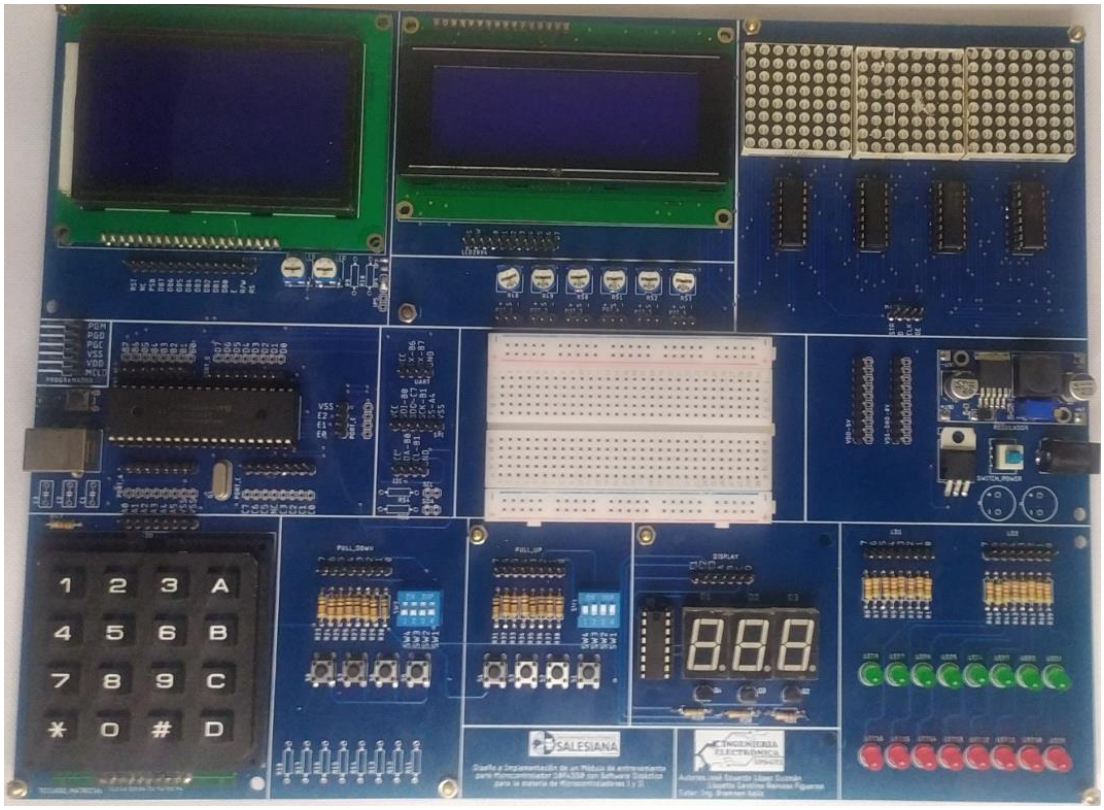


Figura 51. *Modulo didáctico.*

A continuación, detallamos los resultados PRÁCTICA por PRÁCTICA de nuestro proyecto previo a toda la implementación física donde se intervino en construcción de módulos, adecuación de láminas educativas, colocación de elementos de control eléctrico, cableado de control, programación e integración tenemos lo siguiente.

- En la primera práctica se tiene como resultado la lectura de los estados en los bits del puerto b, el valor del puerto b se escribe en el puerto d

realizando la variación entre estado altos y bajos en el módulo didáctico.

- En la segunda práctica se tiene como resultado una aplicación la cual permite al estudiante aprender el correcto uso de las librerías de la pantalla Lcd y del teclado matricial teniendo como función la de una calculadora básicas de dos dígitos.
- En la tercera práctica se tiene un aplicativo el cual tendrá la función de permitir la visualización de una variable en un display de 7 segmentos, teniendo como función el decremento o incremento de dicho valor mediante la lectura de dos variables digitales conectándose todo a un microcontrolador Pic 18f4550
- En la cuarta práctica se tiene como resultado la lectura de la distancia de un sensor ultrasónico mediante su acondicionamiento mediante un microcontrolador 18f4550 y su posterior visualización en una pantalla Lcd de 20 x4.
- En la quinta práctica se tiene como resultado la comunicación entre dos controladores Pic 18f4550 donde el primero sirve como emisor del valor de un puerto asignado y el segundo sirve como receptor del valor y se muestra en un puerto asignado.
- En la sexta práctica se tiene como resultado una aplicación la cual envía un valor entero mediante el puerto USB integrado en el microcontrolador, el valor se observa mediante el visualizador serial.
- En la séptima práctica se tiene como resultado la conversión analógica a digital de un valor de voltaje de 0 a 5 voltios mediante la división de voltaje de un potenciómetro, donde al ejecutarse el programa y tener un estímulo en el puerto RB0 se activa una interrupción externa del microcontrolador activándose una señal donde todos los valores del puerto d se encienden comunicación entre dos controladores Pic 18f4550 donde el primero sirve como emisor del valor de un puerto asignado y el segundo sirve como receptor del valor y se muestra en un puerto asignado.
- En la octava PRÁCTICA se tiene como resultado la implementación de la comunicación entre dos controladores Pic 18f4550 donde uno será

el maestro que enviará los valores digitales de entradas en el puerto D y el segundo será el esclavo sirve como esclavo receptando los valores y escribiendo los valores digitales de salidas en el puerto D.

- En la novena práctica se tiene como resultado la comunicación entre dos controladores Pic 18f4550 donde el primero envía los valores de un teclado matricial y el receptor muestra los valores en un LCD.
- En la décima práctica se tiene como resultado la comunicación entre dos controladores Pic 18f4550 utilizando dos interrupciones una para el emisor una interrupción por tiempo y otra para el receptor que es para la comunicación serial donde el primero envía los valores de un teclado matricial y el receptor muestra los valores en un LCD.
- En la onceava práctica se tiene como resultado él envió de comandos desde el microcontrolador al GLCD para lograr el entendimiento del uso de las librerías ST7920 del GLCD y un PIC 18f4550
- En la décimo segunda práctica se tiene como resultado la comunicación entre un microcontrolador Pic 18f4550 con un periférico GLCD donde el micro realizara una conversión análoga a digital y controla una figura en la pantalla grafica mostrando el valor de la conversión y el radio a dibujarse de un círculo.
- En la décima tercera práctica se diseñó un aplicativo con el módulo didáctico el cual simula el sistema de una caja fuerte con una pantalla grafica GLD, un teclado matricial y un controlador Pic 18f4550, de manera que se consulta una clave de acceso al usuario.
- En la décima cuarta práctica se una guía del uso de las herramientas 8pixel para la creación de caracteres y visualización en tres matrices led comunicándose con un controlador Pic 18f4550 .
- En la décima quinta PRÁCTICA se obtiene como resultado la visualización de una conversión análoga a digital en unas matrices led mediante la multiplexación y envió de datos por 4 registros de desplazamiento 4094 comunicándose con un controlador Pic 18f4550.

5.2 Elaboración de la programación con el Pic C Compiler

Para el proyecto se utilizó el software PIC c Compiler donde la principal ventaja de este es poder trabajar en lenguaje C permitiendo el uso de librería que son bloques de código que contienen métodos, rutinas para el funcionamiento de determinados periféricos.

Figura 52. *Programación de tarjetas*

En la elaboración de las placas se optó por realizar el soldado de los componentes a utilizar como se muestra en la figura.

Figura 53. *Programación de tarjetas*

5.3 Guía o Manual de Prácticas de Laboratorio

Como resultado de nuestro proyecto de titulación se realizó 15 prácticas didácticas, de las cuales están relacionadas al plan analítico de la materia de microcontroladores enfocada a ser utilizadas en un tiempo de 15 semanas, permitiendo la integración de la materia con la parte PRÁCTICA

Para poder visualizar más detalladas las prácticas propuestas es preciso verificar y observar la sección de anexos del documento de titulación.

CONCLUSIONES

- Mediante el desarrollo de un módulo didáctico en el software para diseño de tarjetas de circuito impreso denominado EAGLE se logra una versatilidad al momento de la creación de la Pcb que tendrá los elementos para las PRÁCTICAS planteadas declaradas en el presente documento donde se plasman los conocimientos teóricos y prácticos en un entorno controlado.

- Se desarrollo una tarjeta que contiene los elementos que sean utilizados en las materias a fines con el uso de microcontroladores utilizando el PIC18F4550 en sus respectivas cátedras.

- Se implemento láminas dedicadas para PRÁCTICAS mediante el uso de un programador Pickit 3 permitiendo la carga de archivo hexadecimales creados con el compilador pic c Compiler basado en lenguaje C.

- El módulo consta con visualizadores gráficos como el Lcd, Glcd, display, matrices led donde se pueden visualizar los resultados de las PRÁCTICAS que se detallan en el presente documento.

- Se realizó un manual de 15 prácticas para el módulo didáctico teniendo como núcleo el PIC18F4550 para complementar la cátedra de diversas asignaturas de la carrera.

RECOMENDACIONES

- Seguir los pasos descritos en los anexos para el ensamble, verificar la alimentación de la fuente antes de cualquier operación.

- En la creación de las aplicaciones tener en cuenta la declaración de los fusibles, librerías y registros de direccionamiento y el control de las variables de entrada y salida de manera organizada.

- Para la programación del microcontrolador respetar la correcta posición de los pines con el periférico compilador hexadecimal Pickit 3.

- Realizar las conexiones detalladas en el anexo de cada PRÁCTICA con su correcto desarrollo para su óptimo funcionamiento.

- Probar controladores de lazo cerrado en cátedras enfocadas al control.

REFERENCIAS BIBLIOGRAFICAS

- CADSOFT EAGLE. (2016). *CADSOFT EAGLE*. Obtenido de CADSOFT EAGLE: https://cadsoft.io/pricing/
- Cohen, a. (2010). *Diseño de Electronica Personalizada - Diseño Digital & Analogico - Sistemas Embebidos - PCB Layout*. Obtenido de ELECTROSOFT INGENIERIA: http://www.pcb.electrosoft.cl/04-articulos-circuitos-impresos-desarrollo-sistemas/01-conceptos-circuitos-impresos/conceptos-circuitos-impresos-pcb.html
- Corvo, H. S. (2021). *Arquitectura Harvard:*. Obtenido de origen, modelo, cómo funciona: https://www.lifeder.com/arquitectura-harvard/
- Digikey. (13 de sept de 2017). *Tutorial sobre ADC/DAC*. Obtenido de https://www.digikey.com/: https://www.digikey.com/es/articles/adc-dac-tutorial
- Franco Reina Rafael Christian, M. P. (2014). *Repositorio Institucional.* Obtenido de repositorio Institucional de la Universidad Politécnica Salesiana / Tesis / Grado: http://dspace.ups.edu.ec/handle/123456789/10418
- González Macías Bryan Xavier, B. E. (2018). *Dspace.ups.edu.ec.* Obtenido de Diseño e implementación del algoritmo de control para un robot balance, usando fuzzy logic en la plataforma de national instruments.: http://dspace.ups.edu.ec/handle/123456789/16009
- González, F. (2018). *Diseño y creación de tarjetas de circuitos impresos*. Obtenido de Batiburrillo - De todo un poco: https://www.batiburrillo.net/diseno-y-creacion-de-tarjetas-de-circuitos-impresos/
- Granda, C. (2015). *areatecnologia*. Obtenido de https://www.areatecnologia.com/electricidad/pulsador.html
- Hetpro. (2015). *Display 7 Segmentos ánodo y cátodo común*. Obtenido de https://hetpro-store.com/TUTORIALES/display-7-segmentos-anodo-catodo-comun
- Li, W. (2012). *Decodificador BCD 7447 y 7448*. Obtenido de https://koalab.tech/aprende/componentes/decodificador-bcd-7447-7448/
- LLAMAS, L. (2020). *QUÉ SON LAS CAPAS DE UNA PCB*. Obtenido de Ingeniería, informática y diseño: https://www.luisllamas.es/que-son-las-capas-de-una-pcb/
- Lui, L. (2016). *ENCENDER UNA MATRIZ LED CON ARDUINO Y MAX7219.* Obtenido de CÓMO FUNCIONA UN MATRIZ LED: https://www.luisllamas.es/matriz-led-arduino-max7219
- Mecafenix. (2017). *Ingeniería Mecafenix*. Obtenido de Potenciómetro ¿Que es y como funciona?: https://www.ingmecafenix.com/electronica/potenciometro/
- Microchips. (2017). *Microcontroladores*. Obtenido de INTRODUCCIÓN Y ARQUITECTURA DE MICROCONTROLADORES: https://microcontroladoressesv.wordpress.com/microcontroladores-pic-y-sus-variedades/

- Morcillo, C. G. (2017). *www.esi.uclm.es/.* Obtenido de Lógica Difusa: https://www.esi.uclm.es/www/cglez/downloads/docencia/2011_Softcomputing/LogicaDifusa.pdf
- Naylampmechatronics. (2016). *Naylampmechatronics*. Obtenido de Interfaz de usuario>Teclado matricial 4x4 - Tipo membrana: https://www.naylampmechatronics.com/interfaz-de-usuario/19-teclado-matricial-4x4-tipo-membrana.html
- naylampmechatronics. (2016). *Resistencias Pull-Up y Pull-Down*. Obtenido de Resistencias Pull-Up y Pull-Down: https://www.naylampmechatronics.com/blog/39_Resistencias-Pull-Up-y-Pull-Down.html
- NI. (2018). *national instrument*. Obtenido de national instrument: https://www.ni.com/es-cr/shop/labview.html
- Ogata, K. (2015). *Ingeniería de Control Moderna.* Editorial Prentice Hall. Obtenido de https://www.picuino.com/es/arduprog/control-ziegler-nichols.html
- Palazzesi, A. (2011). *Ucontrol.* madrid: Revista uControl.
- Perez, V. D. (2015). Implementacion de algoritmos de determinacion de rutas para el robotino de festo.
- Phipps, C. A. (2019). Variable Speed Drive Fundamentals. *The Fairmont Press,*.
- prime, P. t. (2020). *TETRIX® PRIME Dual-Control Robotics Set*. Obtenido de TETRIX® PRIME Dual-Control Robotics Set: https://www.pitsco.com/EC/Shop/TETRIX-Robotics/TETRIX-PRIME/TETRIX-PRIME-Dual-Control-Robotics-Set
- Ramos, O. R. (12 de junio de 2008). *Colección de Tesis Digitales.* Obtenido de Universidad de las Americas Puebla: http://catarina.udlap.mx/u_dl_a/tales/documentos/lmt/ramirez_r_o/
- Sánchez, L. F. (2016). Control cinemático y dinámico. *Laboratorio de Robótica Móvil y Sistemas Automatizados*.
- Santiago Sánchez-Solano Alejandro J. Cabrera, M. B. (2015). *digital.csic.es/.* Obtenido de CONTROLADORES DIFUSOS ADAPTATIVOS COMO MÓDULOS DE PROPIEDAD INTELECTUAL PARA FPGAS: https://digital.csic.es/bitstream/10261/86598/1/Controladores%20difusos.pdf
- Santiago, A. E. (2017). *Módulo de Visualización Dinámica para Señales Biomédicas Empleando una Pantalla de Cristal Líquido Graficadora.* Berlin, Heidelberg: IV Latin American Congress on Biomedical Engineering 2007.
- Sparkfun. (2018). *serial-peripheral-interface-spi*. Obtenido de Sparkfun: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all
- Teslabem. (2017). *Teslabem*. Obtenido de Fundamentos I2C – Aprende.: https://teslabem.com/nivel-intermedio/fundamentos/
- Times, T. J. (2009). Scientist claims he made Segway predecessor in '86. *The Japan Times Online. Retrieved June 18, 2009*.
- TodoElectrodo. (2013). *PIC 18F4550*. Obtenido de Arquitectura del Pic: http://todoelectrodo.blogspot.com/2013/02/pic-18f4550.html

- Villarán, A. V. (2005). *Sistema de Desarrollo para el Microcontrolador PIC18F452.* Sevilla: ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN .
- Zator. (2016). *Tecnología del PC*. Obtenido de Tecnología del PC: https://www.zator.com/Hardware/H2_5_1_1.htm

ANEXOS

Anexo 1 Dimensiones

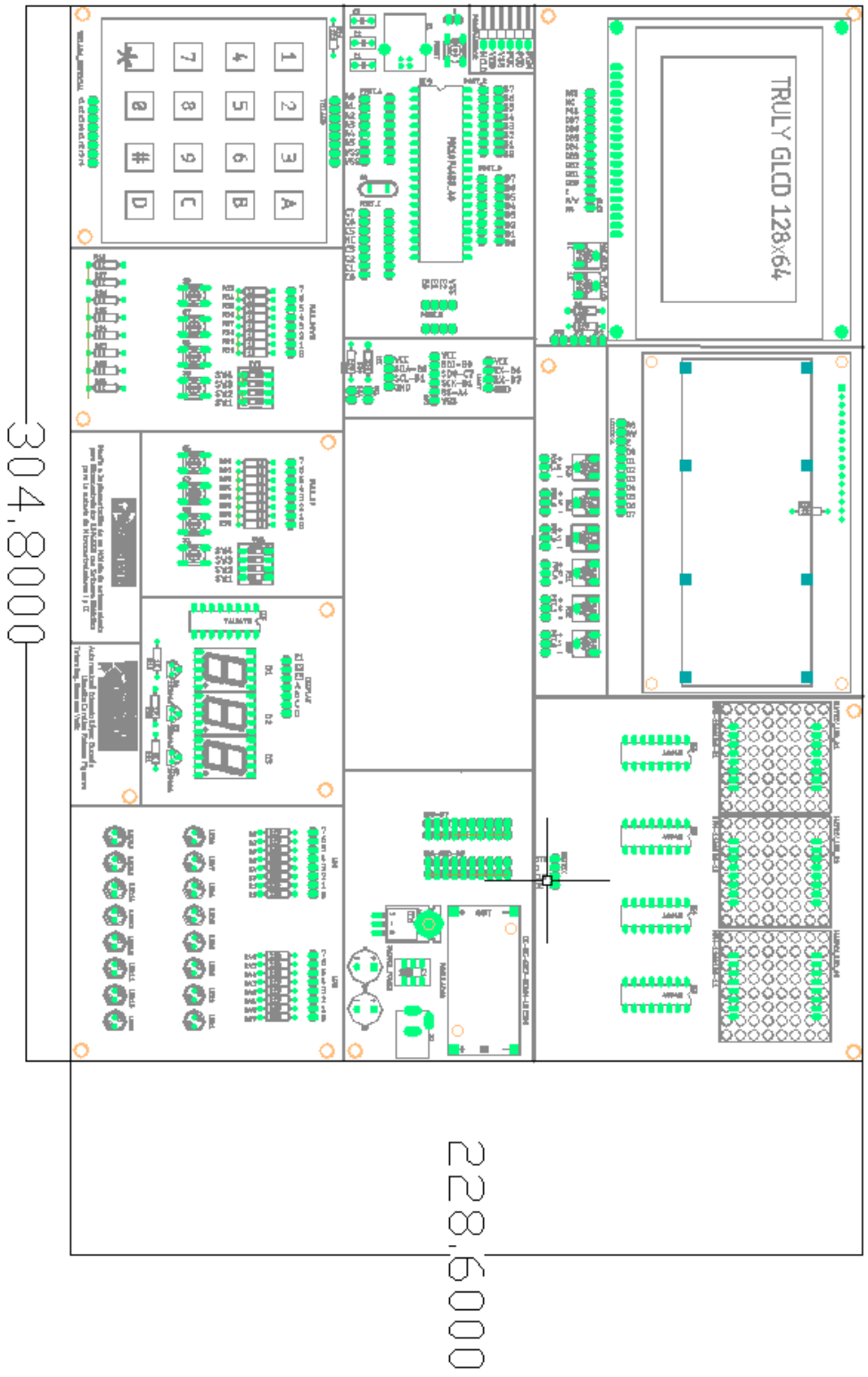


Figura 54. Anexo 1 – Dimensiones de placa en milímetros

Anexo 2 Lista de materiales

Descripción	Cantidad por modulo	Total
PIC18F4550	1	8
Teclado matricial 4x4	1	8
Glcd	1	8
Lcd	1	8
Protoboard	1	8
Led	16	128
potenciómetros	10	80
4094	4	32
Regulador de voltaje	1	8
4n3906	4	32
Display ánodo común	3	24
Matriz led	3	24
Pulsantes	9	72
Switch 4 pines	2	16
Switch on/off	1	8
Resistencias 220 ohm	20	160
Resistencias 1 Kohm	17	136
Resistencias 10 Kohm	5	40
Cristal 20Mhz	1	8
7447	1	8
USB	1	8
Pines macho	80	640
Modulo step Down	1	8
Tuercas y soportes	20	160
Programador PIC	1	8
PCB	1	8
Acrílico	1	8

Figura 55. Anexo 2 – Lista de materiales

ANEXO 3.1: RESOLUCIÓN DE PRÁCTICA 1

AUTOMATIZACIÓN INDUSTRIAL

PRÁCTICA 1

NÚMERO DE ESTUDIANTES: 20

DOCENTE

ING. BREMNEN MARINO VELIZ NOBOA MSc.

TIEMPO ESTIMADO: 2 HORAS

TEMA: "CONTROL DEL ENCENDIDO Y APAGADO DE LEDS MEDIANTE BOTONES UTILIZANDO EL PIC18F4550."

A. Objetivo General

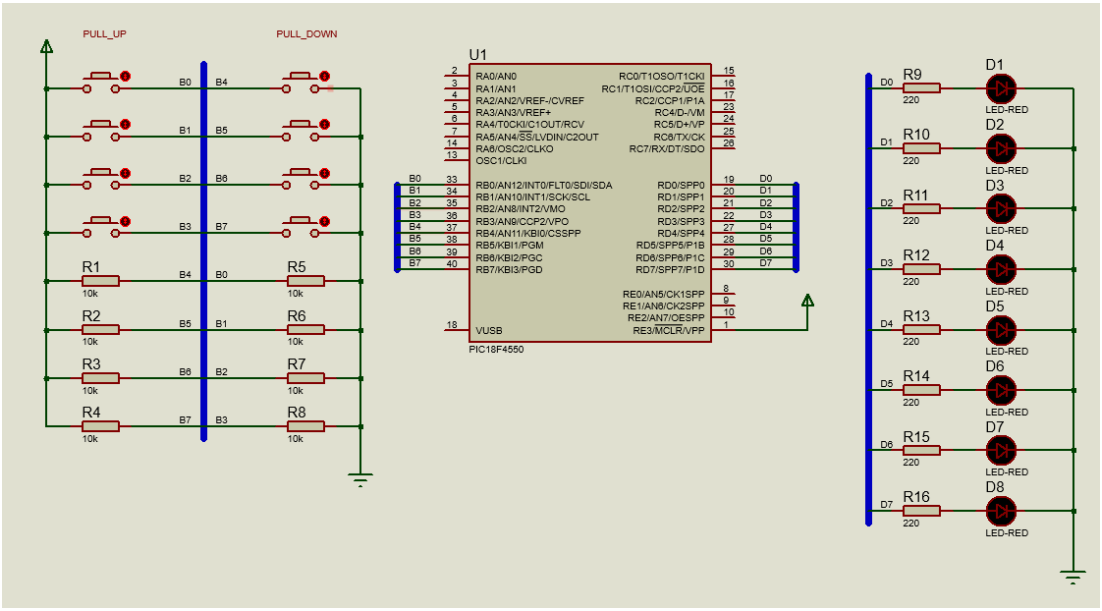
- Realizar el control del encendido y apagado de leds mediante botones utilizando el pic18f4550

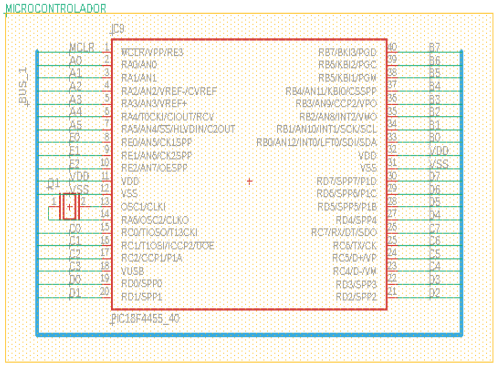
B. Objetivos Específicos

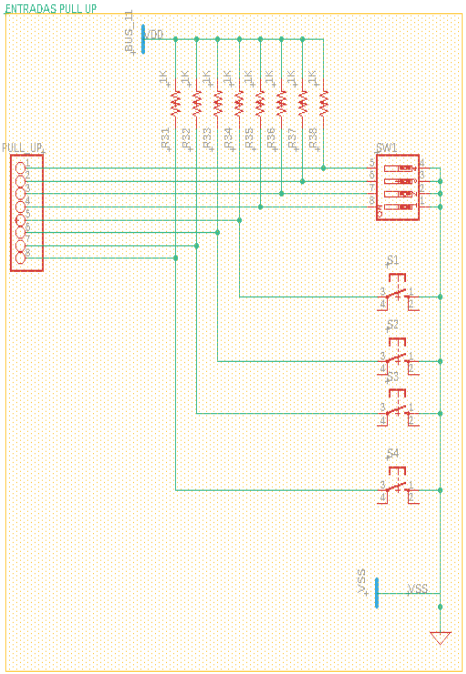
- Realizar las conexiones del módulo didáctico entre los bloques de microcontroladores con los de entradas y salidas digitales.
- Utilizar el comando **tris** para la asignación de entrada o salidas en un puerto.
- Utilizar el comando **port** para declarar un bit o puerto con un estado en alto o bajo.

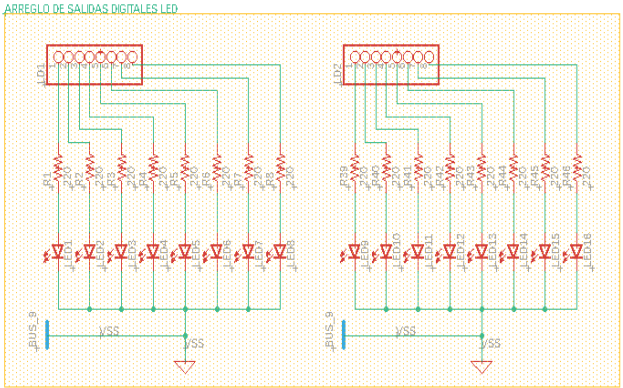
C. Marco Procedimental

1. Conectar los bloques que se muestran en la figura 56 de acuerdo con la tabla 1 de las conexiones de la primera práctica.


PULL_UP
PULL_DOWN
U1
PIC18F4550
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
D1
D2
D3
D4
D5
D6
D7
D8
LED-RED
10k
220


MICROCONTROLADOR
PIC18F4455_40


ENTRADAS PULL UP
PULL_UP
SW1


ARREGLO DE SALIDAS DIGITALES LED

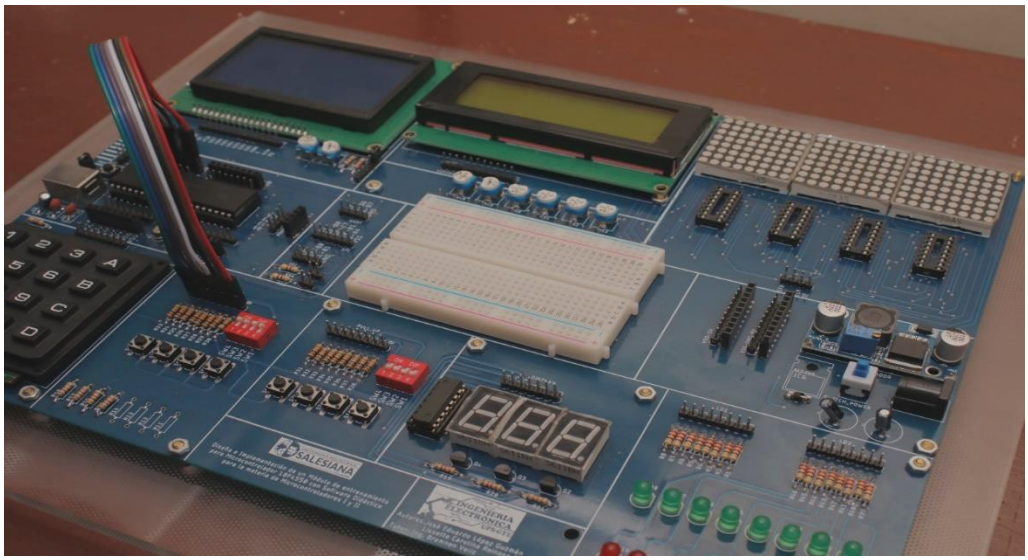


Figura 56. Esquema de conexión de PRÁCTICA 1

PIC18F4550 – B7	PULL DOWN 7
PIC18F4550 – B6	PULL DOWN 6
PIC18F4550 – B5	PULL DOWN 5
PIC18F4550 – B4	PULL DOWN 4
PIC18F4550 – B3	PULL DOWN 3
PIC18F4550 – B2	PULL DOWN 2
PIC18F4550 – B1	PULL DOWN 1
PIC18F4550 – B0	PULL DOWN 0
PIC18F4550 – D7	LED1 7
PIC18F4550 – D6	LED1 6
PIC18F4550 – D5	LED1 5
PIC18F4550 – D4	LED1 4
PIC18F4550 – D3	LED1 3
PIC18F4550 – D2	LED1 2
PIC18F4550 – D1	LED1 1
PIC18F4550 – D0	LED1 0

Tabla 1: Conexiones de PRÁCTICA 1

2. Ejecutar el programa PIC Compiler V 5 de la familia de microchip basado en el lenguaje C como se muestra en la figura.

Figura 57. PIC Compiler V 5 – PRÁCTICA 1

3. Al ejecutar el programa dar clic en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_1** como se muestra en la figura-

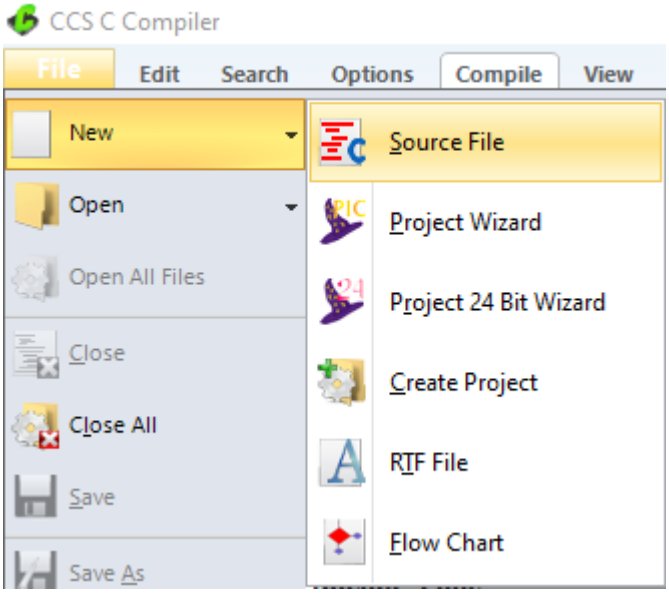


Figura 58. Creación de archivo de código en lenguaje C – PRÁCTICA 1

4. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura 59.

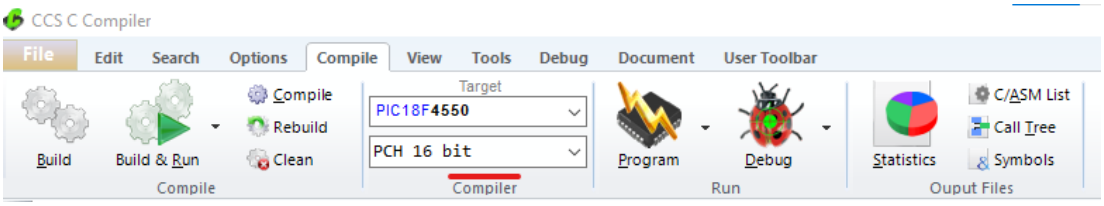


Figura 59. Configuración del microcontrolador en el compilador – PRÁCTICA 1

5. En el archivo creado denominado PRÁCTICA 1, se tendrá un entorno donde escribir el código que se programa en el microcontrolador como se muestra en la figura

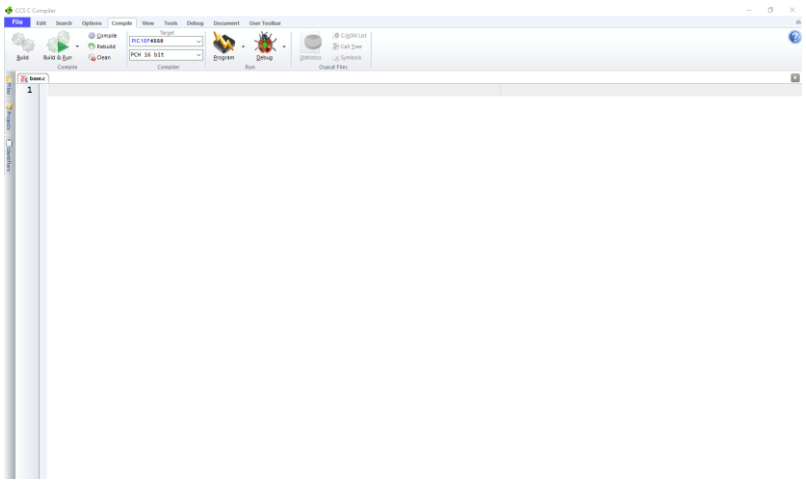


Figura 60. Entorno de programación – PRÁCTICA 1

6. Se procede a llamar a la librería del controlador a utilizar, los fusibles y configurar el cristal de cuarzo a utilizar en el microcontrolador, configurar los puertos a usar con su respectivo registro

```
#include <18f4550.h>//libreria del pic
#fuses   HS // fusible para el cristal
#use delay(clock=20Mhz)//definir la velocidad de cristal
#byte portb=0x0f81//registro del puerto b
#byte portd=0x0f83//registro del puerto d
```

Figura 61. Declaración de librerías -PRÁCTICA 1

7. Posteriormente se declara el puerto b como entradas, seteado un 0 lógico en todo el puerto, y el puerto d como salida.

```
void main()
{
   set_tris_b(255);//designamos el puerto b como entradas
   portb=0;//seteamos en 0 la entrada
   set_tris_d(0);//designar el puerto d como salidas
   portd=0;//seteamos en 0 la entrada
```

Figura 62. Configuración de puertos como entrada y salidas -PRÁCTICA 1

8. En el bloque main principal se procede a declarar que el valor en el puerto d tendrá el valor que se encuentre en el puerto b.

```
while(true)
{
   portd=portb;//asignamos el valor de la entrada al pueto d
}
```

Figura 63. Bloque main -PRÁCTICA 1

9. En el segmento 3 se configura la activación de color verde en el display de HMI el dibujo de la bomba como se muestra en la figura 64

```
void main()
{
   set_tris_b(255);//designamos el puerto b como entradas
   portb=0;//seteamos en 0 la entrada
   set_tris_d(0);//designar el puerto d como salidas
   portd=0;//seteamos en 0 la entrada
   while(true)
   {
      portd=portb;//asignamos el valor de la entrada al pueto d
   }
}
```

Figura 64. Programa final – Práctica 1

10. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura 65 y 66.

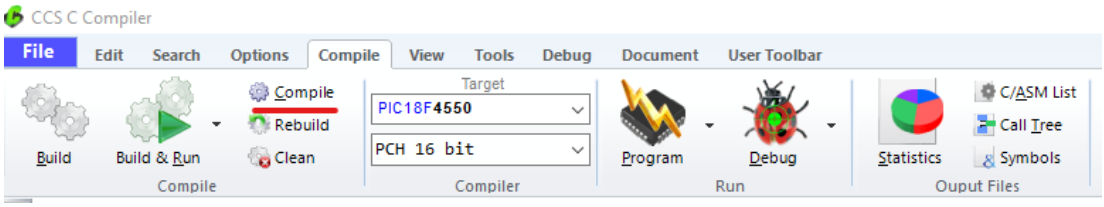


Figura 65. Botón para compilar el programa receptor – Práctica 1

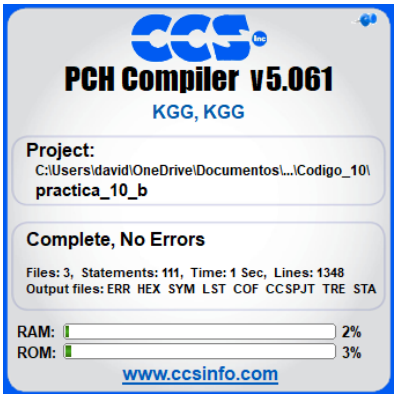


Figura 66. Compilación del programa receptor – Práctica 1

11. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el

microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura 67.

Figura 67. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 1

12. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

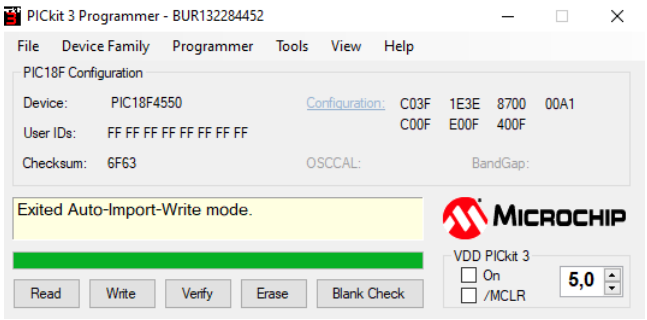


Figura 68. Programación de PIC mediante el Pickit 3 – Práctica 10

D. Registro de Resultados

En la primera práctica se tiene como resultado la lectura de los estados en los bits del puerto b, el valor del puerto b se escribe en el puerto d realizando la variación entre estado altos y bajos en el módulo didáctico.

E. Conexiones físicas de la PRÁCTICA

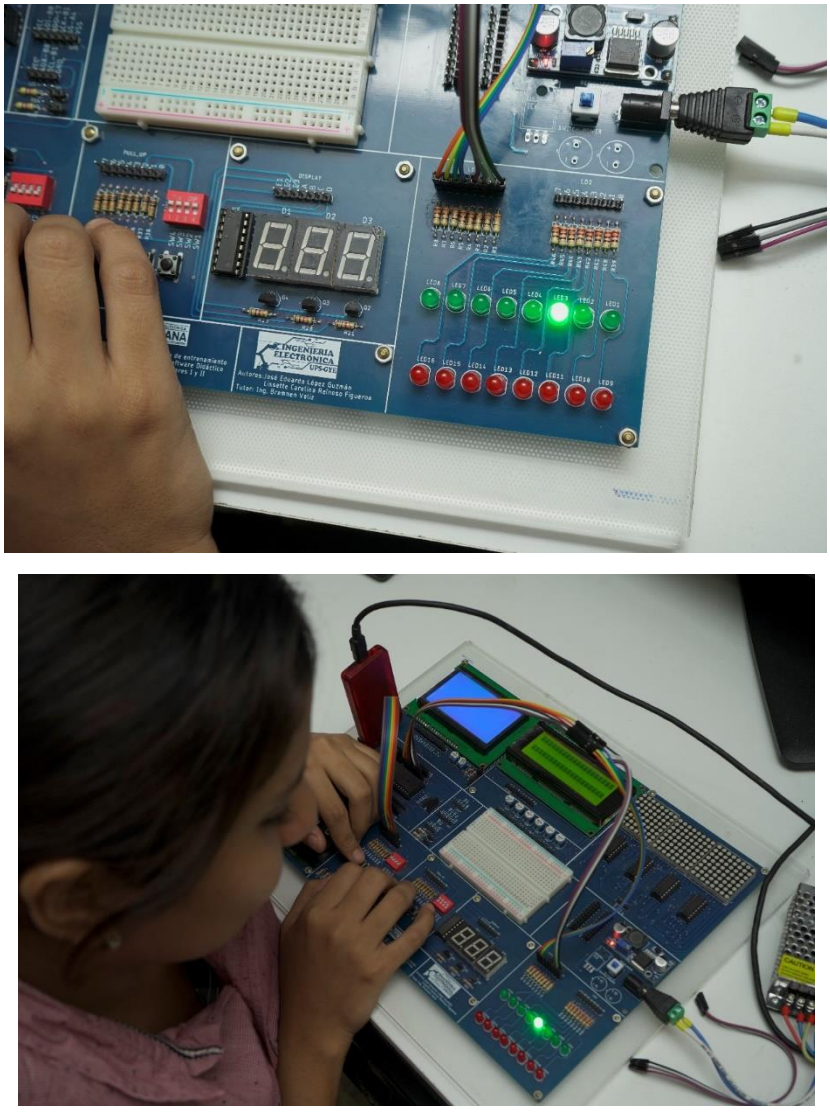


Figura 69. Conexiones físicas - Práctica 1

ANEXO 3.2: RESOLUCIÓN DE PRÁCTICA 2
AUTOMATIZACIÓN INDUSTRIAL

PRÁCTICA 2

NÚMERO DE ESTUDIANTES: 20

DOCENTE

ING. BREMNEN MARINO VELIZ NOBOA MSc.

TIEMPO ESTIMADO: 2 HORAS

TEMA: "DISEÑO DE UNA CALCULADORA GRAFICA UTILIZANDO EL LCD, TECLADO MATRICIAL Y PIC18F4550"

Elaborado por:	**Revisado por:**	**Aprobado por:** Ing. Orlando Barcia Ayala
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

A. Objetivo General

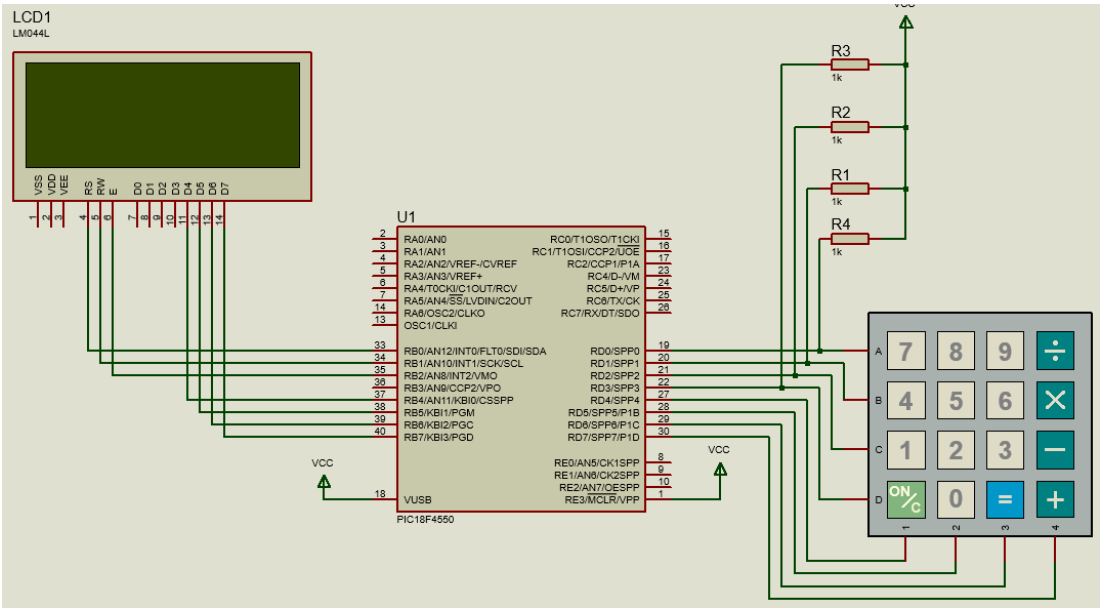
- Diseñar una calculadora grafica utilizando el LCD, Teclado Matricial y PIC18F4550

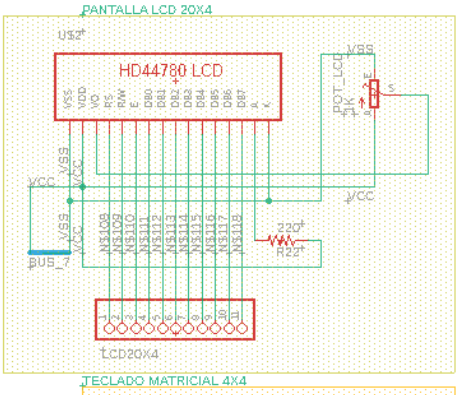
B. Objetivos Específicos

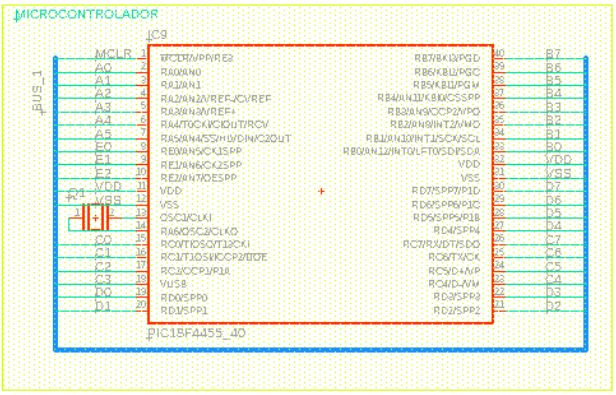
- Realizar las conexiones del módulo didáctico entre los bloques de microcontroladores con el bloque de LCD y TECLADO.
- Utilizar librería para el manejo de una pantalla Lcd de 20 x 4.
- Utilizar librería para el manejo de un teclado matricial de 4x4.

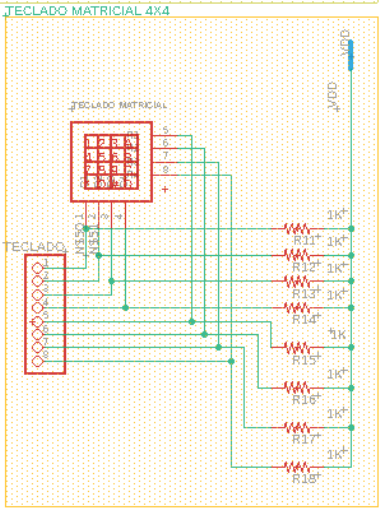
C. Marco Procedimental

1. Conectar los bloques que se muestran en la figura 70 de acuerdo con la tabla 2 de las conexiones de la segunda práctica.


LCD1
LM044L
U1
PIC18F4550
R3
R2
R1
R4
1k
VCC


PANTALLA LCD 20X4
HD44780 LCD
LCD20X4


MICROCONTROLADOR
PIC18F4455_40


TECLADO MATRICIAL 4X4
TECLADO MATRICIAL
TECLADO

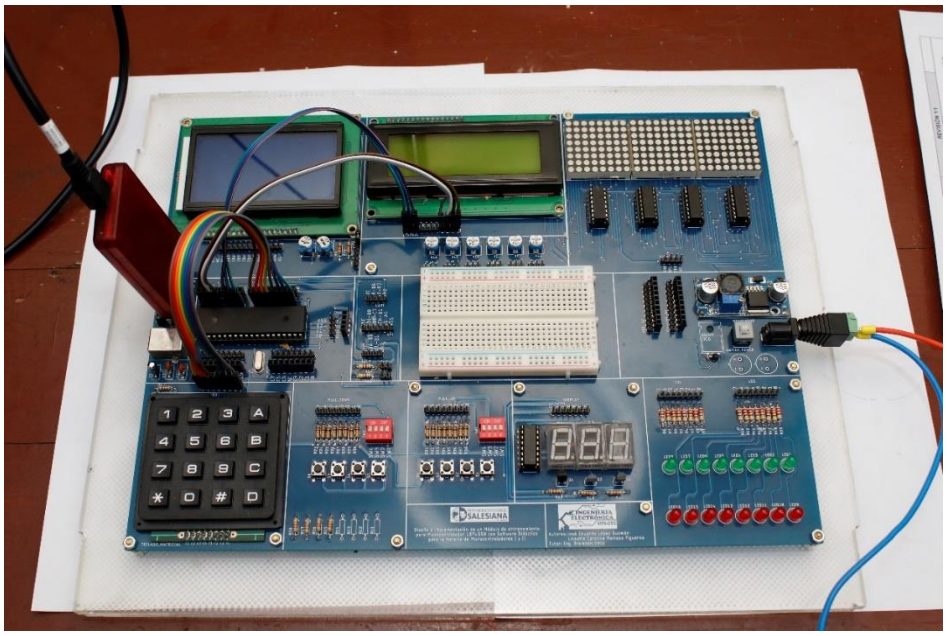


Figura 70. Esquema de conexión de PRÁCTICA 2

PIC18F4550 – B7	LCD D7
PIC18F4550 – B6	LCD D6
PIC18F4550 – B5	LCD D5
PIC18F4550 – B4	LCD D4
PIC18F4550 – B2	LCD E
PIC18F4550 – B1	LCD RW
PIC18F4550 – B0	LCD RS
PIC18F4550 – D7	TECLADO f4
PIC18F4550 – D6	TECLADO f3
PIC18F4550 – D5	TECLADO f2
PIC18F4550 – D4	TECLADO f1
PIC18F4550 – D3	TECLADO c4
PIC18F4550 – D2	TECLADO c3
PIC18F4550 – D1	TECLADO c2
PIC18F4550 – D0	TECLADO c1

Tabla 2: Conexiones de PRÁCTICA 2

2. Ejecutar el programa PIC Compiler V 5 de la familia de microchip basado en el lenguaje C como se muestra en la figura.

Figura 71. PIC Compiler V 5 – PRÁCTICA 2

3. Al ejecutar el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_2** como se muestra en la figura-

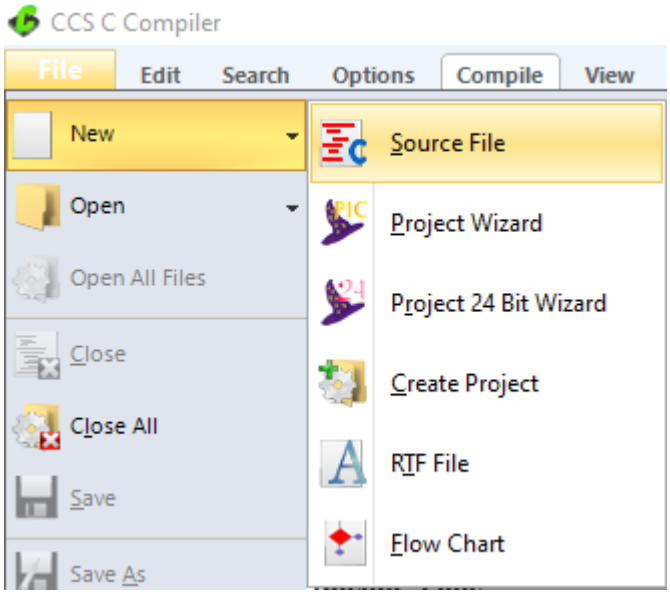


Figura 72. Creación de archivo de código en lenguaje C – PRÁCTICA 2

4. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

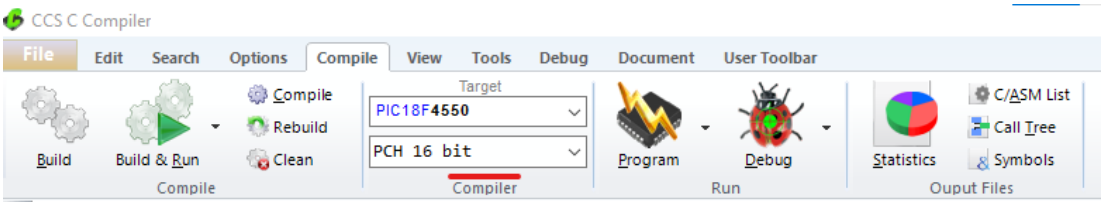


Figura 73. Configuración del microcontrolador en el compilador – PRÁCTICA 2

5. Se procede a llamar a la librería del controlador a utilizar, los fusibles y configurar el cristal de cuarzo a utilizar en el microcontrolador, configurar los puertos a usar con su respectivo registro, librerías del teclado y el Lcd

```
#INCLUDE <18f4550.h>
#FUSES HS,xt                     //High speed Osc (> 4mhz)
#FUSES PUT                      //Power Up Timer
#FUSES NOPROTECT                //Code not protected from reading
#FUSES NOBROWNOUT               //No brownout reset
#FUSES NOMCLR                     //Master Clear pin enabled
#FUSES NOLVP                    //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#use delay(clock=20M)
#define LCD_RS_PIN      PIN_B0       //B0 COMO RS del LCD
#define LCD_RW_PIN      PIN_B1       //B1 COMO RW del LCD
#define LCD_ENABLE_PIN  PIN_B2       //B2 COMO E del LCD
#define LCD_DATA4       PIN_B4       //B4 COMO D4 del LCD
#define LCD_DATA5       PIN_B5       //B5 COMO D5 del LCD
#define LCD_DATA6       PIN_B6       //B6 COMO D6 del LCD
#define LCD_DATA7       PIN_B7       //B7 COMO D7 del LCD
#include <lcd.c> //LIBRERIA LCD
//#define use_portb_kbd TRUE
#include <kb4.c> //LIBRERIA TECLADO
#include <math.h> //LIBRERIA MATEMATICA
#byte porta=0x0f80 //REGISTRO PUERTO A
#byte portb=0x0f81//REGISTRO PUERTO B
#byte portc=0x0f82//REGISTRO PUERTO C
#byte portd=0x0f83//REGISTRO PUERTO D
#byte porte=0x0f84//REGISTRO PUERTO E
```

Figura 74. Declaración de librerías y registros -PRÁCTICA 2

6. Declarar las variables a utilizar y los sub-métodos a utilizar como se muestra en figura 64.

```
int x=1,y=0; //variables posicion del lcd
long a=0,b=0;//variables de banderas
float d1=0,d2=0;//variables digitos
int signo=0;//variable de la operacion a realizar
float resultado=0.0;//variable del resultado
```

Figura 75. Declaración de variables -PRÁCTICA 2

7. Posteriormente se declara el método principal del código como es muestra en la figura

```
void main()
{
   lcd_init();// inicia libreria lcd
   Kbd_init(); //inicia libreria teclado
   printf(lcd_putc,"\f Calculadora Basica");//mensaje
   delay_ms(500);//espera
   printf(lcd_putc,"\f");//limpia lcd
while(true)
{
b=0;//inicia bandera en 0 para ingreso de valor 1
tecla(); //espera digito
b=1;//inicia bandera en 1 para ingreso de valor 2
tecla();//espera digito
if(signo==1){resultado=d1+d2;} //Realiza suma
if(signo==2){resultado=d1-d2;}//Realiza resta
if(signo==3){resultado=d1*d2;}//Realiza multiplicacion
if(signo==4){resultado=d1/d2;}//Realiza division
   lcd_gotoxy(1,2);   printf(lcd_putc,"\f%f",resultado);//muestra resultado
   delay_ms(2000);//espera de 2 segundos
   printf(lcd_putc,"\f");//limpia lcd
   x=1;//reset de puntero
   a=0;//reset de a
}
}
```

Figura 76. Método principal – Práctica 2

8. Se procede a configurar el sub-método tecla donde se adquiere los valores del Lcd y se ejecuta de acuerdo a la operación matemática que se desea al seleccionar el botón ‘A’ se ejecuta la suma, el botón ‘B’ se ejecuta

la resta, el botón ‘C’ se ejecuta la multiplicación, el botón ‘D’ se ejecuta la división, donde al pulsar ‘#’ da el resultado de acuerdo a la operación seleccionada, es importante primero pulsar un valor numérico del 0 al 9 presente en el teclado como se muestra en la figura.

```
void tecla(){
int t;
char k='\n';

   while(true){
   do{ //espera hasta que se presione una tecla
   k=kbd_getc(); //Captura valor del teclado
   } while(k=='\n');

    if(k=='A'){if(b==0){d1=a;a=0;lcd_gotoxy(x,1); printf(lcd_putc,"+");x++;signo=1;break;}else{d2=a;a=0;signo=1;break;}}
    if(k=='B'){if(b==0){d1=a;a=0;lcd_gotoxy(x,1); printf(lcd_putc,"-");x++;signo=2;break;}else{d2=a;a=0;signo=2;break;}}
    if(k=='C'){if(b==0){d1=a;a=0;lcd_gotoxy(x,1); printf(lcd_putc,"*");x++;signo=3;break;}else{d2=a;a=0;signo=3;break;}}
    if(k=='D'){if(b==0){d1=a;a=0;lcd_gotoxy(x,1); printf(lcd_putc,"/");x++;signo=4;break;}else{d2=a;a=0;signo=4;break;}}
    if(k=='*'){x=0;a=0;}
    if(k=='#'){if(b==0){d1=a;a=0;break;}if(b==1){d2=a;a=0;break;}}
    if((k!='#')&&(k!='*')&&(k!='A')&&(k!='B')&&(k!='C')&&(k!='D'))
   {
   t=k-48;
   a=(a*10)+t;
   lcd_gotoxy(x,1);
   printf(lcd_putc,"%d",t);
   delay_ms(5);
   x++;
   }
 }

}
```

Figura 77. Sub método tecla – Práctica 2

9. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura.

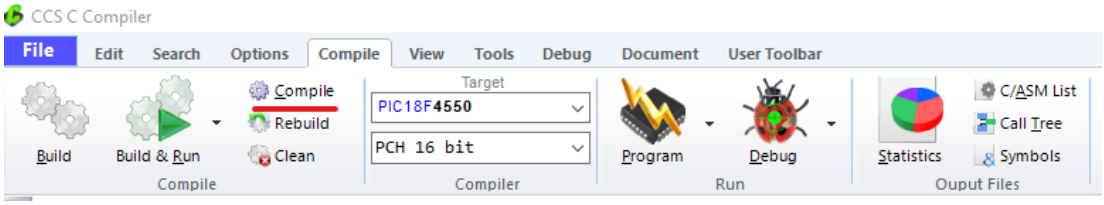


Figura 78. Botón para compilar el programa – Práctica 2

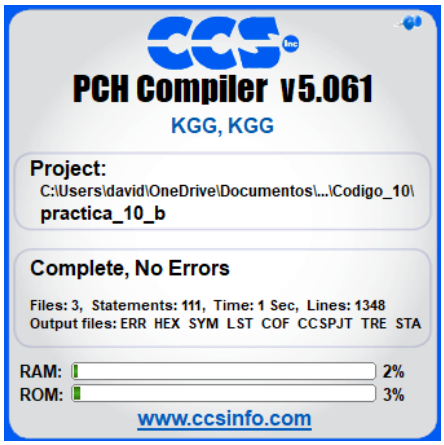


Figura 79. Compilación del programa – Práctica 2

10. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 80. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 2

11. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión

seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

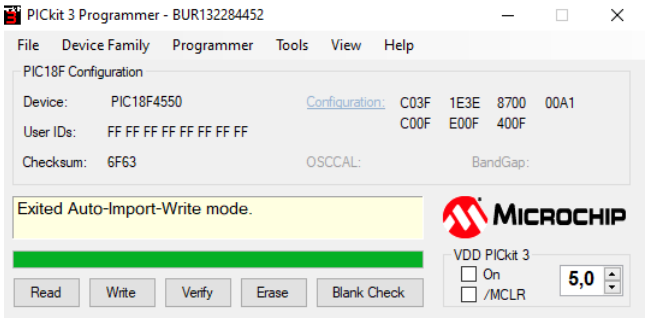


Figura 81. Programación de PIC mediante el Pickit 3 – Práctica 2

D. Registro de Resultados

En la segunda práctica se tiene como resultado una aplicación la cual permite al estudiante aprender el correcto uso de las librerías de la pantalla Lcd y del teclado matricial teniendo como función la de una calculadora básicas de dos dígitos.

E. Resultados de PRÁCTICA

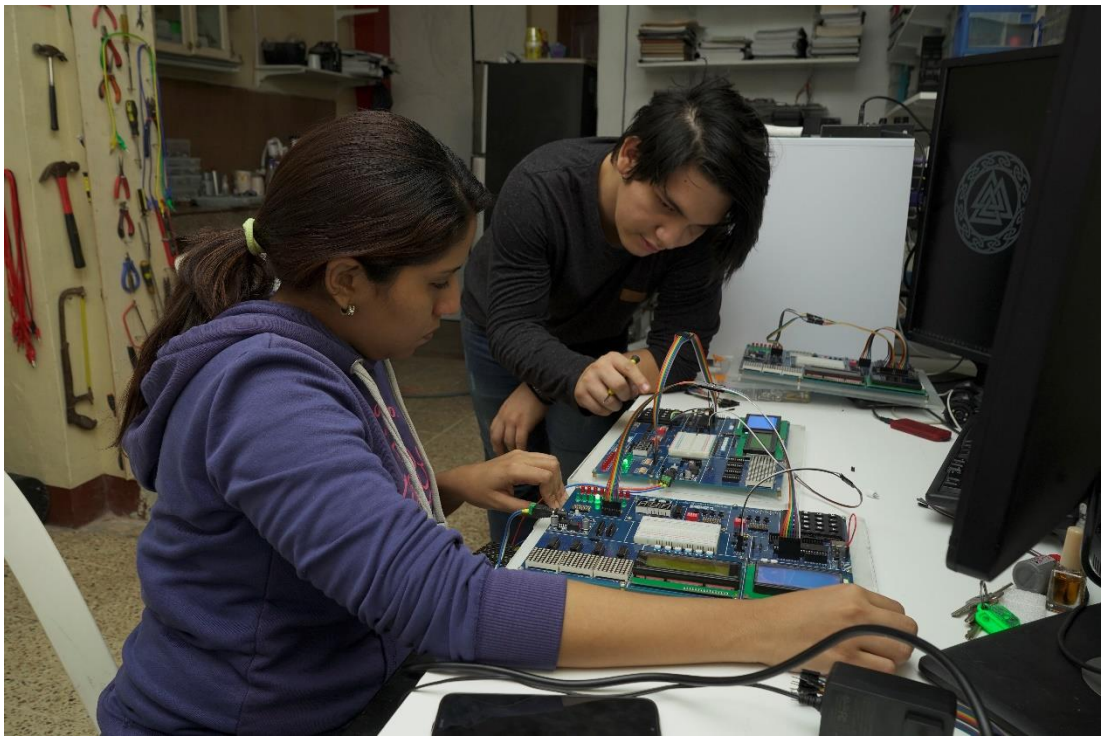


Figura 82. Conexiones físicas - Práctica 2

ANEXO 3.3: RESOLUCIÓN DE PRÁCTICA 3
AUTOMATIZACIÓN INDUSTRIAL

PRÁCTICA 3

NÚMERO DE ESTUDIANTES: 20

DOCENTE

ING. BREMNEN MARINO VELIZ NOBOA MSc.

TIEMPO ESTIMADO: 2 HORAS

TEMA: "CONTADOR ASCENDENTE Y DESCENDENTE UTILIZANDO LOS DISPLAY DE 7 SEGMENTOS Y BOTONES."

Elaborado por:	**Revisado por:**	**Aprobado por:** Ing. Orlando Barcia Ayala
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

A. Objetivo General

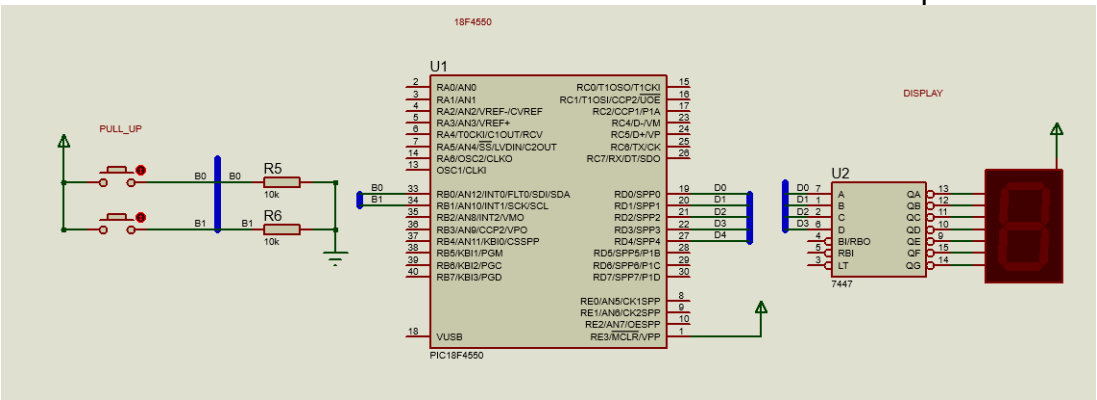
- Diseñar un contador ascendente y descendente utilizando los display de 7 segmentos y botone.

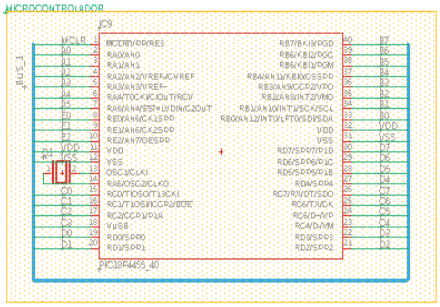
B. Objetivos Específicos

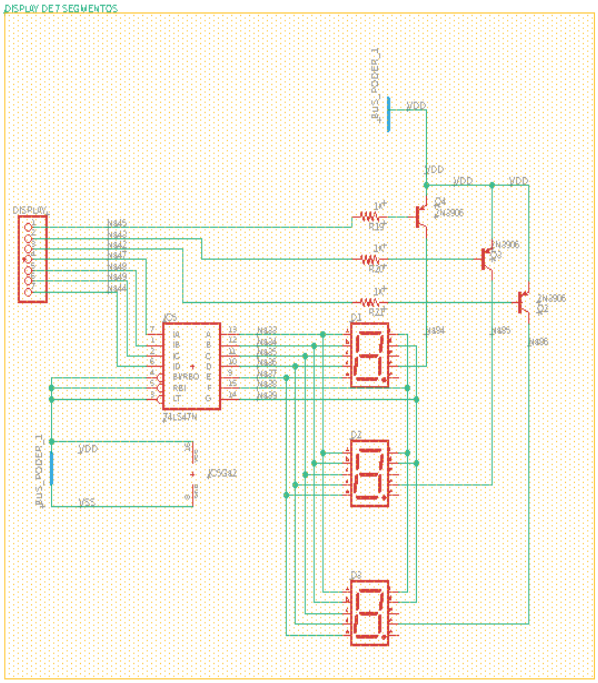
- Implementar un aplicativo para la lectura de dos entradas digitales.
- Controlar el estado de un display de 7 segmentos mediante una variable en el código.

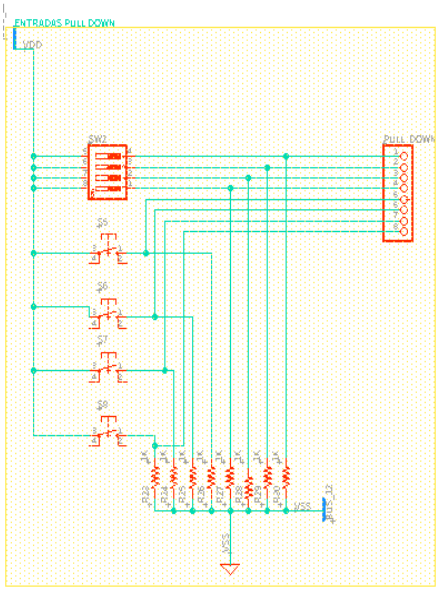
C. Marco Procedimental

1. Conectar los bloques que se muestran en la figura 83 de acuerdo con la tabla 3 de las conexiones de la tercera práctica.




MICROCONTROLADOR
PIC18F4455_40


DISPLAY DE 7 SEGMENTOS
BUS_PODER_1
VDD
VSS
DISPLAY
74LS47N
2N2906


ENTRADAS PULL DOWN
VDD
SW7
PULL DOWN
VSS

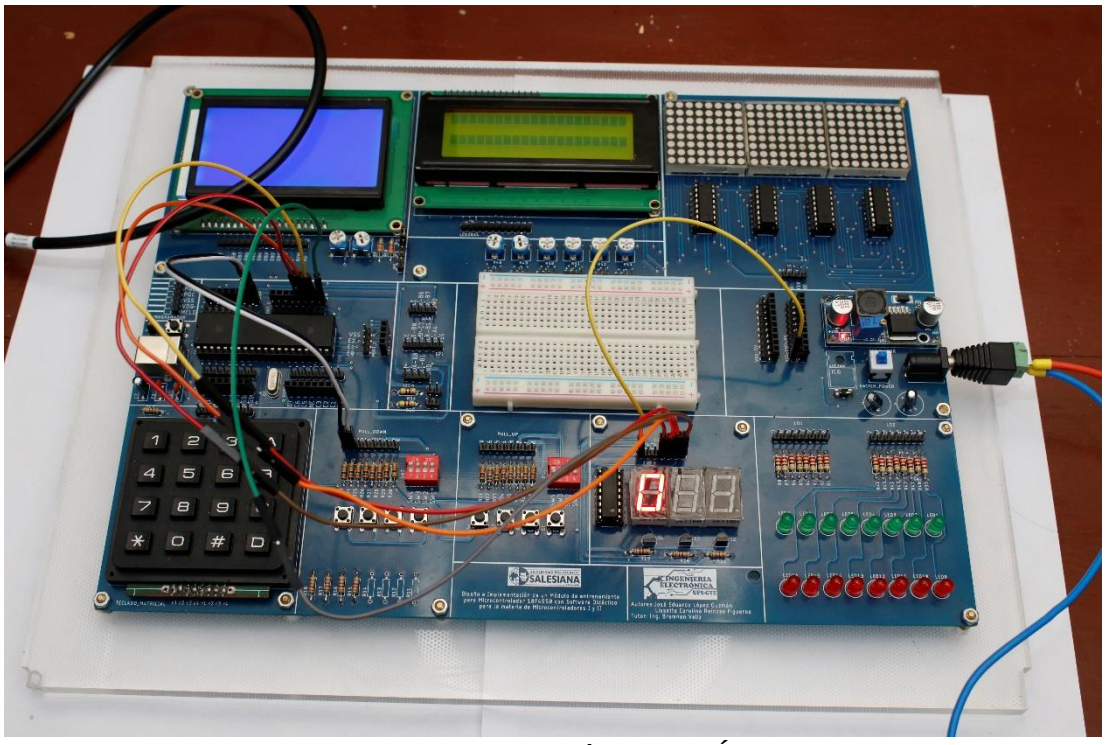


Figura 83. Esquema de conexión de PRÁCTICA 3

PIC18F4550 – B1	Pull up 7
PIC18F4550 – B0	Pull up 6
PIC18F4550 – D3	Display D
PIC18F4550 – D2	Display C
PIC18F4550 – D1	Display B
PIC18F4550 – D0	Display A
GND	Display E1

Tabla 3: Conexiones de PRÁCTICA 3

2. Ejecutar el programa PIC Compiler V 5 de la familia de microchip basado en el lenguaje C como se muestra en la figura.

Figura 84. PIC Compiler V 5 – PRÁCTICA 3

3. Al ejecutar el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_3** como se muestra en la figura-

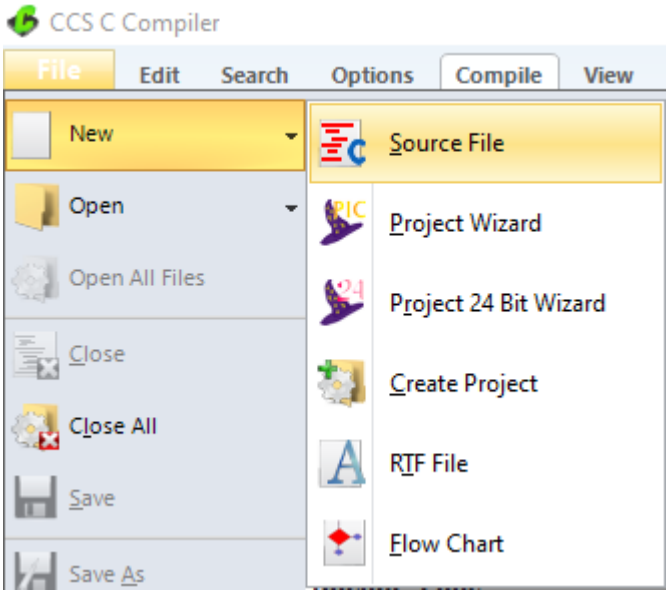


Figura 85. Creación de archivo de código en lenguaje C – PRÁCTICA 3

4. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

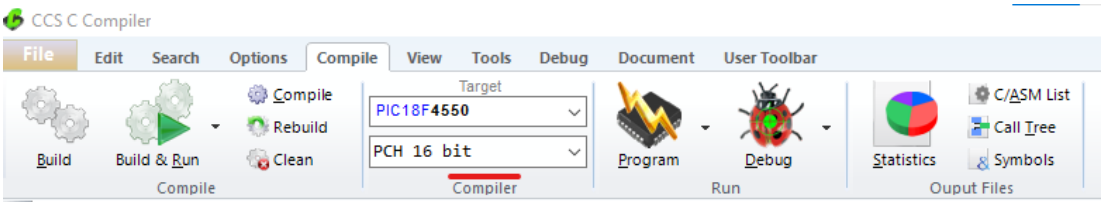


Figura 86. Configuración del microcontrolador en el compilador – PRÁCTICA 3

5. Se procede a llamar a la librería del controlador a utilizar, los fusibles y configurar el cristal de cuarzo a utilizar en el microcontrolador, configurar los puertos a usar con su respectivo registro, librerías del teclado y el Lcd

```
#include <18f4550.h>
#fuses XT
#use delay(clock=20Mhz)
#byte portb=0x0f81
#byte portd=0x0f83
```

Figura 87. Declaración de librerías y registros -PRÁCTICA 3

6. Declarar las variables a utilizar y los sub-métodos a utilizar como se muestra en figura 64.

```
unsigned int conteo;
```

Figura 88. Declaración de variables -PRÁCTICA 3

7. Se procede a configurar el método principal donde se configura el puerto b como entrada solo al bit 0 y 1 cuya función será de adquirir un valor digital y aumentar o decrementar una variable llamada conteo la cual enviará su valor al puerto d donde se conectará a un conversor de código bcd a 7 segmentos de esta manera visualizar el valor en el display. Teniendo como seguro si el valor de conteo supera a 9 se reseteará a 0 dicho valor como se muestra en la figura.

```
void main(){
set_tris_b(3);// bit 0 y bit 1 como entrada
portb=0;
set_tris_d(0);
portd=0;
while(true){
if(portb==1){
conteo++;
delay_ms(250);
}
if(portb==2){
conteo--;
delay_ms(250);
}
if (conteo>=9)
{
conteo=0;
}
portd=conteo;


}
}
```

Figura 89. Sub método tecla – Práctica 3

8. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura.

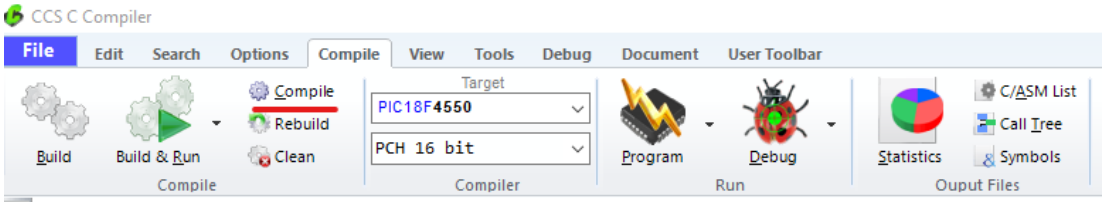


Figura 90. Botón para compilar el programa – Práctica 3

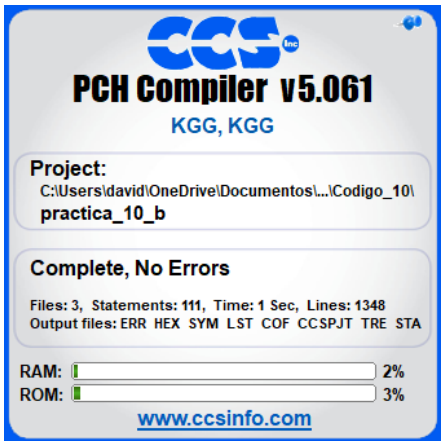


Figura 91. Compilación del programa – Práctica 3

9. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 92. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 3

10. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión

seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

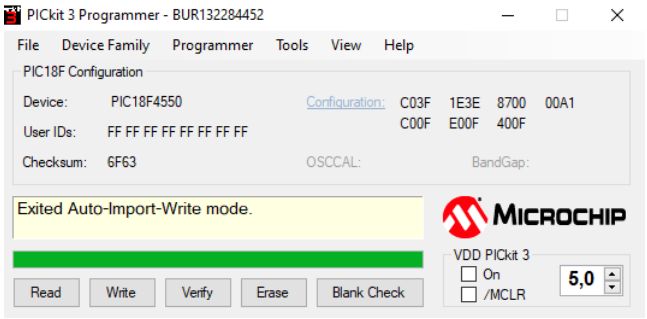


Figura 93. Programación de PIC mediante el Pickit 3 – Práctica 3

D. Registro de Resultados

En la tercera práctica se tiene un aplicativo el cual tendrá la función de permitir la visualización de una variable en un display de 7 segmentos, teniendo como función el decremento o incremento de dicho valor mediante la lectura de dos variables digitales conectándose todo a un microcontrolador Pic 18f4550

E. Conexiones físicas de la PRÁCTICA

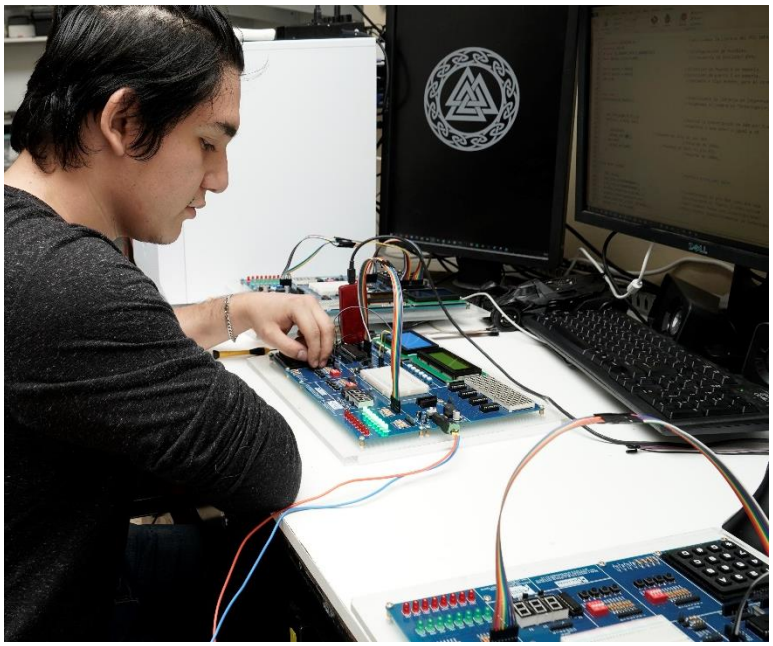


Figura 94. Conexiones físicas - Práctica 3

ANEXO 3.4: RESOLUCIÓN DE PRÁCTICA 4
AUTOMATIZACIÓN INDUSTRIAL

PRÁCTICA 4

NÚMERO DE ESTUDIANTES: 20

DOCENTE

ING. BREMNEN MARINO VELIZ NOBOA MSc.

TIEMPO ESTIMADO: 2 HORAS

TEMA: "DETECTOR DE DISTANCIA MEDIANTE EL SENSOR ULTRASÓNICO VISUALIZANDO LOS DATOS EN UN LCD."

Elaborado por:	**Revisado por:**	**Aprobado por:** Ing. Orlando Barcia Ayala
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

A. Objetivo General

- Diseñar un aplicativo para detectar la distancia mediante el Sensor Ultrasónico visualizando los datos en un LCD

B. Objetivos Específicos

- Realizar el acondicionamiento para la lectura de un sensor ultrasónico.
- Cablear los bloques de microcontrolador, pantalla y Lcd con el bloque del protoboard donde estará el sensor.

C. Marco Procedimental

1. Realizar las conexiones que se muestran en la figura.

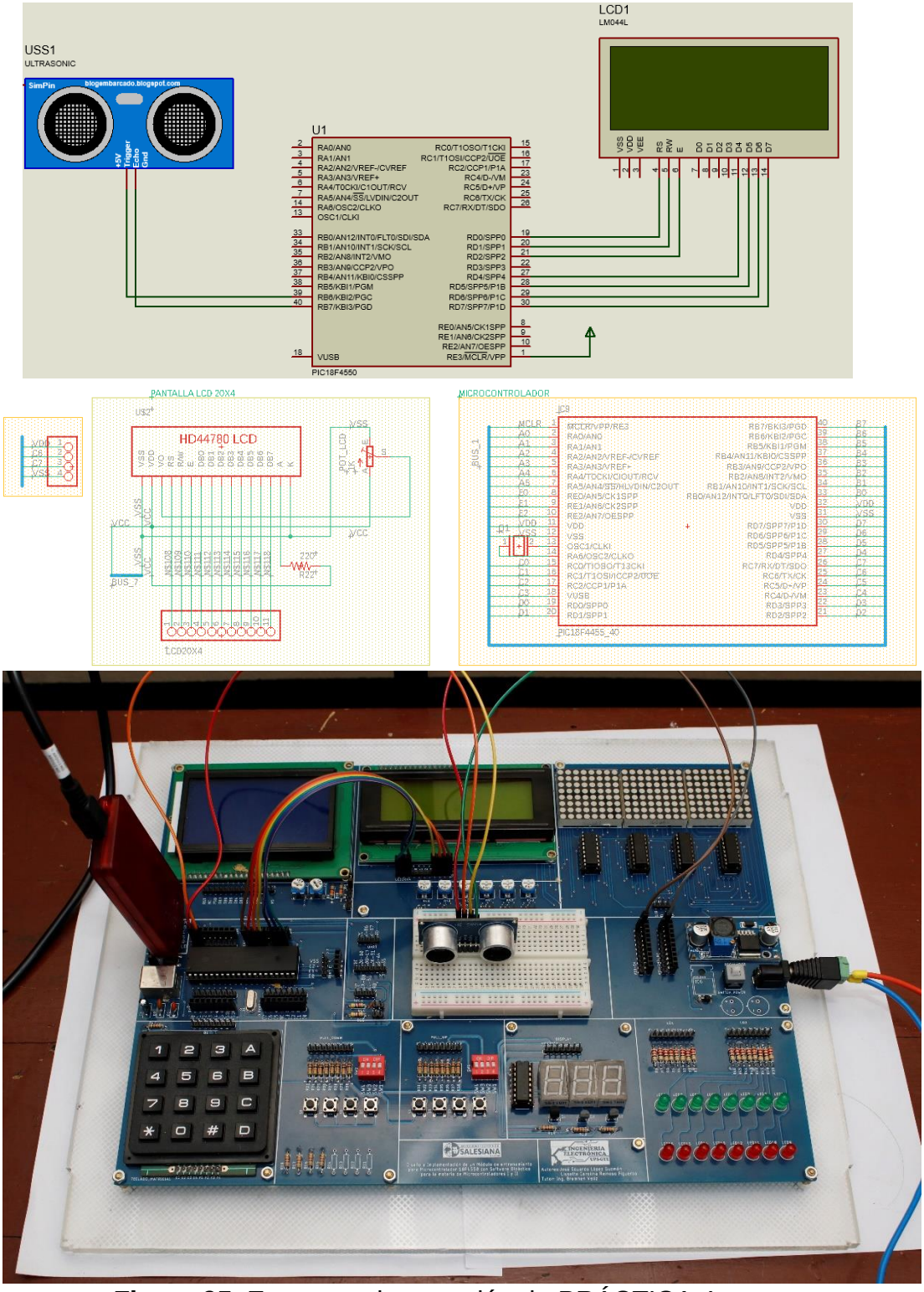


Figura 95. Esquema de conexión de PRÁCTICA 4

Elaborado por:	**Revisado por:**	**Aprobado por:** Ing. Orlando Barcia Ayala
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

PIC18F4550 – B7	ULTRASONICO TRIGGER
PIC18F4550 – B6	ULTRASONICO ECO
PIC18F4550 – D7	LCD D7
PIC18F4550 – D6	LCD D6
PIC18F4550 – D5	LCD D5
PIC18F4550 – D4	LCD D4
PIC18F4550 – D2	LCD E
PIC18F4550 – D1	LCD RW
PIC18F4550 – D0	LCD RS

Tabla 4: Conexiones de PRÁCTICA 4

2. Ejecutar el programa PIC Compiler V 5 de la familia de microchip basado en el lenguaje C como se muestra en la figura.

Figura 96. PIC Compiler V 5 – PRÁCTICA 4

3. Al ejecutar el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_4** como se muestra en la figura-

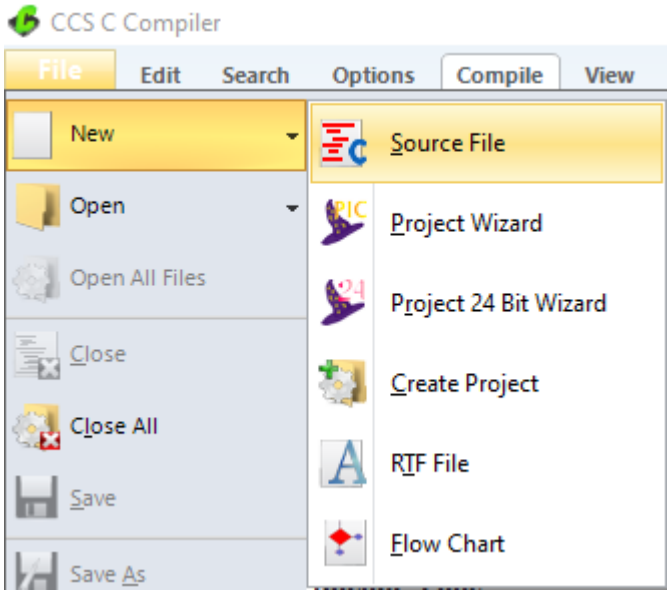


Figura 97. Creación de archivo de código en lenguaje C – PRÁCTICA 4

4. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

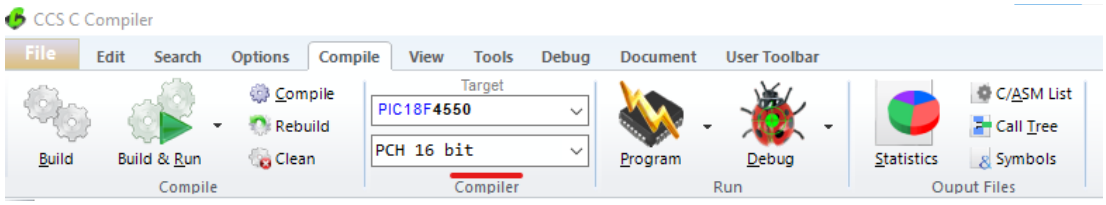


Figura 98. Configuración del microcontrolador en el compilador – PRÁCTICA 4

5. Se procede a llamar a la librería del controlador a utilizar, los fusibles y configurar el cristal de cuarzo a utilizar en el microcontrolador, configurar los puertos a usar con su respectivo registro, librerías del teclado y el Lcd

```
#include <18f4550.h>
#Fuses HSPLL, NOWDT, NOPROTECT, NOLVP, NODEBUG, USBDIV, PLL2, CPUDIV1, VREGEN,HS,XT
#use delay (clock=20M)         //Seleccionamos la frecuencia de reloj de 20MHz
#define LCD_RS_PIN      PIN_D0                                    ////
#define LCD_RW_PIN      PIN_D1                                    ////
#define LCD_ENABLE_PIN  PIN_D2                                    ////
#define LCD_DATA4       PIN_D4                                    ////
#define LCD_DATA5       PIN_D5                                    ////
#define LCD_DATA6       PIN_D6                                    ////
#define LCD_DATA7       PIN_D7
#include  <lcd.c>
#include <math.h>       //LIBRERÍA DE MATEMÁTICAS
```

Figura 99. Declaración de librerías y registros -PRÁCTICA 4

6. Declarar las variables a utilizar y los sub-métodos a utilizar como se muestra en figura 64.

```
#define Echo PIN_B7
#define Trigger PIN_B6
//PROTOTIPO DE FUNCIÓN PARA MEDIR DISTANCIA
void mide_distancia(void);
long tiempo_eco=0;  //tiempo que dura el eco
float  distancia=0;   //distancia en cm
```

Figura 100. Declaración de variables -PRÁCTICA 4

7. Posteriormente se declara el método principal del código como es muestra en la figura

```
void main(void){
   //TICK=(4/Fosc)*Prescaler---->TICK=(4/48000000)*8=0.000666ms
   setup_timer_1(T1_INTERNAL | T1_DIV_BY_8); //Config del timer1
   lcd_init();                   //Inicializa LCD
   delay_ms(50);                 //Esperamos 50ms
   lcd_putc(" /f");               //Borra la pantalla
   lcd_gotoxy(1,1);              //Cursor en la posicion columna 1 fila 1
   lcd_putc("Sens Ultrasonico"); //Escribimos en la lcd
   while(true){                  //Ciclo infinito
      mide_distancia();          //Llamamos la función mide_distancia()
      delay_ms(30);              //Esperamos 30ms
   }
}
```

Figura 101. Segmento 1 – Práctica 4

8. Se procede a configurar el sub-métodos tecla donde se adquiere los valores del Lcd y se ejecuta de acuerdo a la operación matemática que se desea al seleccionar el botón 'A' se ejecuta la suma, el botón 'B' se ejecuta la resta, el botón 'C' se ejecuta la multiplicación, el botón 'D' se ejecuta la división, donde al pulsar '#' da el resultado de acuerdo a la operación seleccionada, es importante primero pulsar un valor numérico del 0 al 9 presente en el teclado como se muestra en la figura.

```
void mide_distancia(void){          //Función mide_distancia
   output_high(Trigger);            // pulso de disparo
   delay_us(10);                    //Esperamos 10 us
   output_low(Trigger);             //Apagamos el pulso
   while(!input_state(Echo)){
   }                                //espera flanco de subida
   set_timer1(0);                   //Iniciamos el timer1
   while(input_state(Echo)){
   }                                //esperamos flanco de bajada
   tiempo_eco = get_timer1();       //Leemos el valor del timer1
   distancia = (tiempo_eco/2)*.000666*34; //Convertimos a distancia
   lcd_gotoxy(1,2);                 //Cursor en la posicion columna 1 fila
   printf(lcd_putc,"nDist: %1.2f cm", distancia); //Imprimimos la distan
   delay_ms(200);                   //Esperamos 200ms
}
```

Figura 102. Sub método tecla – Práctica 4

9. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura.

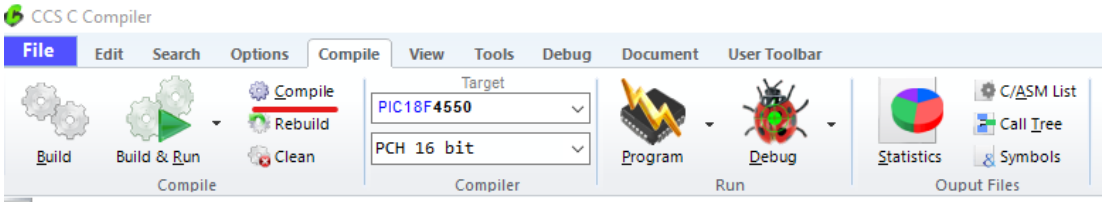


Figura 103. Botón para compilar el programa – Práctica 4

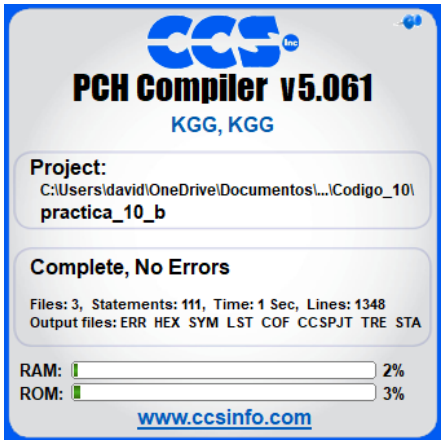


Figura 104. Compilación del programa – Práctica 4

10. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 105. Conexión del Pickit 3 con tarjeta de entrenamiento – Práctica 4

11. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

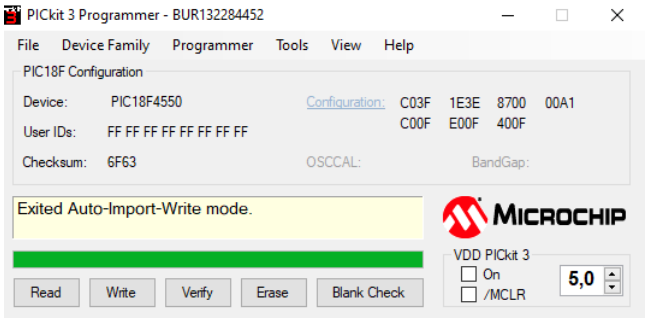


Figura 106. Programación de PIC mediante el Pickit 3 – Práctica 4

D. Registro de Resultados

En la cuarta práctica se tiene como resultado la lectura de la distancia de un sensor ultrasónico mediante su acondicionamiento mediante un microcontrolador 18f4550 y su posterior visualización en una pantalla Lcd de 20 x4.

E. Conexiones físicas de la PRÁCTICA

Figura 107. Conexiones físicas - Práctica 4

ANEXO 3.5: RESOLUCIÓN DE PRÁCTICA 5
AUTOMATIZACIÓN INDUSTRIAL

PRÁCTICA 5

NÚMERO DE ESTUDIANTES: 20

DOCENTE

ING. BREMNEN MARINO VELIZ NOBOA MSc.

TIEMPO ESTIMADO: 2 HORAS

TEMA: "CONEXIONES ENTRE 2 PIC 18F4550."

Elaborado por:	**Revisado por:**	**Aprobado por:** Ing. Orlando Barcia Ayala
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

A. Objetivo General

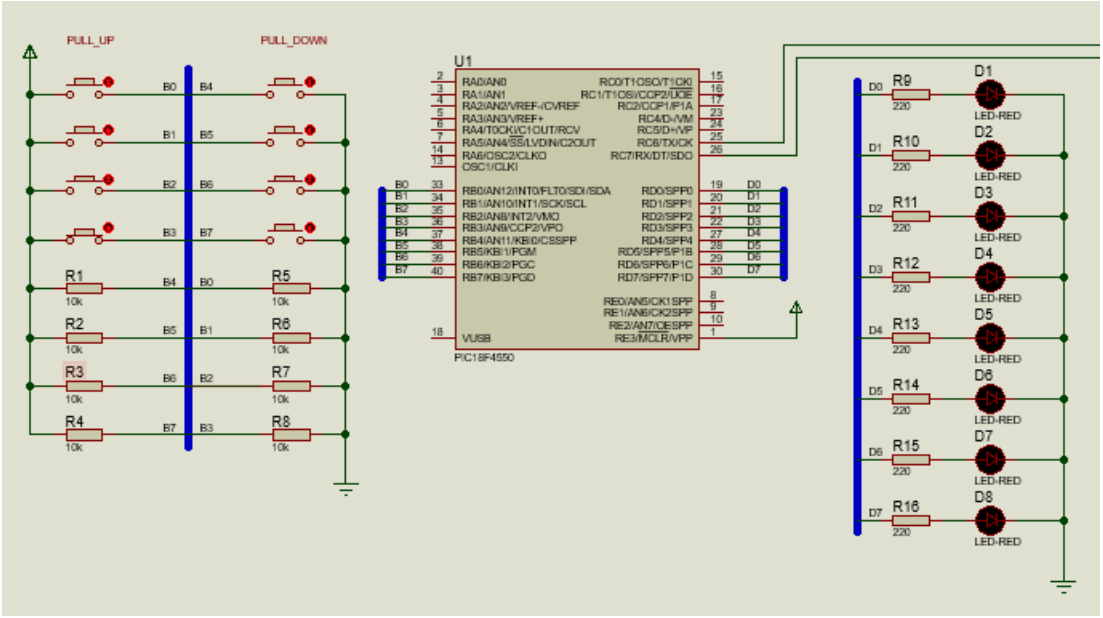
- Implementar un aplicativo para el envío y recepción de datos entre dos microcontroladores 18f4550.

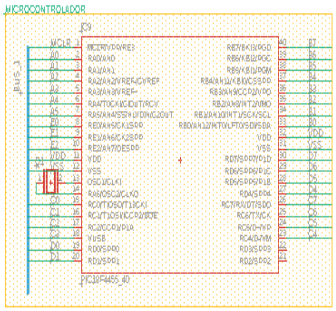
B. Objetivos Específicos

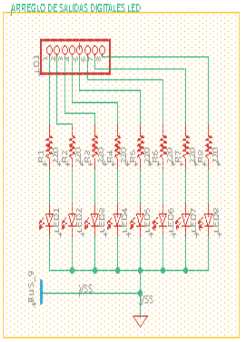
- Definir variables para él envió de datos desde un microcontrolador a otro
- Elaborar el programa de recepción de datos mediante protocolo UART
- Realizar el cableado respectivo entre los dos módulos didácticos

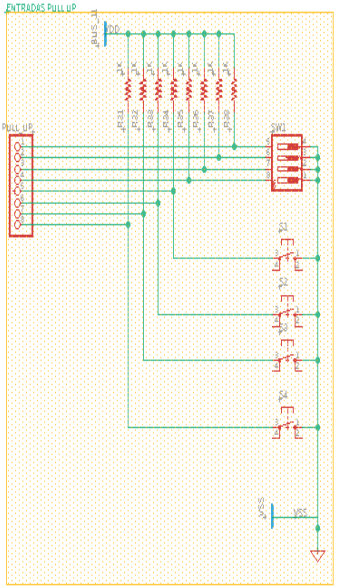
C. Marco Procedimental

1. Se procede a realizar el primer circuito que sirve como el emisor de información donde se toma de referencia el esquemático de la figura









Elaborado por:	**Revisado por:**	**Aprobado por:** Ing. Orlando Barcia Ayala
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

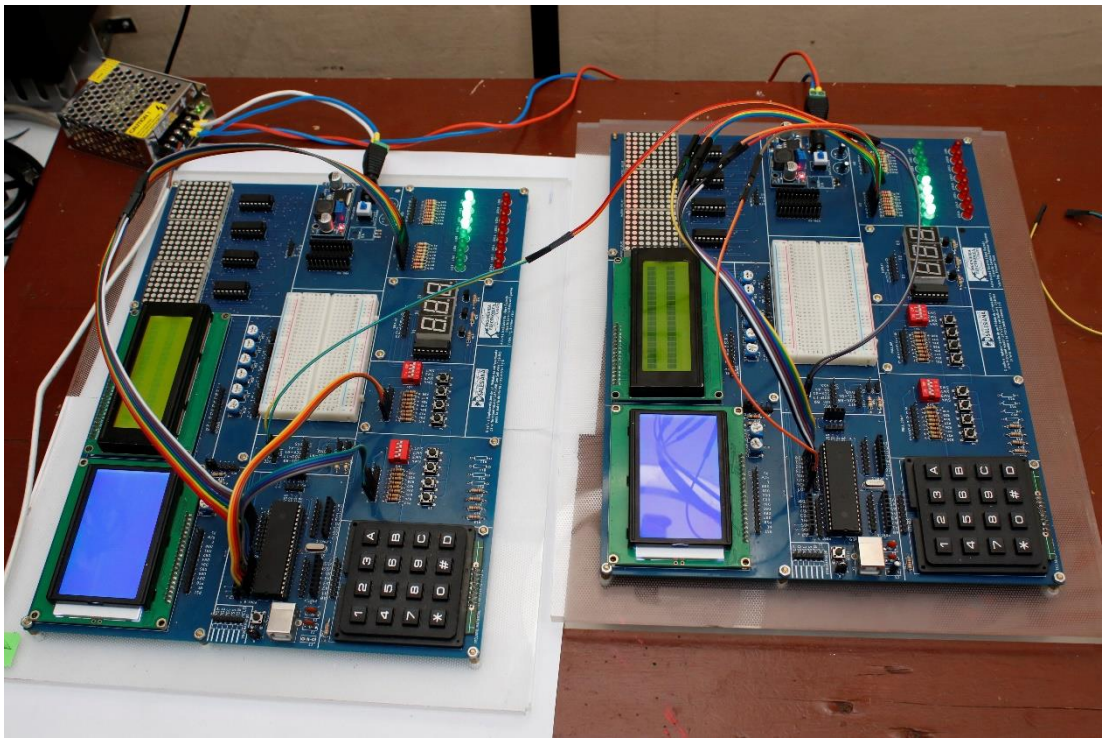


Figura 108. Esquema de conexión de PRÁCTICA 5

MODULO 1 PIC18F4550 – B7	PULL DOWN 7
MODULO 1 PIC18F4550 – B6	PULL DOWN 6
MODULO 1 PIC18F4550 – B5	PULL DOWN 5
MODULO 1 PIC18F4550 – B4	PULL DOWN 4
MODULO 1 PIC18F4550 – B3	PULL DOWN 3
MODULO 1 PIC18F4550 – B2	PULL DOWN 2
MODULO 1 PIC18F4550 – B1	PULL DOWN 1
MODULO 1 PIC18F4550 – B0	PULL DOWN 0
MODULO 1 PIC18F4550 – D7	LED1 7
MODULO 1 PIC18F4550 – D6	LED1 6
MODULO 1 PIC18F4550 – D5	LED1 5
MODULO 1 PIC18F4550 – D4	LED1 4

MODULO 1 PIC18F4550 – D3	LED1 3
MODULO 1 PIC18F4550 – D2	LED1 2
MODULO 1 PIC18F4550 – D1	LED1 1
MODULO 1 PIC18F4550 – D0	LED1 0
MODULO 1 PIC18F4550 – C7	MODULO 2 PIC18F4550 – C6
MODULO 1 PIC18F4550 – C6	MODULO 2 PIC18F4550 – C7

Tabla 5: Conexiones emisor de PRÁCTICA 5

2. Ejecutar el programa PIC Compiler V 5 de la familia de microchip basado en el lenguaje C como se muestra en la figura.

Figura 109. PIC Compiler V 5 – PRÁCTICA 5

3. Al ejecutar el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_5** como se muestra en la figura-

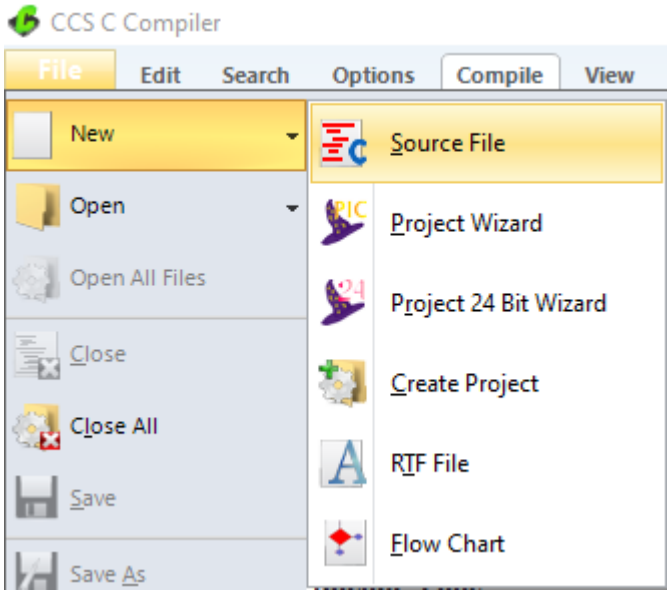


Figura 110. Creación de archivo de código en lenguaje C – PRÁCTICA 5

4. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

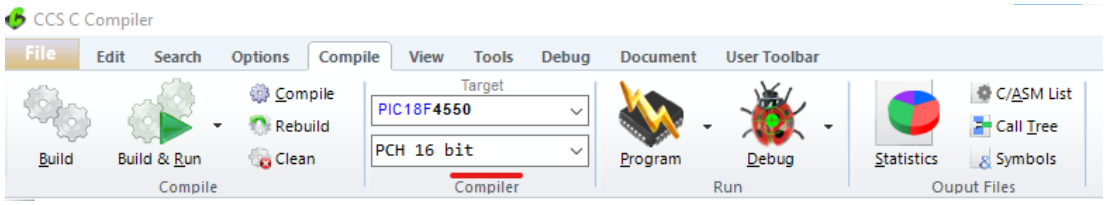


Figura 111. Configuración del microcontrolador en el compilador – PRÁCTICA 5

5. Se procede a llamar a la librería del microcontrolador, definir los fusibles, velocidad del cristal. y la librería del UART donde de define la velocidad y los puertos a utilizar como se muestra en la figura.

```
#include <18f4550.h>
//Configuración de fusibles.
#fuses HS, NOWRT, NOWDT,NOPROTECT, NOPUT, NOLVP
//Frecuencia de oscilador 20MHz. Para el 18F4550 Cristal externo escribir "crystal"
#use delay(crystal=20000000)
//Librería stdlib
#include <stdlib.h>
//velocidad 9600, 8 bits, Sin paridad, TX C6, RX C7
#use RS232(baud=9600,bits=8,parity=N,xmit=pin_c6,rcv=pin_c7)
```

Figura 112. Declaración de librerías y registros -PRÁCTICA 5

6. Declarar los puertos y su respectivo registro como se muestra en la figura.

```
#byte portb=0x0f81
#byte portc=0x0f82
#byte portd=0x0f83
```

Figura 113. Declaración de registro del puerto-PRÁCTICA 5

7. Posteriormente se declara el método principal del código donde se configura el puerto b como entrada y el puerto d como salida donde en el lazo principal obtendrá el valor del puerto b y se mostrara en el puerto d, seguido del envió del valor del puerto d mediante serial como se muestra en la figura

```
void main(){
set_tris_b(255);
portb=0;
set_tris_c(0);
portc=0;
set_tris_d(0);
portd=0;
while(true){
unsigned int a;
portd=portb;
 a=(int)portd;
   putc(a);

  delay_ms(100);
}
}
```

Figura 114. Segmento principal – Práctica 5

8. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura.

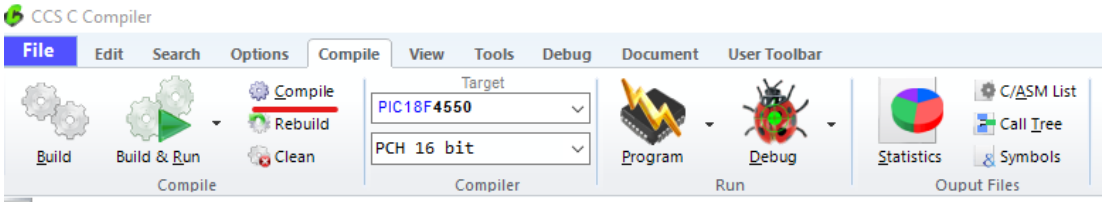


Figura 115. Botón para compilar el programa receptor – Práctica 5

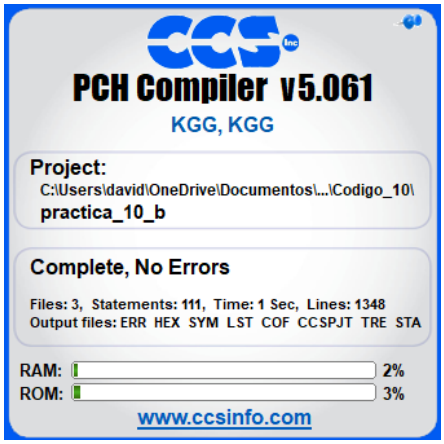


Figura 116. Compilación del programa – Práctica 5

9. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 117. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 5

10. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión

seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

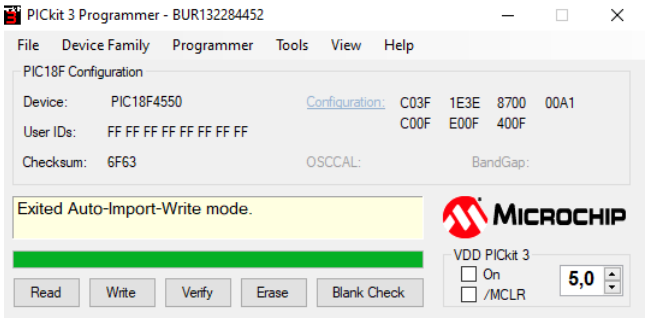


Figura 118. Programación de PIC mediante el Pickit 3 – Práctica 5

11. Se a programar el código del receptor, llamar a la librería del microcontrolador, definir los fusibles , velocidad del cristal. y la librería del UART donde de define la velocidad y los puertos a utilizar como se muestra en la figura.

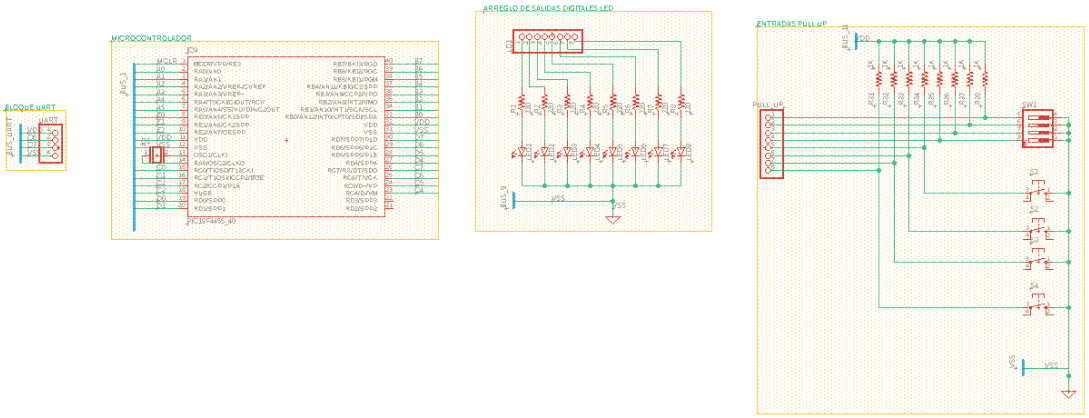


Figura 119. Esquemático del receptor -PRÁCTICA 5

MODULO 2 PIC18F4550 – D7	LED1 7
MODULO 2 PIC18F4550 – D6	LED1 6
MODULO 2 PIC18F4550 – D5	LED1 5
MODULO 2 PIC18F4550 – D4	LED1 4
MODULO 2 PIC18F4550 – D3	LED1 3
MODULO 2 PIC18F4550 – D2	LED1 2
MODULO 2 PIC18F4550 – D1	LED1 1
MODULO 2 PIC18F4550 – D0	LED1 0
MODULO 2 PIC18F4550 – C7	MODULO 1 PIC18F4550 – C6
MODULO 2 PIC18F4550 – C6	MODULO 1 PIC18F4550 – C7

Tabla 6: Conexiones receptor de PRÁCTICA 5

```
#include <18f4550.h>
//Configuración de fusibles.
#fuses HS, NOWRT, NOWDT,NOPROTECT, NOPUT, NOLVP
//Frecuencia de oscilador 20MHz. Para el 18F4550 Cristal externo escribir "crystal"
#use delay(crystal=20000000)
//Librería stdlib
#include <stdlib.h>
//velocidad 9600, 8 bits, Sin paridad, TX C6, RX C7
#use RS232(baud=9600,bits=8,parity=N,xmit=pin_c6,rcv=pin_c7)
```

Figura 120. Declaración de librerías y registros -PRÁCTICA 5

12. Declarar los puertos y su respectivo registro , junto a las variables a utilizar como se muestra en la figura.

```
int dato;
int8 tiempo=100;
#byte portd=0x0f83
```

Figura 121. Declaración de registro del puerto-PRÁCTICA 5

13. Declarar la interrupción de comunicación de manera que al momento de que el controlador detecte algún dato en el puerto serial guarde el dato mediante el comando getc como se muestra en la figura.

```
#int_rda                          //Habilita la función de interrupción por recepción de datos
void rda_isr()                    //Función de interrupción recepción de datos.
{
   dato=getc();                   //Asigna el valor de recepción a la variable dato.

}
```

Figura 122. Declaración de registro del puerto-PRÁCTICA 5

14. Posteriormente se declara el método principal del código donde se configura la interrupción por comunicación serial y en el lazo principal se escribe el valor obtenido en el puerto d como se muestra en la figura

```
void main (void)                  //Función principal
{
   set_tris_d(0);
   portd=0;                  //
   enable_interrupts(int_rda);    //Habilitamos la interrupción por recepción
   enable_interrupts(GLOBAL);     //Habilitamos las interrupciones globales.

   while(true)
   {
      putc(dato);
      portd=dato;
      delay_ms(tiempo);
   }
}
```

Figura 123. Segmento principal emisor – Práctica 5

15. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura.

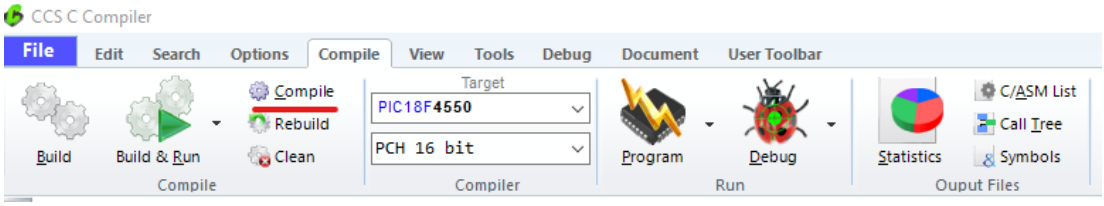


Figura 124. Botón para compilar el programa receptor – Práctica 5

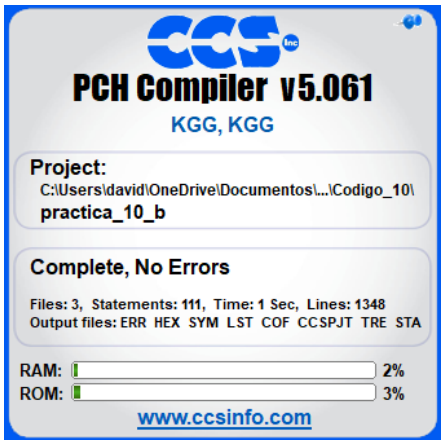


Figura 125. Compilación del programa – Práctica 5

16. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 126. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 5

17. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

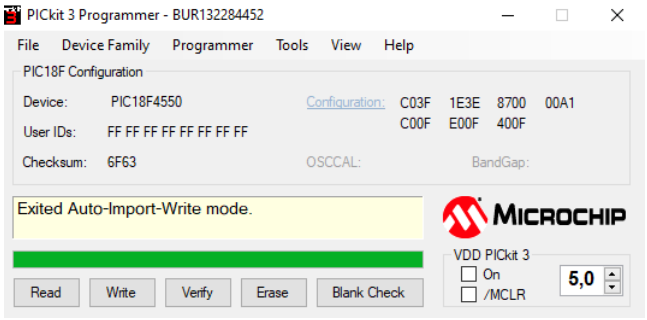


Figura 127. Programación de PIC mediante el Pickit 3 – Práctica 5

D. Registro de Resultados

En la quinta práctica se tiene como resultado la comunicación entre dos controladores Pic 18f4550 donde el primero sirve como emisor del valor de un puerto asignado y el segundo sirve como receptor del valor y se muestra en un puerto asignado.

E. Conexiones físicas de la PRÁCTICA

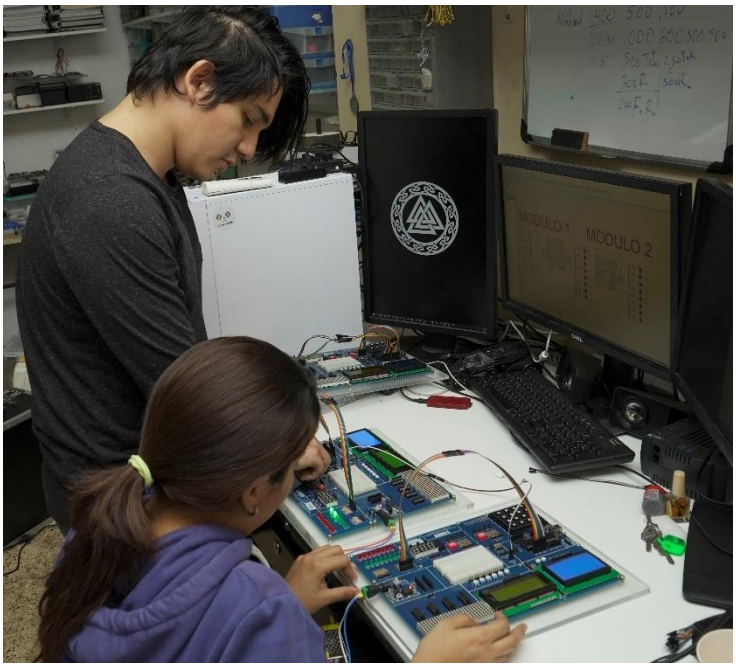


Figura 128. Conexiones físicas - Práctica 5

ANEXO 3.6: RESOLUCIÓN DE PRÁCTICA 6
AUTOMATIZACIÓN INDUSTRIAL

PRÁCTICA 6

NÚMERO DE ESTUDIANTES: 20

DOCENTE

ING. BREMNEN MARINO VELIZ NOBOA MSc.

TIEMPO ESTIMADO: 2 HORAS

TEMA: "CONEXIÓN MEDIANTE EL PIC18F4550 Y LA PC."

Elaborado por:	**Revisado por:**	**Aprobado por:** Ing. Orlando Barcia Ayala
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

A. Objetivo General

- Implementar una conexión entre el Pic 18f4550 y la PC
- Utilizar la comunicación USB del Pic 18f4550.

B. Objetivos Específicos

- Definir variables para él envió de datos desde un microcontrolador a otro
- Elaborar el programa de recepción de datos mediante protocolo UART
- Realizar el cableado respectivo entre los dos módulos didácticos

C. Marco Procedimental

1. Realizar las conexiones que se muestran en la figura.

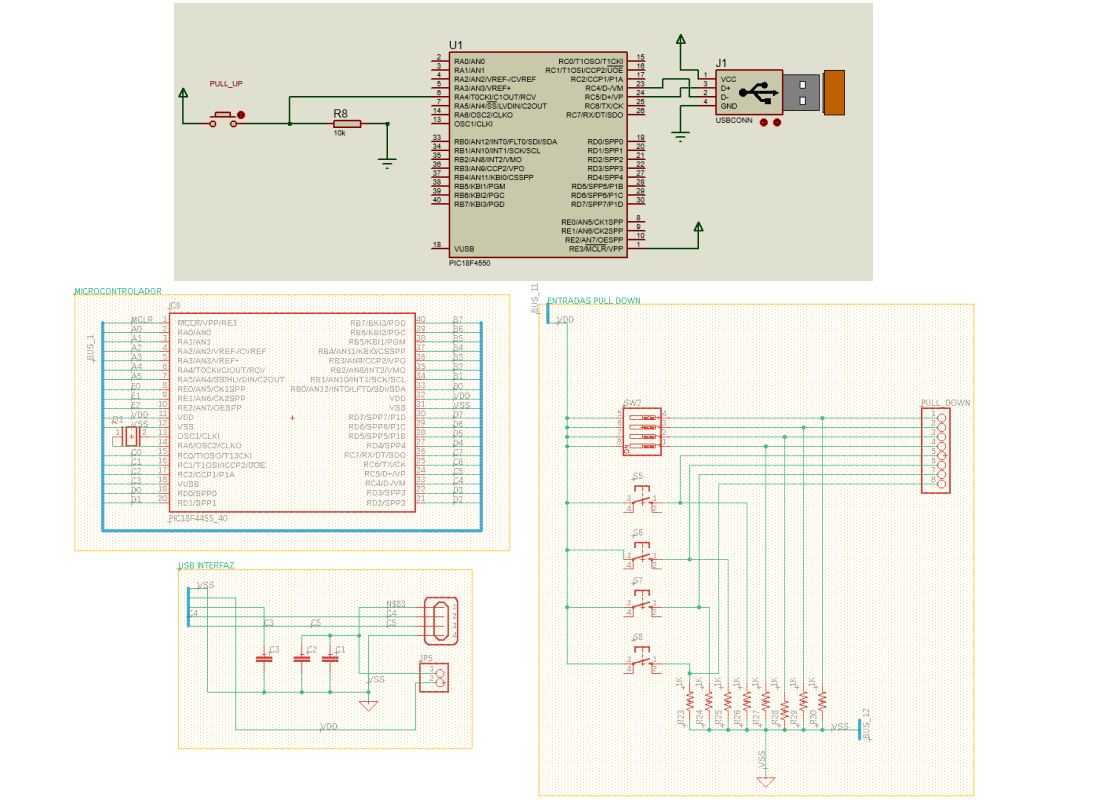


Figura 129. Esquema de conexión de PRÁCTICA 6

Elaborado por:	**Revisado por:**	**Aprobado por:** Ing. Orlando Barcia Ayala
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

MODULO 1 PIC18F4550 – A4	PULL UP _0

Tabla 7: Conexiones emisor de PRÁCTICA 6

2. Ejecutar el programa PIC Compiler V 5 de la familia de microchip basado en el lenguaje C como se muestra en la figura.

Figura 130. PIC Compiler V 5 – PRÁCTICA 6

3. Al ejecutar el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_6** como se muestra en la figura-

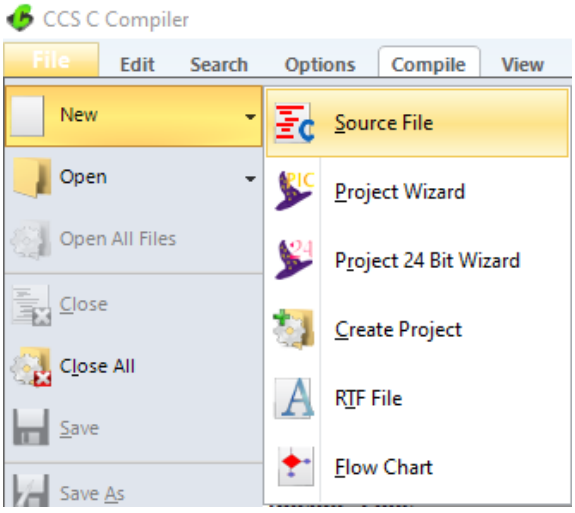


Figura 131. Creación de archivo de código en lenguaje C – PRÁCTICA 6

4. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

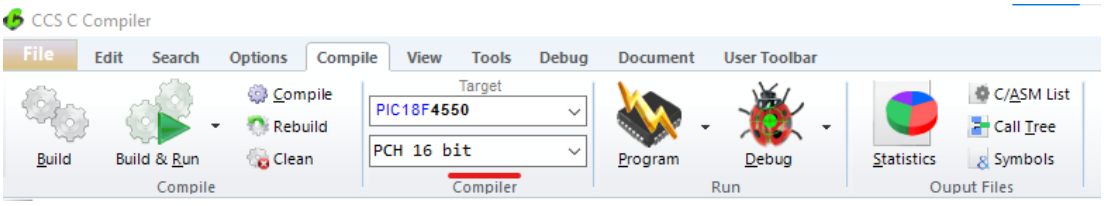


Figura 132. Configuración del microcontrolador en el compilador – PRÁCTICA 6

5. Se procede a llamar a la librería del controlador a utilizar, los fusibles y configurar el cristal de cuarzo a utilizar en el microcontrolador, configurar los puertos a usar con su respectivo registro, declarar el pin donde se utiliza el modo USB al momento de recibir un bit con lógica negada.

```
#include <18F4550.h>
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
#use delay(clock=20000000)
#define BUTTON_PRESSED() !input(PIN_A4)
#define PIN_USB_SENSE   PIN_A3
#define HW_ADC_CONFIG   ADC_CLOCK_INTERNAL
#define HW_ADC_CHANNEL  0
#define HW_ADC_PORTS    AN0
#define HW_INIT() setup_adc_ports(HW_ADC_PORTS)
#define USB_CDC_ISR() handle_incoming_usb()
#define USB_CDC_DELAYED_FLUSH
#define USB_CDC_DATA_LOCAL_SIZE  128
```

Figura 133. Declaración de librerías y registros -PRÁCTICA 6

6. Declarar la librería para el modo USB para que el computador le detecte como un puerto serial, también se define los métodos de espera cuando el USB no este activo y el tamaño del buffer de salida del puerto, incluyendo la librería del stdlib como se muestra en figura 64.

```
#include "ex_usb_common.h"

static void handle_incoming_usb(void);

#include <usb_cdc.h>

#if (getenv("DATA_EEPROM") > 0)
 #define MY_READ_EEPROM(x)    read_eeprom(x)
 #define MY_WRITE_EEPROM(x, y)   write_eeprom(x, y)
#else
 char data[256];
 #define MY_READ_EEPROM(x)    data[x]
 #define MY_WRITE_EEPROM(x, y)   data[x] = y
#endif

#include <stdlib.h>
```

Figura 134. Declaración de librerías del buffer del USB -PRÁCTICA 6

7. Posteriormente se declara un sub-método que funcione al momento de conectar el USB de esta manera permitir la activación del puerto en el computador como es muestra en la figura

```
void usb_debug_task(void)
{
   static int8 last_connected;
   static int8 last_enumerated;
   int8 new_connected;
   int8 new_enumerated;

   new_connected=usb_attached();
   new_enumerated=usb_enumerated();

   if (new_connected)
      LED_ON(LED2);
   else
      LED_OFF(LED2);

   if (new_enumerated)
      LED_ON(LED3);
   else
      LED_OFF(LED3);

   if (new_connected && !last_connected)
      uart_printf("\r\n\nUSB conectado...");
   if (!new_connected && last_connected)
      uart_printf("\r\n\nUSB desconectado");
   if (new_enumerated && !last_enumerated)
      uart_printf("\r\n\nUSB Puerto numero");
   if (!new_enumerated && last_enumerated)
      uart_printf("\r\n\nUSB sin asignar");

   last_connected=new_connected;
   last_enumerated=new_enumerated;
}
```

Figura 135. Sub método del USB en espera – PRÁCTICA 6

8. Se procede a programar el lazo principal donde se da inicio al USB mediante el llamado de los métodos usb_init_c y hw_init, al ejecutar envía los valores mediante el puerto USB del microcontrolador el valor de x que oscilara entre un rango de 0 a 10 como se muestra en la figura.

```
void main(void)
{
int x=0;
   HW_INIT();
   usb_init_cs();
   for(;;)
   {
      usb_task();
      printf(usb_cdc_putc_fast, "\r\nPRUEBA DE COMUNICACION %d",x);
      delay_ms(200);
      usb_debug_task();
      x++;
      if (x>100)
      {
      x=0;
      }
   }
}
```

Figura 136. Bloque principal– Práctica 6

1. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura.

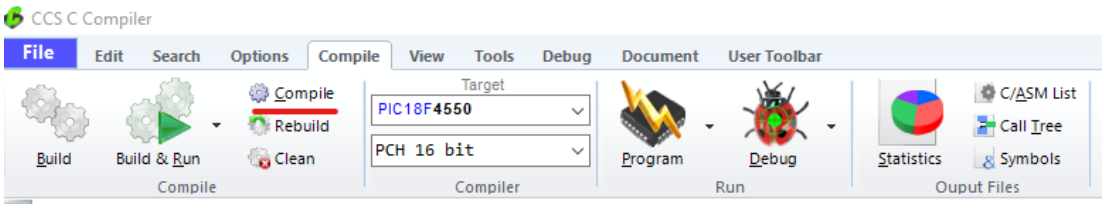


Figura 137. Botón para compilar el programa receptor – Práctica 6

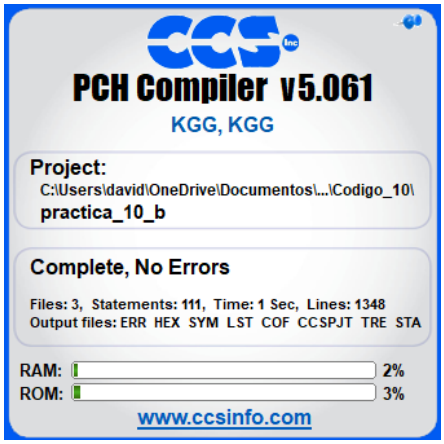


Figura 138. Compilación del programa – Práctica 6

2. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 139. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 6

3. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión

seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

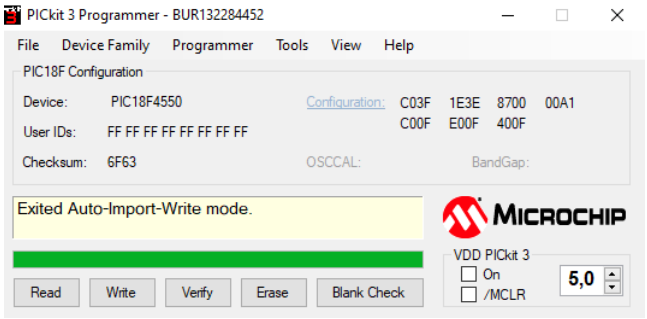


Figura 140. Programación de PIC mediante el Pickit 3 – Práctica 6

D. Registro de Resultados

En la sexta práctica se tiene como resultado una aplicación la cual envía un valor entero mediante el puerto USB integrado en el microcontrolador, el valor se observa mediante el visualizador serial.

E. Resultados - Práctica 6

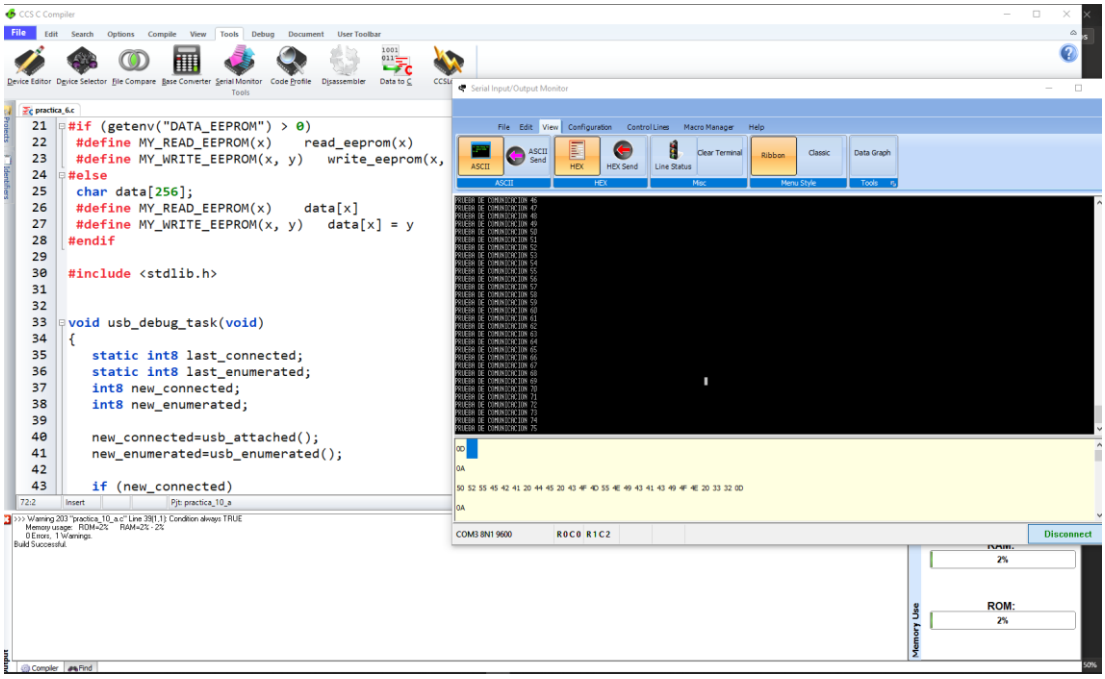


Figura 141. Resultados - Práctica 6

ANEXO 3.7: RESOLUCIÓN DE PRÁCTICA 7
AUTOMATIZACIÓN INDUSTRIAL

PRÁCTICA 7

NÚMERO DE ESTUDIANTES: 20

DOCENTE

ING. BREMNEN MARINO VELIZ NOBOA MSc.

TIEMPO ESTIMADO: 2 HORAS

TEMA: "INTERRUPCIÓN UTILIZANDO BOTON Y SEÑAL ANALÓGICA MEDIANTE EL PIC18F4550"

A. Objetivo General

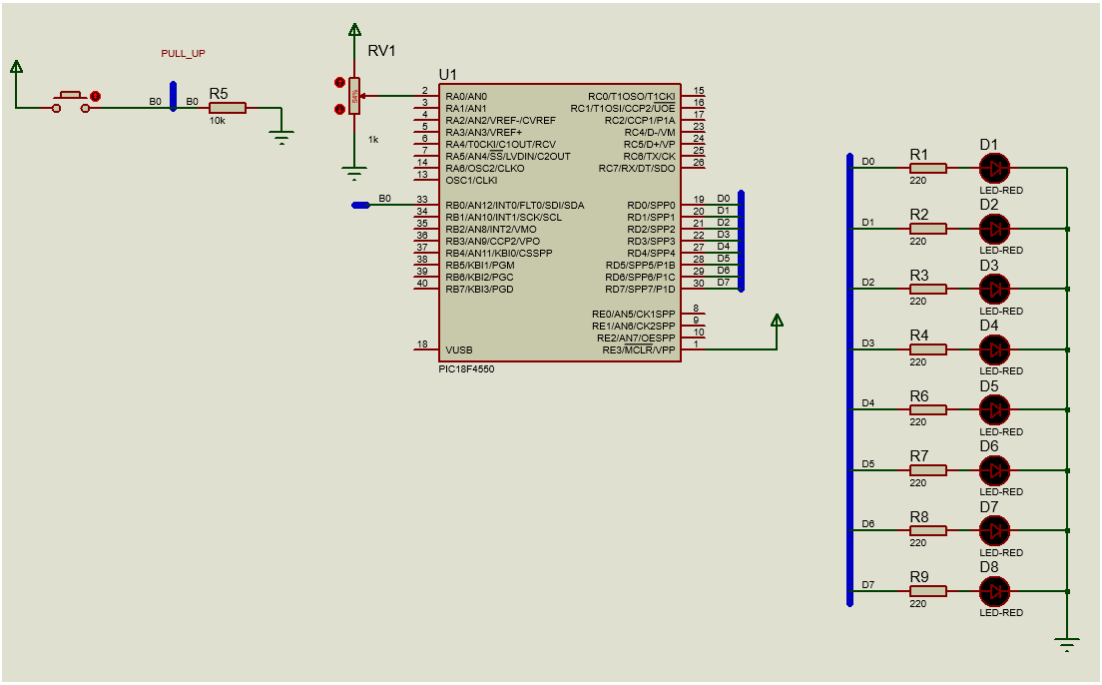
- Implementar las interrupciones utilizando un botón y señal analógicas mediante el pic18f455

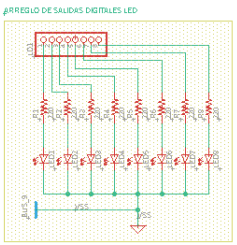
B. Objetivos Específicos

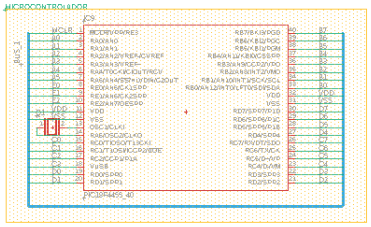
- Implementar un código en lenguaje C utilizando la interrupción externa RB0
- Realizar la conversión analógica y escritura del valor en un puerto del microcontrolador cuando este activa la interrupción.

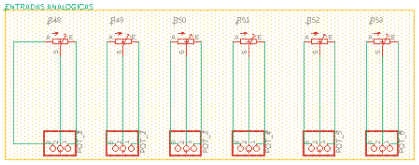
C. Marco Procedimental

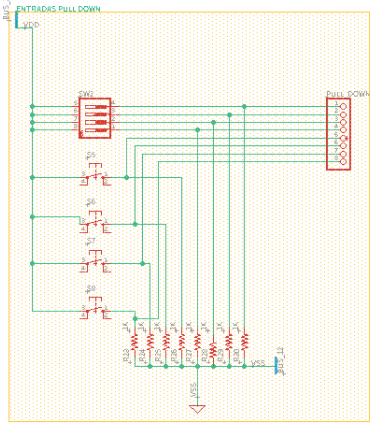
1. Se procede a realizar el primer circuito que sirve como el emisor de información donde se toma de referencia el esquemático de la figura


PULL_UP
B0
R5
10k
RV1
1k
U1
RA0/AN0
RA1/AN1
RA2/AN2/VREF-/CVREF
RA3/AN3/VREF+
RA4/T0CKI/C1OUT/RCV
RA5/AN4/SS/LVDIN/C2OUT
RA6/OSC2/CLKO
OSC1/CLKI
RB0/AN12/INT0/FLT0/SDI/SDA
RB1/AN10/INT1/SCK/SCL
RB2/AN8/INT2/VMO
RB3/AN9/CCP2/VPO
RB4/AN11/KBI0/CSSPP
RB5/KBI1/PGM
RB6/KBI2/PGC
RB7/KBI3/PGD
RC0/T1OSO/T1CKI
RC1/T1OSI/CCP2/UOE
RC2/CCP1/P1A
RC4/D-/VM
RC5/D+/VP
RC6/TX/CK
RC7/RX/DT/SDO
RD0/SPP0
RD1/SPP1
RD2/SPP2
RD3/SPP3
RD4/SPP4
RD5/SPP5/P1B
RD6/SPP6/P1C
RD7/SPP7/P1D
RE0/AN5/CK1SPP
RE1/AN6/CK2SPP
RE2/AN7/OESPP
RE3/MCLR/VPP
VUSB
PIC18F4550
D0
D1
D2
D3
D4
D5
D6
D7
R1
R2
R3
R4
R6
R7
R8
R9
220
D1
D2
D3
D4
D5
D6
D7
D8
LED-RED


ARREGLO DE SALIDAS DIGITALES LED


MICROCONTROLADOR


ENTRADAS ANALOGICAS


ENTRADAS Pull DOWN

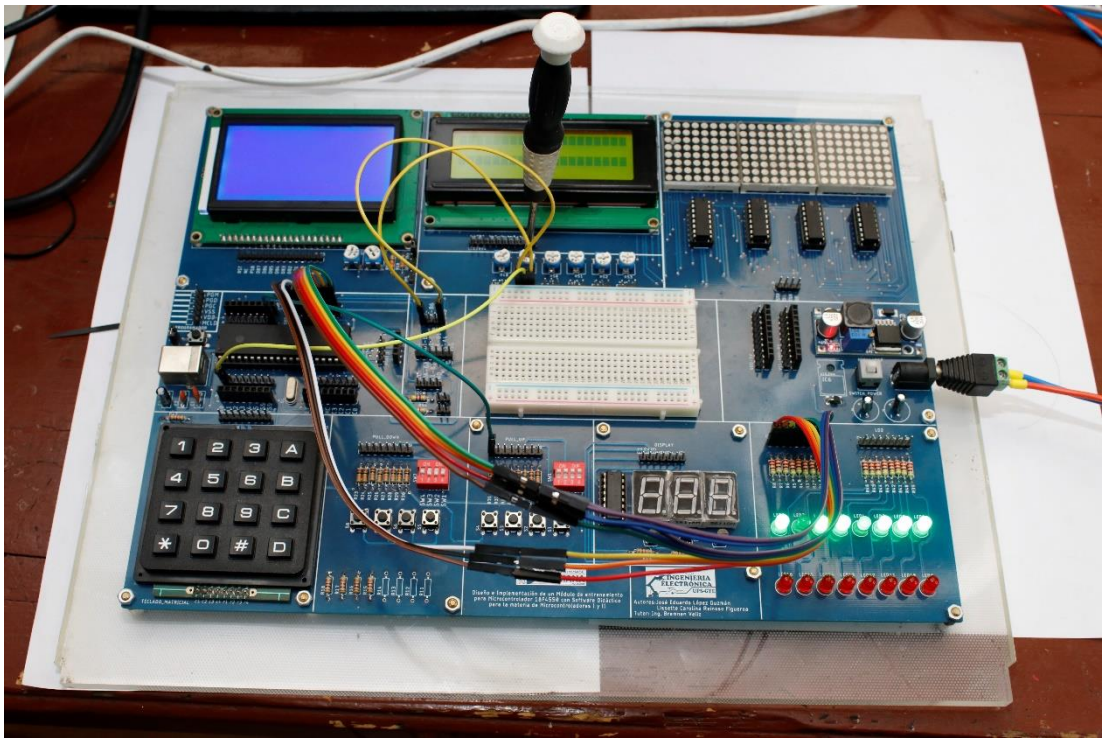


Figura 142. Esquema de conexión de PRÁCTICA 7

MODULO 1 PIC18F4550 – B0	PULL DOWN 0
MODULO 1 PIC18F4550 – D7	LED1 7
MODULO 1 PIC18F4550 – D6	LED1 6
MODULO 1 PIC18F4550 – D5	LED1 5
MODULO 1 PIC18F4550 – D4	LED1 4
MODULO 1 PIC18F4550 – D3	LED1 3
MODULO 1 PIC18F4550 – D2	LED1 2
MODULO 1 PIC18F4550 – D1	LED1 1
MODULO 1 PIC18F4550 – D0	LED1 0
MODULO 1 PIC18F4550 – A0	POT 1 S
MODULO 1 PIC18F4550 – VCC	POT 1 VCC
MODULO 1 PIC18F4550 – GND	POT 1 GND

Tabla 8: Conexiones de PRÁCTICA 7

2. Ejecutar el programa PIC Compiler V 5 de la familia de microchip basado en el lenguaje C como se muestra en la figura.

Figura 143. PIC Compiler V 5 – PRÁCTICA 7

3. Al ejecutar el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_7** como se muestra en la figura-

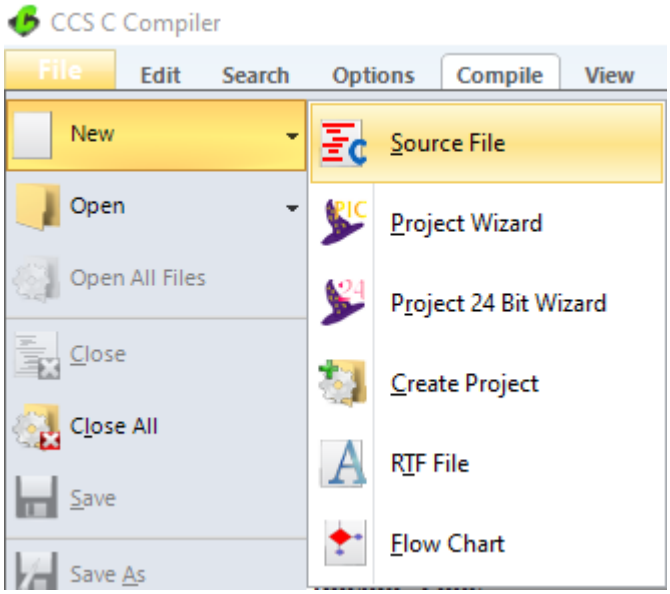


Figura 144. Creación de archivo de código en lenguaje C – PRÁCTICA 7

4. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

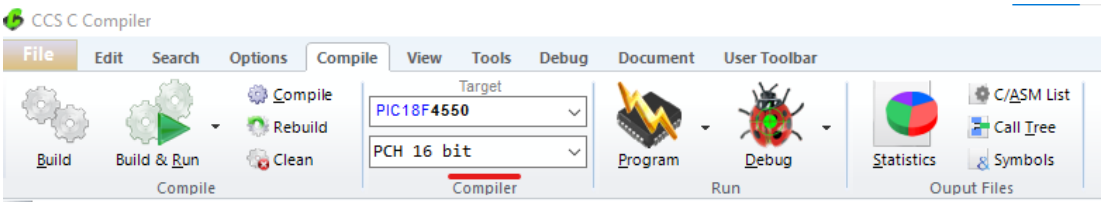


Figura 145. Configuración del microcontrolador en el compilador – PRÁCTICA 7

5. Se procede a llamar a la librería del microcontrolador, definir los fusibles, velocidad del cristal. donde de define la velocidad y los puertos a utilizar, definir la cantidad de bit que se utiliza en el conversor analógico a decimal como se muestra en la figura.

```
#include <18F4550.h>
#device ADC=8
#fuses HS,NOWDT,NOPUT,NOPROTECT
#use delay (clock=20M)
```

Figura 146. Declaración de librerías y registros -PRÁCTICA 7

6. Declarar los puertos y su respectivo registro como se muestra en la figura.

```
#byte portb=0x0f81
#byte portc=0x0f82
#byte portd=0x0f83
```

Figura 147. Declaración de registro del puerto-PRÁCTICA 7

7. Crear el método que se ejecutara cada vez que se realice la interrupción externa en el puerto RB0 por activación de un flanco de bajada, donde la interrupción muestra el encendido y apagado de todas las salidas del puerto d durante un intervalo de 100 ms por 10 ciclos como se muestra en la figura.

```
#int_ext
Interrupcion_Pulso()

{
   ext_int_edge(H_TO_L);
   for(x=1; x<=10; x++)
   {
      portd=255  ;
      delay_ms(100);
     portd=0  ;
      delay_ms(100);
   }
}
```

Figura 148. Método de la interrupción externa RB0– Práctica 7

8. En el método principal se procede a declarar el puerto b0 como entrada y el puerto d como salida desde el d0 al d7 , se habilita la interrupción global y se designa el pin o interrupción externa perteneciente al rb0, declaramos el ADC y el puerto de la conversión analógica A0, en el lazo o bucle de repetición se realiza la lectura del puerto A0 posteriormente se publica ese valor en el puerto d mediante la asignación de la variable valué como se muestra en la figura .

```
void main (void)
{
   int value;
   set_tris_b(0b00000001);
   set_tris_d(0b00000000);
   enable_interrupts(GLOBAL);
   enable_interrupts(INT_EXT);
   setup_adc(ADC_CLOCK_INTERNAL);
   setup_adc_ports(AN0);
   while (true)
   {
   set_adc_channel(0);
   delay_us(10);
   value=read_adc();
   portd=value;
   }
}
```

Figura 149. Método principal del programa– Práctica 7

1. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura.

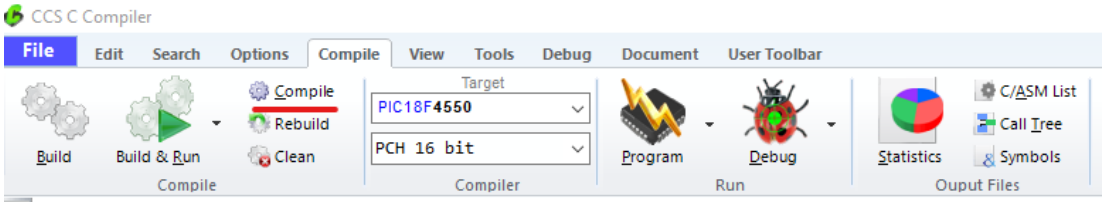


Figura 150. Botón para compilar el programa receptor – Práctica 7

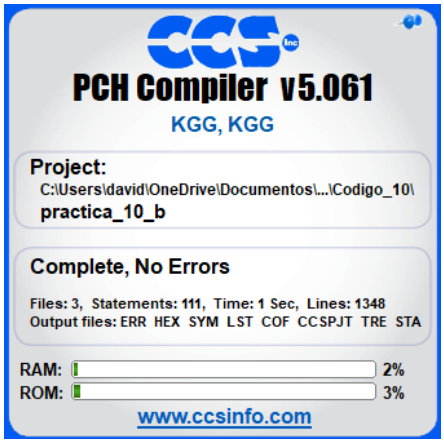


Figura 151. Compilación del programa – Práctica 7

2. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 152. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 7

3. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

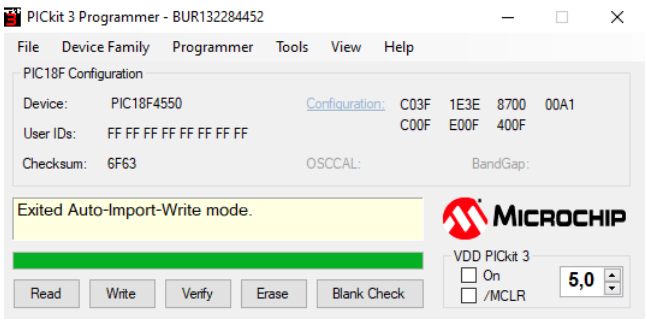


Figura 153. Programación de PIC mediante el Pickit 3 – Práctica 7

d. Registro de Resultados

En la séptima práctica se tiene como resultado la conversión analógica a digital de un valor de voltaje de 0 a 5 voltios mediante la división de voltaje de un potenciómetro, donde al ejecutarse el programa y tener un estímulo en el puerto RB0 se activa una interrupción externa del microcontrolador activándose una señal donde todos los valores del puerto d se encienden comunicación entre dos controladores Pic 18f4550 donde el primero sirve como emisor del valor de un puerto asignado y el segundo sirve como receptor del valor y se muestra en un puerto asignado.

e. Conexiones físicas de la PRÁCTICA

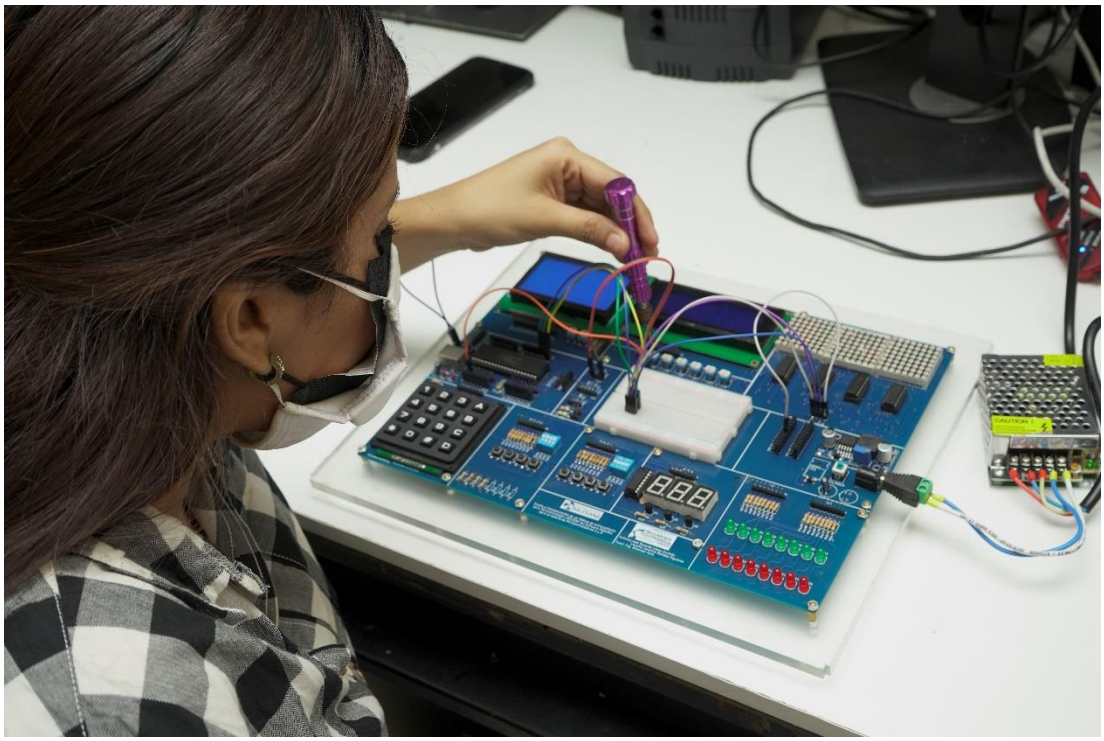


Figura 154. Conexiones físicas - Práctica 7

ANEXO 3.8: RESOLUCIÓN DE PRÁCTICA 8
AUTOMATIZACIÓN INDUSTRIAL

PRÁCTICA 8

NÚMERO DE ESTUDIANTES: 20

DOCENTE

ING. BREMNEN MARINO VELIZ NOBOA MSc.

TIEMPO ESTIMADO: 2 HORAS

TEMA: “CONEXIONES ENTRE 2 PIC 18F4550 MEDIANTE COMUNICACIÓN I2C.”

A. Objetivo General

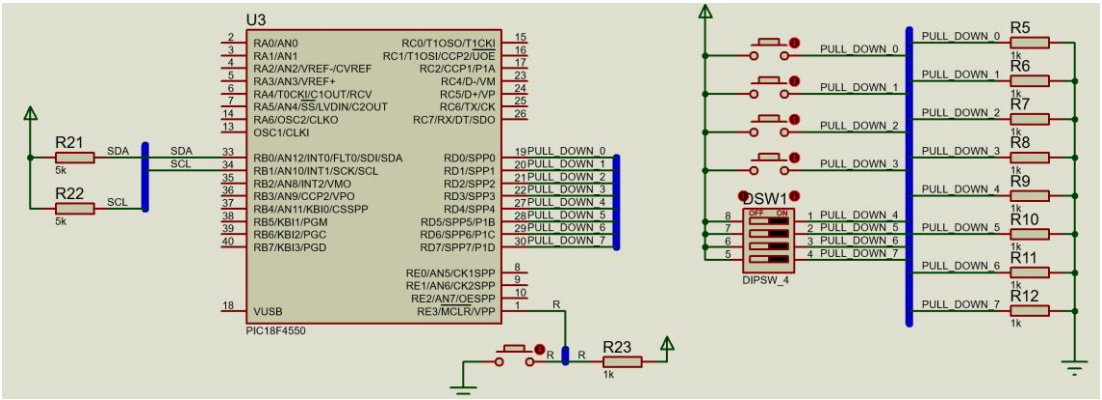
- Implementar un aplicativo para el envío y recepción de datos entre dos microcontroladores 18f4550 por protocolo i2c

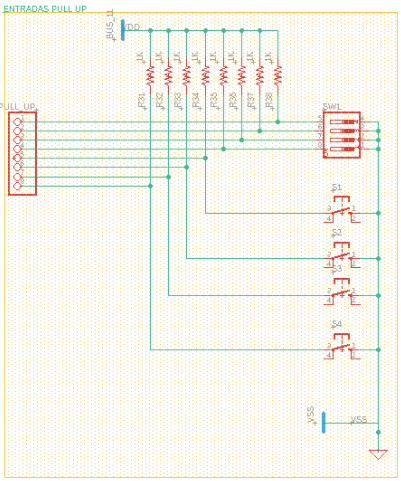
B. Objetivos Específicos

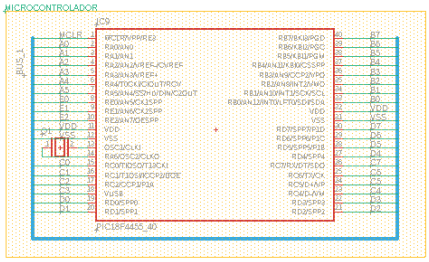
- Programar un microcontrolador como esclavo que recepta los valores del maestro.
- Elaborar el programa de transmisión de datos de un microcontrolador a otro mediante protocolo i2c
- Realizar el cableado respectivo entre los dos módulos didácticos.

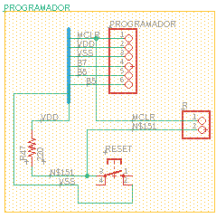
C. Marco Procedimental

1. Se procede a realizar el programa del emisor o maestro en la comunicación i2c, el esquemático de las conexiones se muestra en la figura.









Elaborado por:	**Revisado por:**	**Aprobado por:** Ing. Orlando Barcia Ayala
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

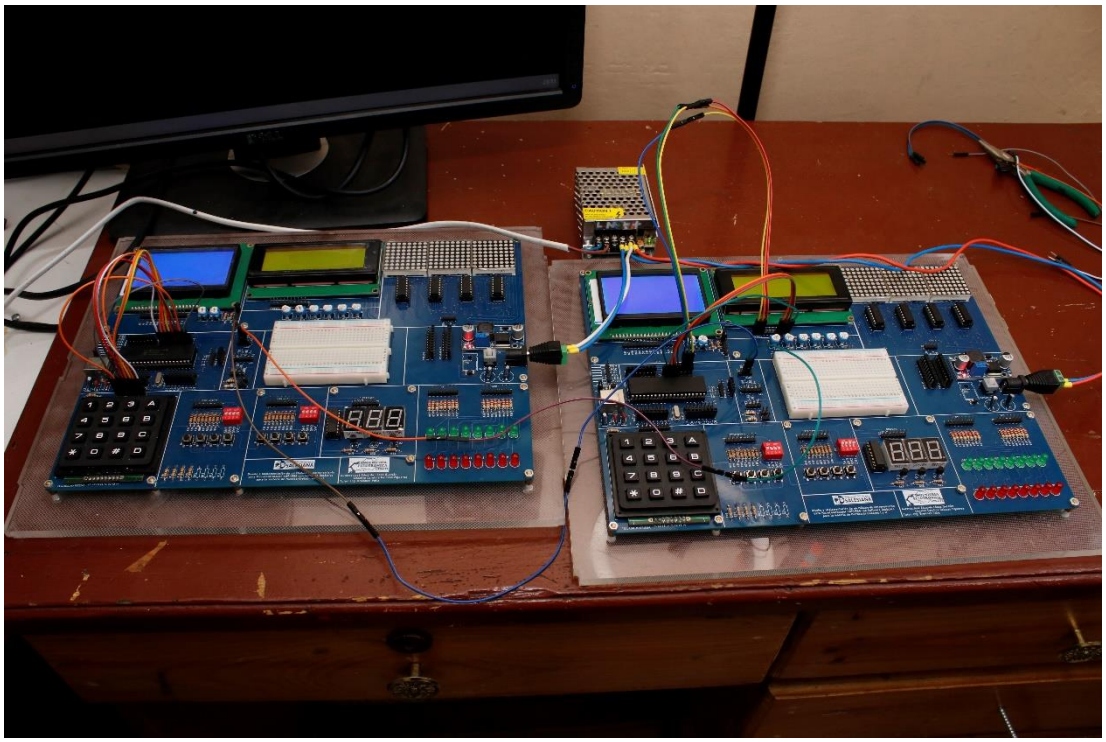


Figura 155. Esquema de conexión del emisor o maestro en PRÁCTICA 8

MODULO 1 PIC18F4550 – B0	SDA
MODULO 1 PIC18F4550 – B1	SCL
MODULO 1 PIC18F4550 – D7	PULL DOWN 7
MODULO 1 PIC18F4550 – D6	PULL DOWN 6
MODULO 1 PIC18F4550 – D5	PULL DOWN 5
MODULO 1 PIC18F4550 – D4	PULL DOWN 4
MODULO 1 PIC18F4550 – D3	PULL DOWN 3
MODULO 1 PIC18F4550 – D2	PULL DOWN 2
MODULO 1 PIC18F4550 – D1	PULL DOWN 1
MODULO 1 PIC18F4550 – D0	PULL DOWN 0
MODULO 1 PIC18F4550 – A0	PULL DOWN 0

Tabla 9: Conexiones de emisor en PRÁCTICA 8

2. Ejecutar el programa PIC Compiler V 5 de la familia de microchip basado en el lenguaje C como se muestra en la figura.

Figura 156. PIC Compiler V 5 – PRÁCTICA 8

3. Al ejecutar el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_8_a** como se muestra en la figura-

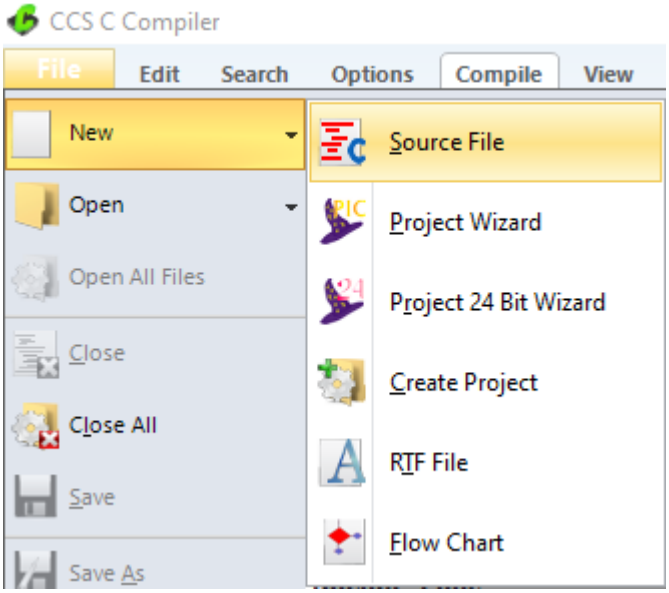


Figura 157. Creación de archivo de código en lenguaje C – PRÁCTICA 8

4. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

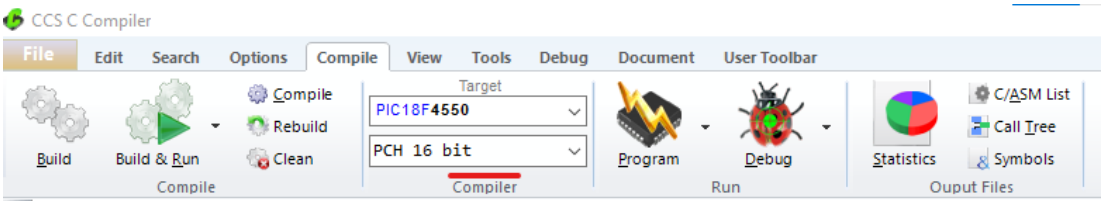


Figura 158. Configuración del microcontrolador en el compilador – PRÁCTICA 8

5. Se procede a llamar a la librería del microcontrolador, definir los fusibles, velocidad del cristal donde de define la velocidad y los puertos a utilizar como se muestra en la figura.

```
#include <18f4550.h>            //Declaración de libreria PIC18F4550
#use delay(clock=20Mhz)          //Declaracion del cristal de 20 Mhz
#FUSES NOWDT                     //No Watch Dog Timer
#FUSES HS                        //High speed Osc (> 4mhz)
#FUSES PUT                       //Power Up Timer
#FUSES NOPROTECT                 //Code not protected from reading
#FUSES NOBROWNOUT                //No brownout reset
#FUSES NOMCLR                    //Master Clear pin enabled
#FUSES NOLVP                     //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD
```

Figura 159. Declaración de librerías y registros -PRÁCTICA 8

6. Declarar el comando para utilizar el Spi y declarar los pines y velocidad a utilizar y los puertos con su respectivo registro como se muestra en la figura.

```
#use I2C(master,sda=PIN_B0, fast=450000, scl=PIN_B1, force_sw) //Declaración I2C
#BYTE portb = 0xF81    //Dirección de Puerto B en memoria.
#BYTE portc = 0xF82    //Dirección de Puerto C en memoria.
#BYTE portd = 0xF83    //Dirección de Puerto D en memoria.
```

Figura 160. Declaración de registro del puerto-PRÁCTICA 5

7. Posteriormente se declara el método principal del código donde se configura el puerto d como entrada con el comando **set_tris_d(255)** y en el lazo de repetición while se declara que el valor del puerto d se guarde en una variable llamada **data,** se inicia el i2c, se escribe la dirección de destino **0xa0**(dirección del esclavo) seguido del dato a enviar, se detiene él envió indicando el fin de la trama, se espera 100 milisegundos hasta la

.

```
void main()
{
   int16 data;//variable donde se almacena información del puerto
   set_tris_d(255);  //declaración del puerto d como entrada
      while (TRUE)
   {
      data=input_d();//guarda los datos de entrada del puerto D en la variable data
     i2c_start ();//condicion de arranque del spi
      i2c_write (0xa0);//direccion del eclavo que recivira la informacón
      i2c_write (data);//envia la informacion proveniente del puerto D
      i2c_stop ();//termina la comunicación del maestro

    i2c_stop ();//termina la comunicación del maestro
    delay_ms(100);
}
}
```

Figura 161. Segmento principal emisor – Práctica 1

8. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura 133 Y 135.

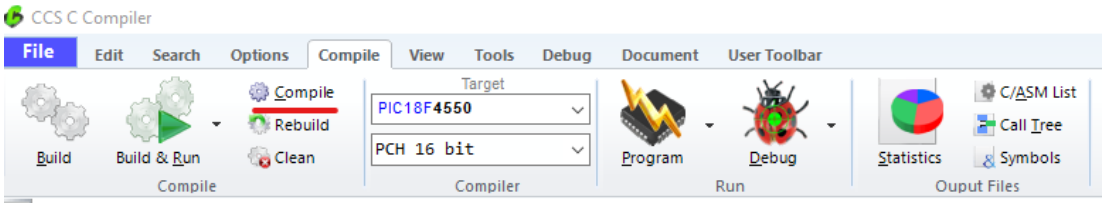


Figura 162. Botón para compilar el programa emisor – Práctica 1

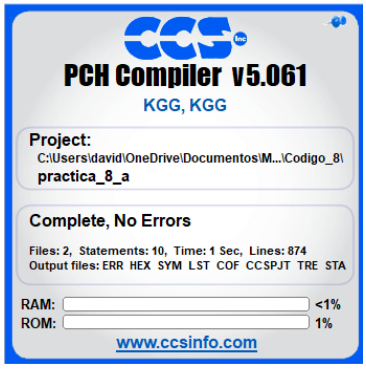


Figura 163. Compilación del programa emisor – Práctica 1

9. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 164. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 8

10. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

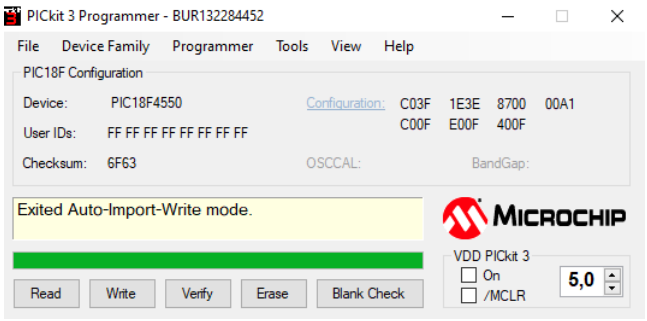


Figura 165. programación de PIC mediante el Pickit 3 – Práctica 8

11. Se procede a realizar el programa del emisor o maestro en la comunicación i2c, el esquemático de las conexiones se muestra en la figura.

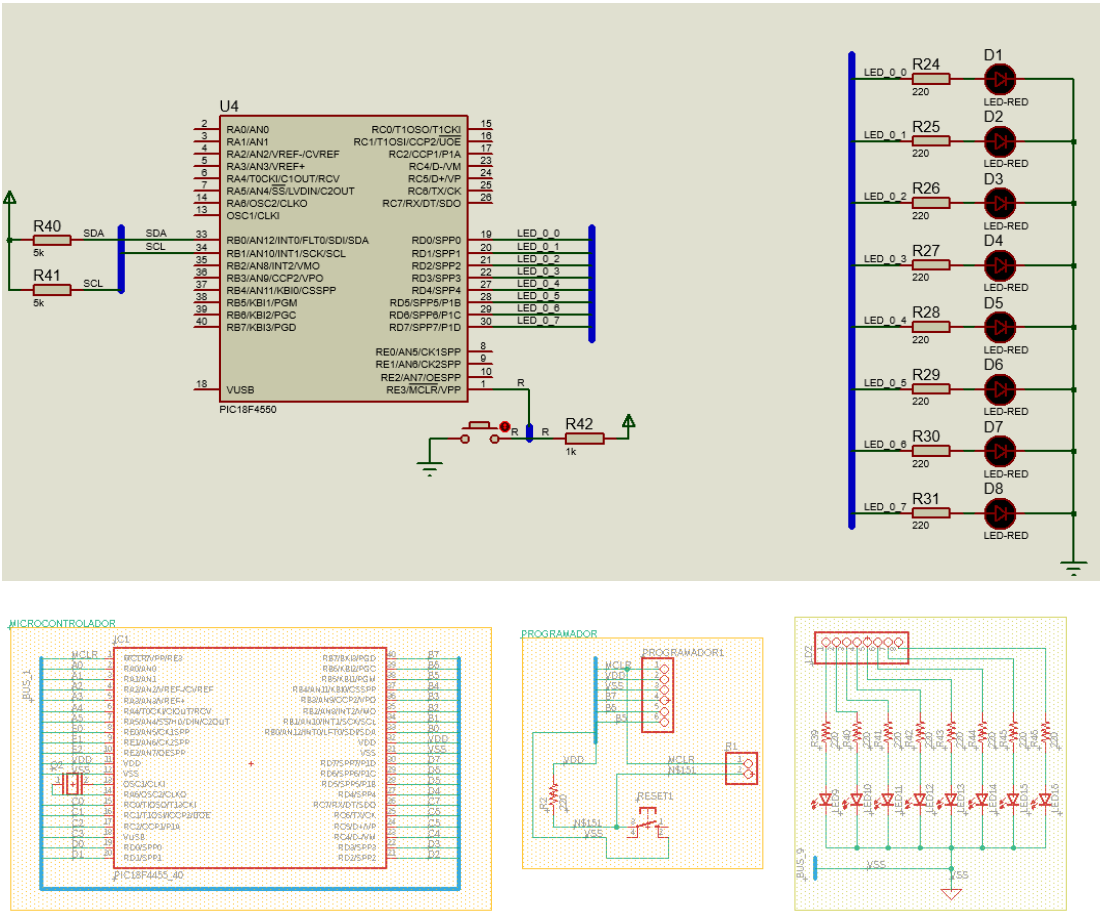


Figura 166. Esquema de conexión del receptor o esclavo en PRÁCTICA 8

MODULO 1 PIC18F4550 – B0	SDA
MODULO 1 PIC18F4550 – B1	SCL
MODULO 1 PIC18F4550 – D7	LED 1 7
MODULO 1 PIC18F4550 – D6	LED 1 6
MODULO 1 PIC18F4550 – D5	LED 1 5
MODULO 1 PIC18F4550 – D4	LED 1 4
MODULO 1 PIC18F4550 – D3	LED 1 3
MODULO 1 PIC18F4550 – D2	LED 1 2
MODULO 1 PIC18F4550 – D1	LED 1 1
MODULO 1 PIC18F4550 – D0	LED 1 0

Tabla 10: Conexiones de receptor en PRÁCTICA 8

12. Se procede a crear un archivo nuevo en el pic c Compiler para programar el código del receptor, en el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_8_b** como se muestra en la figura-

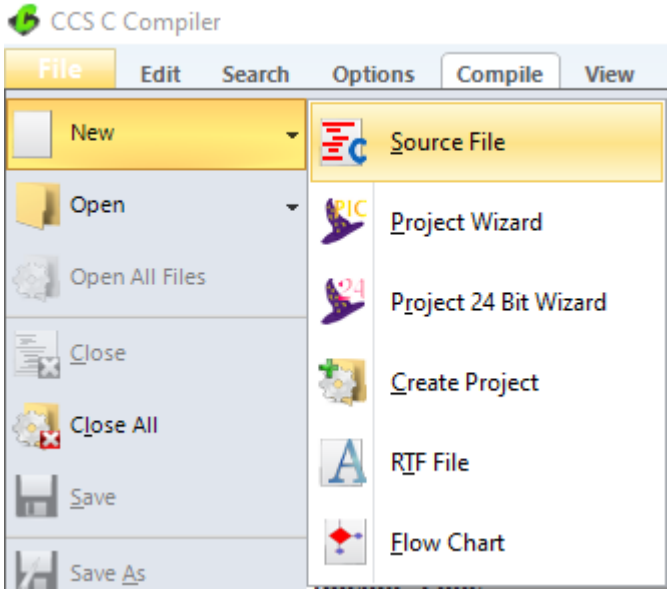


Figura 167. Creación de archivo de código receptor en lenguaje C – PRÁCTICA 8

13. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

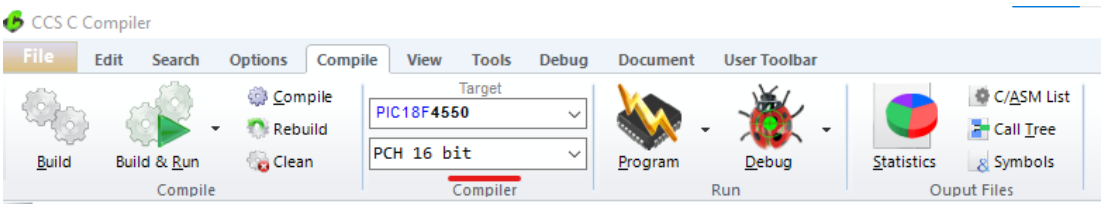


Figura 168. Configuración del microcontrolador en el compilador – PRÁCTICA 8

14. Se procede a llamar a la librería del microcontrolador, definir los fusibles, velocidad del cristal donde de define la velocidad y los puertos a utilizar como se muestra en la figura.

```
#include <18f4550.h>            //Declaración de libreria PIC18F4550
#use delay(clock=20Mhz)          //Declaracion del cristal de 20 Mhz
#FUSES NOWDT                     //No Watch Dog Timer
#FUSES HS                        //High speed Osc (> 4mhz)
#FUSES PUT                       //Power Up Timer
#FUSES NOPROTECT                 //Code not protected from reading
#FUSES NOBROWNOUT                //No brownout reset
#FUSES NOMCLR                    //Master Clear pin enabled
#FUSES NOLVP                     //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD
```

Figura 169. Declaración de librerías y registros -PRÁCTICA 8

15. Declarar el comando para utilizar el Spi y declarar los pines y velocidad a utilizar y los puertos con su respectivo registro como se muestra en la figura.

```
#use I2C(slave, SDA=PIN_B0, SCL=PIN_B1,  address=0xa0)//Declaración I2C
#BYTE portb = 0xF81   //Dirección de Puerto B en memoria.
#BYTE portc = 0xF82   //Dirección de Puerto C en memoria.
#BYTE portd = 0xF83   //Dirección de Puerto D en memoria.
```

Figura 170. Declaración de registro del puerto-PRÁCTICA 8

16. Posteriormente se declara el método principal del código donde se configura el puerto d como salida con el comando **set_tris_d(0)** y en el lazo de repetición while se declara el comando **i2c_poll** de manera que al leer un dato en el puerto i2c del microcontrolador tomara el valor mediante el comando **i2c_read** y se asignara el valor al puerto d, caso contrario terminara la conexión y entra a modo espera que tenga un dato en el puerto i2c como se muestra en la figura.

```
void main()
{
     int8 dato;//variable donde se almacena el dato del i2c
   set_tris_d(0); //asignacion de puerto d como salida
   portd=0;//declarar bit en bajo en todo el puerto
      while (TRUE)
   {
     if(i2c_poll())
      {
         dato=i2c_read();//lee los datos enviados por el maestro
         if (dato!=0xa0){//desprecia la direccion para evitar leerla como dato
            portd=dato;//envia los datos al puerto D de salida
         }
         i2c_stop ();//termina la comunicación del esclavo
      }
      else
      {
         i2c_stop();//termina la comunicación del esclavo
      }
}
}
```

Figura 171. Segmento principal receptor – Práctica 1

17. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura 133 Y 135.

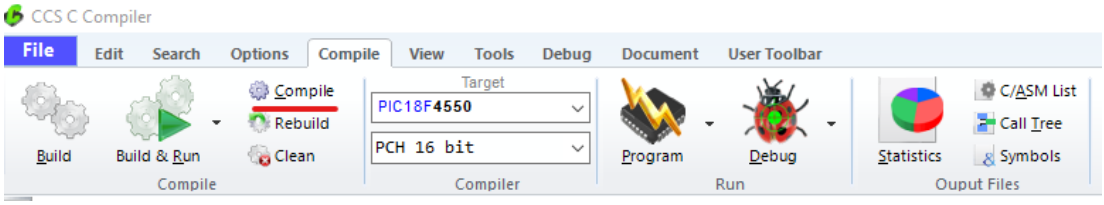


Figura 172. Botón para compilar el programa receptor – Práctica 1

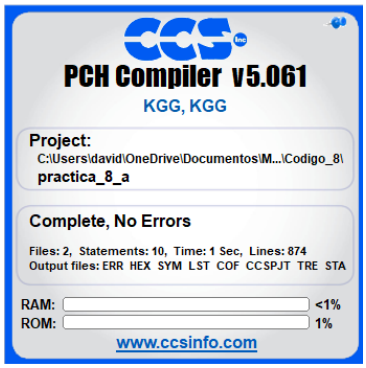


Figura 173. Compilación del programa receptor – Práctica 1

18. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 174. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 8

19. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

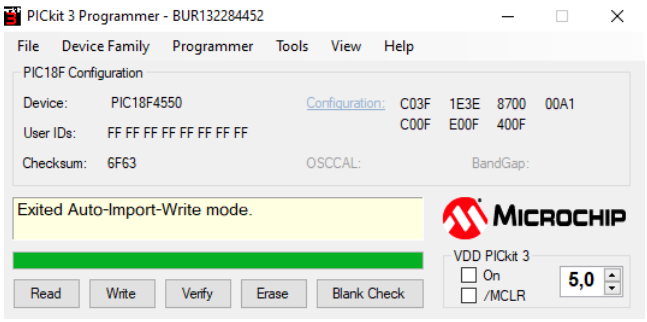


Figura 175. Programación de PIC mediante el Pickit 3 – Práctica 8

D. Registro de Resultados

En la octava PRÁCTICA se tiene como resultado la implementación de la comunicación entre dos controladores Pic 18f4550 donde uno será el maestro que enviará los valores digitales de entradas en el puerto D y el segundo será el esclavo sirve como esclavo receptando los valores y escribiendo los valores digitales de salidas en el puerto D.

E. Conexiones físicas de la PRÁCTICA

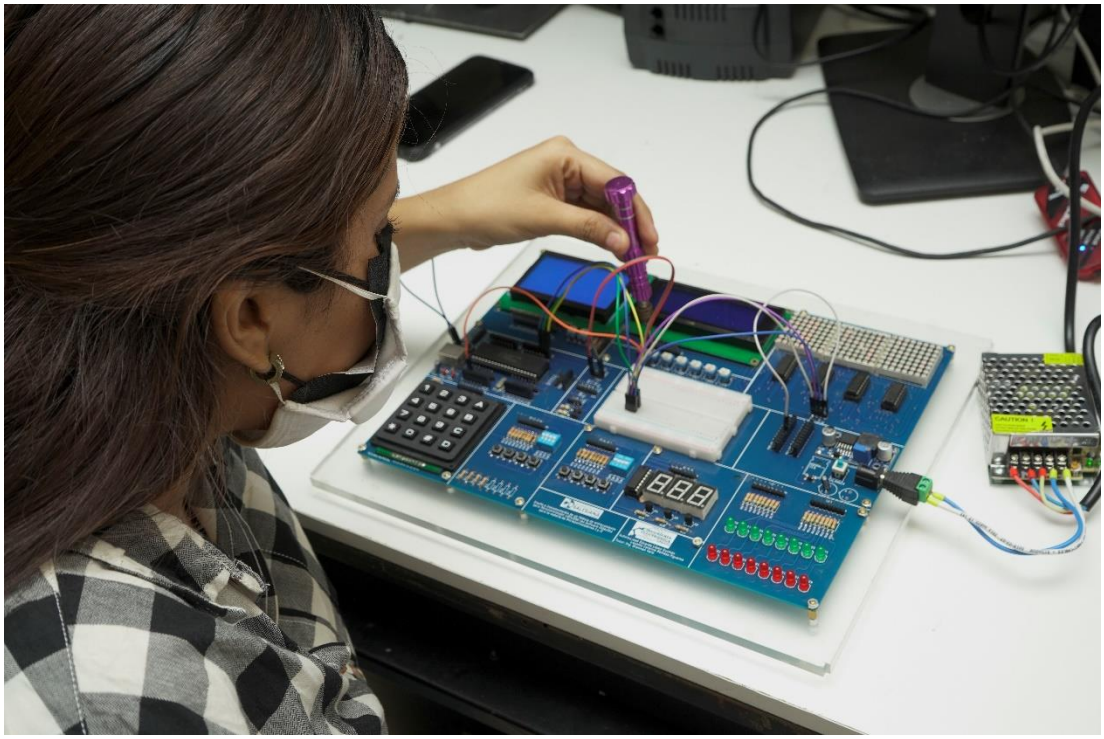


Figura 176. Conexiones físicas - Práctica 8

ANEXO 3.9: RESOLUCIÓN DE PRÁCTICA 9
AUTOMATIZACIÓN INDUSTRIAL

PRÁCTICA 9

NÚMERO DE ESTUDIANTES: 20

DOCENTE

ING. BREMNEN MARINO VELIZ NOBOA MSc.

TIEMPO ESTIMADO: 2 HORAS

TEMA: "ENVÍO Y RECEPCIÓN DE DATOS UTILIZANDO LCD 20X4, TECLADO MATRICIAL MEDIANTE EL PIC18F4550"

A. Objetivo General

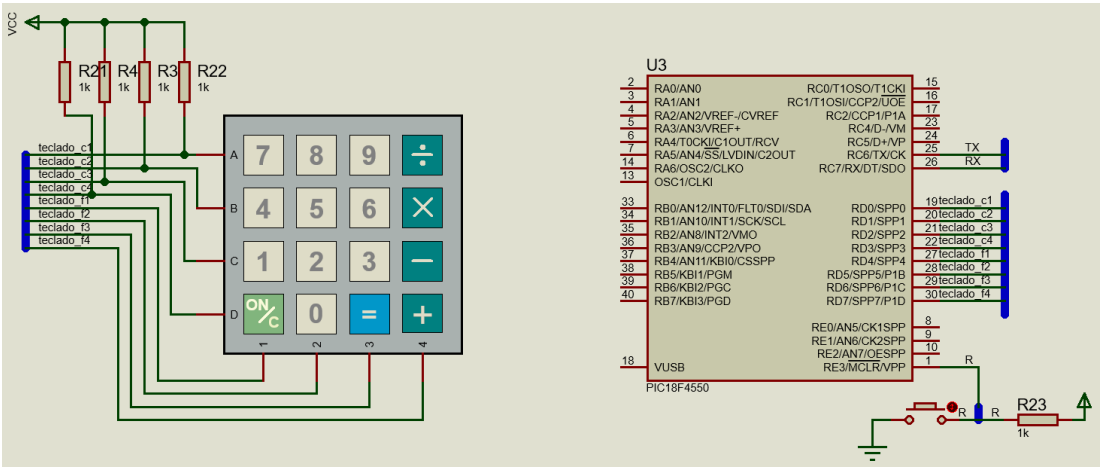
- Implementar un aplicativo para el envío y recepción de datos entre dos microcontroladores 18f4550 utilizando UART.

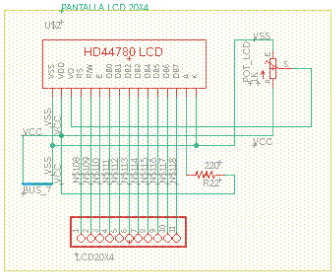
B. Objetivos Específicos

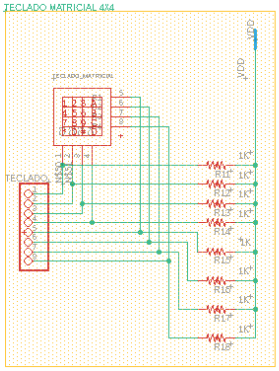
- Realizar él envió de datos obtenidos de un teclado matricial 4x4 por el UART.
- Realizar la conexión entre dos módulos Pic 18F4550 por UART.
- Realizar la recepción de datos y que se visualicen en un LCD 4x20.

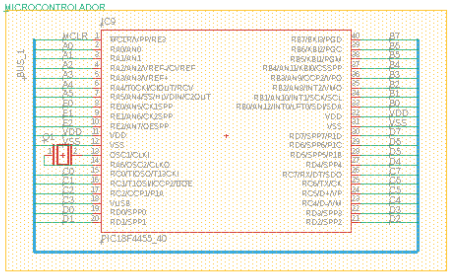
C. Marco Procedimental

1. Se procede a realizar el programa del emisor que envía los valores del teclado matricial 4x4 por una comunicación UART, el esquemático de las conexiones se muestra en la figura.









Elaborado por:	**Revisado por:**	**Aprobado por:** Ing. Orlando Barcia Ayala
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

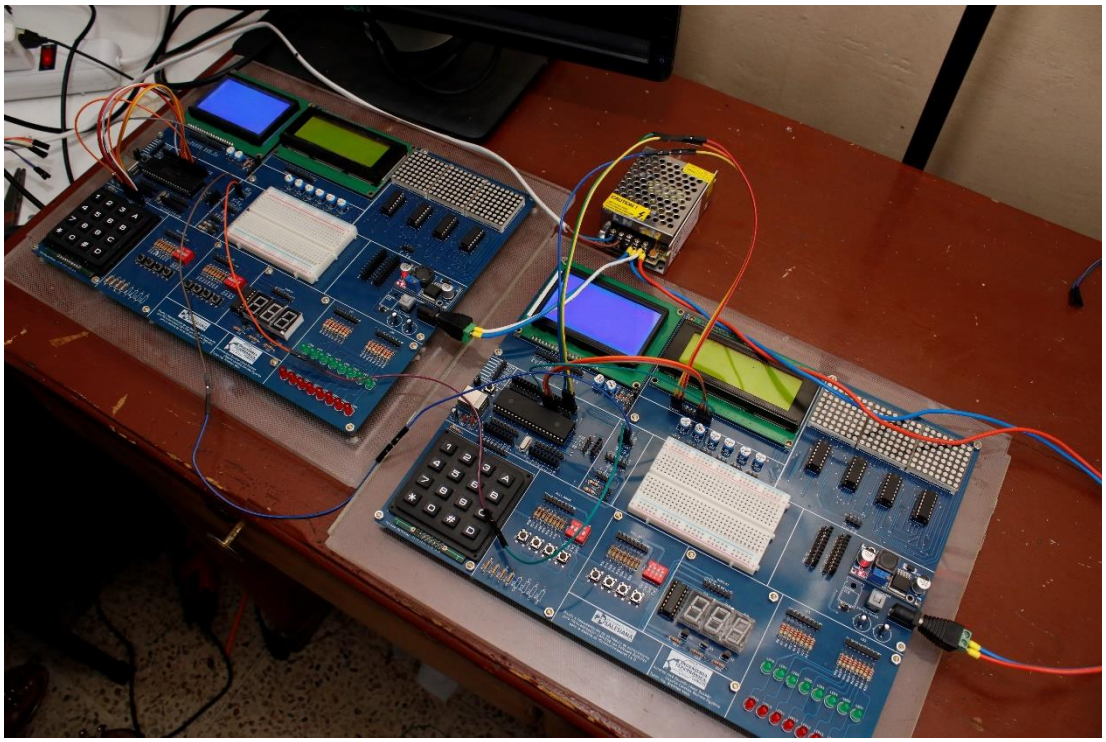


Figura 177. Esquema de conexión del emisor o maestro en PRÁCTICA 9

PIC18F4550 – C7	RX
PIC18F4550 – C6	TX
PIC18F4550 – D7	TECLADO f4
PIC18F4550 – D6	TECLADO f3
PIC18F4550 – D5	TECLADO f2
PIC18F4550 – D4	TECLADO f1
PIC18F4550 – D3	TECLADO c4
PIC18F4550 – D2	TECLADO c3
PIC18F4550 – D1	TECLADO c2
PIC18F4550 – D0	TECLADO c1

Tabla 11: Conexiones de PRÁCTICA 9

2. Ejecutar el programa PIC Compiler V 5 de la familia de microchip basado en el lenguaje C como se muestra en la figura.

Figura 178. PIC Compiler V 5 – PRÁCTICA 9

3. Al ejecutar el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_9_a** como se muestra en la figura-

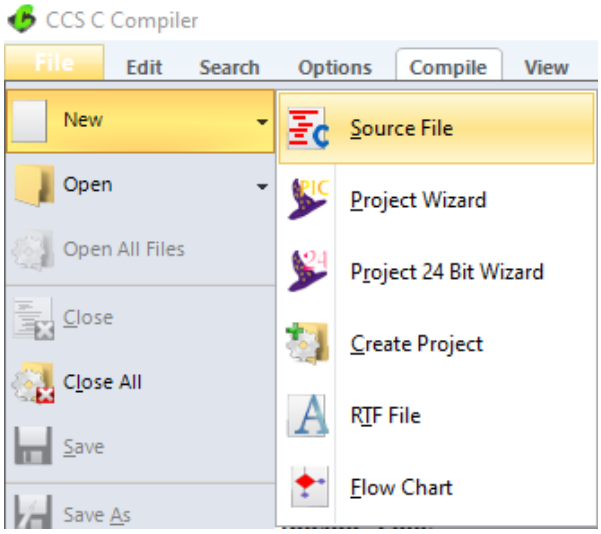


Figura 179. Creación de archivo de código en lenguaje C – PRÁCTICA 9

4. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

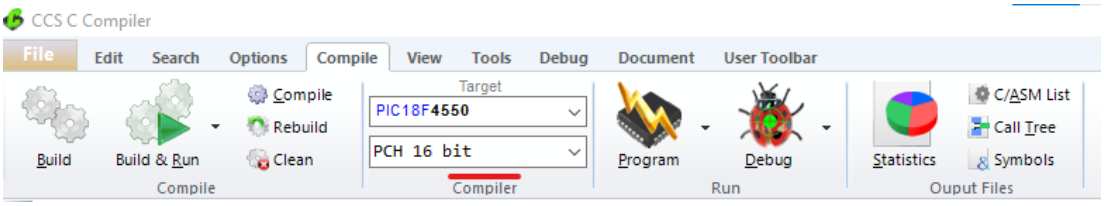


Figura 180. Configuración del microcontrolador en el compilador – PRÁCTICA 9

5. Se procede a llamar a la librería del microcontrolador, definir los fusibles, velocidad del cristal donde de define la velocidad y los puertos a utilizar como se muestra en la figura.

```
#include <18F4550.h>            //No Watch Dog Timer
#fuses INTRC_IO                 //clock fusible
#FUSES NOWDT                    //No Watch Dog Timer
#FUSES HS                       //High speed Osc (> 4mhz)
#FUSES PUT                      //Power Up Timer
#FUSES NOPROTECT                //Code not protected from reading
#FUSES NOBROWNOUT               //No brownout reset
#FUSES NOMCLR                   //Master Clear pin enabled
#FUSES NOLVP                    //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD
#use delay(clock = 20000000)
```

Figura 181. Declaración de librerías y registros -PRÁCTICA 9

6. Declarar el comando para utilizar el UART y declarar los pines y velocidad a utilizar y los puertos con su respectivo registro como se muestra en la figura.

```
#use rs232(baud=9600,xmit=pin_c6, rcv=pin_c7, bits=8, parity=N)//Declaracion UART
#include <kb4.c>      //Libreria del teclado
#BYTE portb = 0xF81   //Dirección de Puerto B en memoria.
#BYTE portc = 0xF82   //Dirección de Puerto C en memoria.
#BYTE portd = 0xF83   //Dirección de Puerto D en memoria.
//Declaración de pines del lcd en el puerto D
#define LCD_RS_PIN      PIN_B0
#define LCD_RW_PIN      PIN_B1
#define LCD_ENABLE_PIN  PIN_B2
#define LCD_DATA4       PIN_B4
#define LCD_DATA5       PIN_B5
#define LCD_DATA6       PIN_B6
#define LCD_DATA7       PIN_B7
#include <lcd.c>  //Declaración de libreria
```

Figura 182. Declaración de registro del puerto-PRÁCTICA 9

7. Posteriormente se declara el método principal del código donde se configura las variables para la tecla y el valor a enviarse, posterior se inicia el teclado mediante el comando **kbd_init,** posteriormente se obtiene el valor de teclado con el comando **kbd_get**, se convierte el valor a decimal y se envía por el serial mediante el comando **printf** y posterior reinicio de variable k y espera de 100 milisegundo como se muestra en la figura.

```
void main(){
   char k;                    //Variable para dato del teclado
   int t;                     //Variable para a enviarse
   Kbd_init();                //Inicio de teclado
   lcd_init();// inicio de libreria LCD
   while(TRUE)                //Lazo principal
      {
         k=kbd_getc();        //obtención de tecla presionada
         if(k!=0){            //Lazo de consulta cuando se presione tecla
         t=k-48;              //Conversion de valor caracter a numerico
         printf("%d\r\n",t);  //Envio de dato numerico por UART
         lcd_gotoxy(1,1);//Se configura el puntero del lcd
         printf(lcd_putc,"Dato: %d",t);//Se escribe en el lcd
         k=0;                 //Limpieza de valor para nueva lectura
         delay_ms(100);       //Espera de 100 mili segundo hasta siguiente iteración
      }
   }
}
```

Figura 183. Segmento principal emisor – Práctica 1

8. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura 133 Y 135.

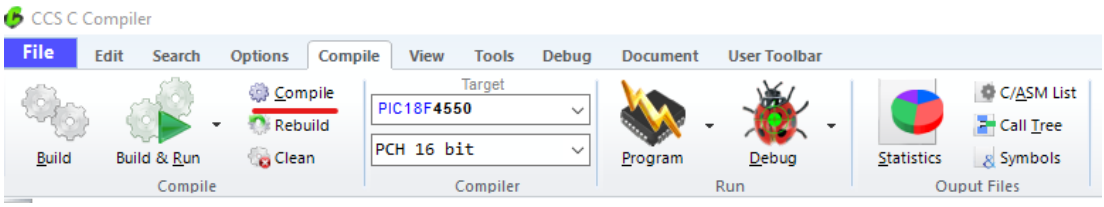


Figura 184. Botón para compilar el programa emisor – Práctica 1

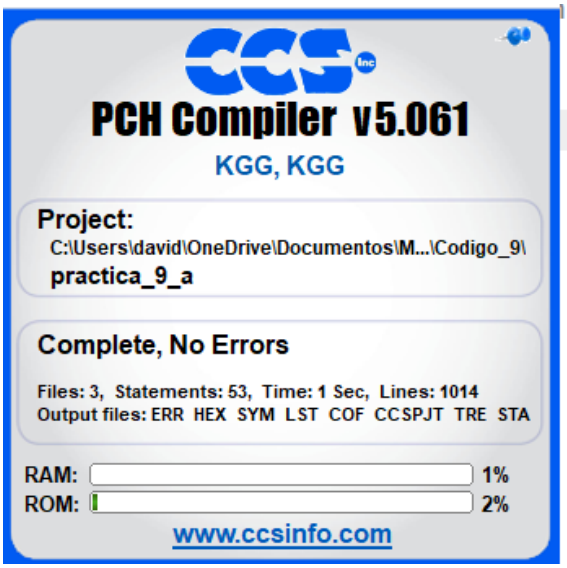


Figura 185. Compilación del programa emisor – Práctica 9

9. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 186. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 9

10. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

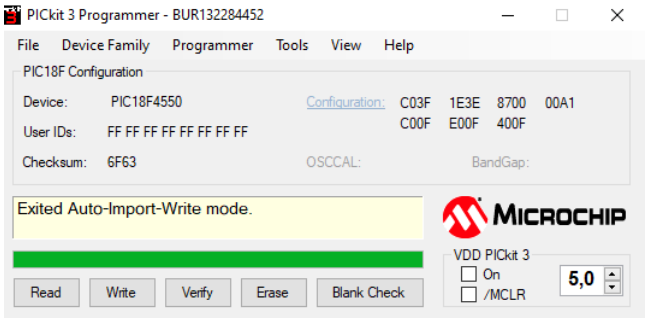


Figura 187. programación de PIC mediante el Pickit 3 – Práctica 9

11. Se procede a realizar el programa receptor que recibe el valor del otro microcontrolador y se visualizara en una pantalla Lcd, el esquemático de las conexiones se muestra en la figura.

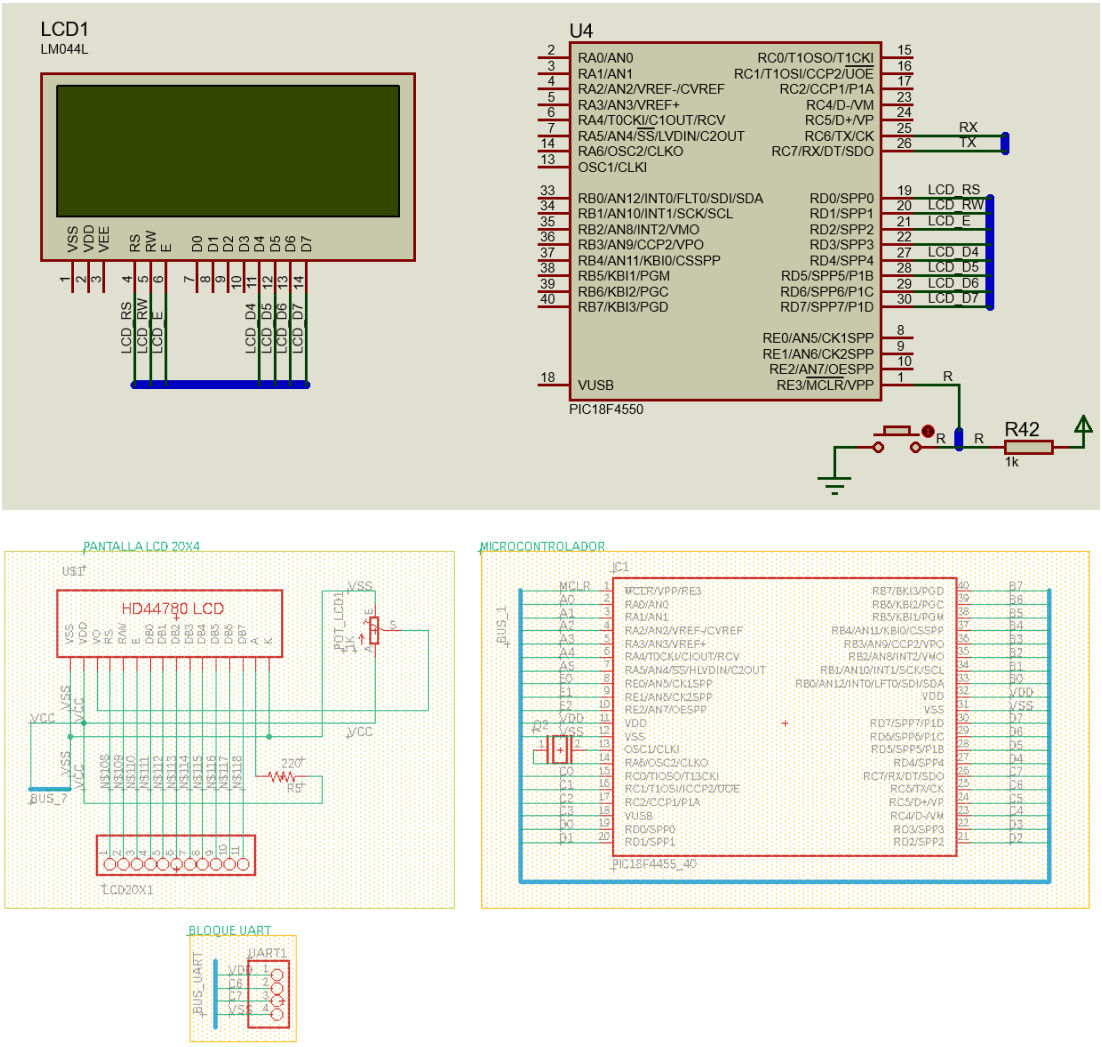


Figura 188. Esquema de conexión del receptor en PRÁCTICA 9

PIC18F4550 – B7	LCD D7
PIC18F4550 – B6	LCD D6
PIC18F4550 – B5	LCD D5
PIC18F4550 – B4	LCD D4
PIC18F4550 – B2	LCD E
PIC18F4550 – B1	LCD RW

PIC18F4550 – B0	LCD RS
PIC18F4550 – C6	RX
PIC18F4550 – C7	TX

Tabla 12: Conexiones de PRÁCTICA 9

12. Se procede a crear un archivo nuevo en el pic c Compiler para programar el código del receptor, en el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_9_b** como se muestra en la figura-

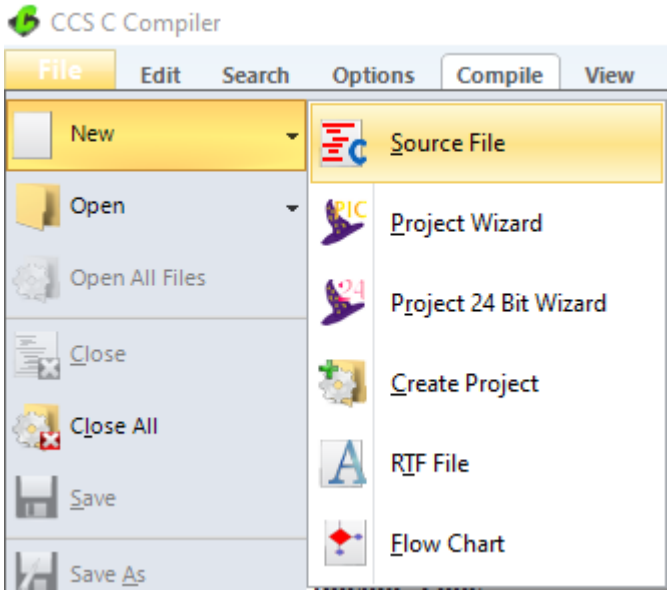


Figura 189. Creación de archivo de código receptor en lenguaje C – PRÁCTICA 9

13. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

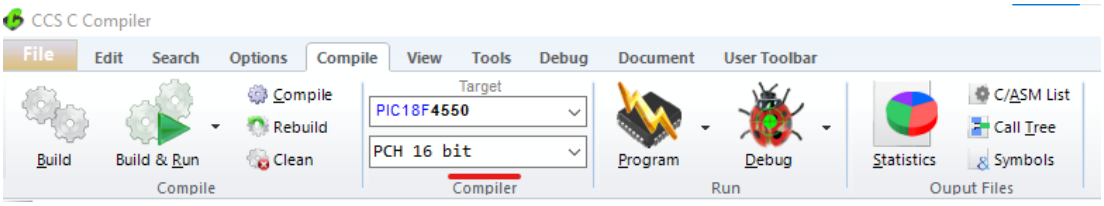


Figura 190. Configuración del microcontrolador en el compilador – PRÁCTICA 8

14. Se procede a llamar a la librería del microcontrolador, definir los fusibles, velocidad del cristal donde de define la velocidad, los puertos a utilizar como se muestra en la figura.

```
#include <18f4550.h>            //Declaración de libreria PIC18F4550
#use delay(clock=20Mhz)         //Declaracion del cristal de 20 Mhz
#FUSES NOWDT                    //No Watch Dog Timer
#FUSES HS                       //High speed Osc (> 4mhz)
#FUSES PUT                      //Power Up Timer
#FUSES NOPROTECT                //Code not protected from reading
#FUSES NOBROWNOUT               //No brownout reset
#FUSES NOMCLR                   //Master Clear pin enabled
#FUSES NOLVP                    //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD
```

Figura 191. Declaración de librerías y registros -PRÁCTICA 8

15. Declarar el comando para utilizar el UART y declarar los pines y velocidad a utilizar, declara la librería del Lcd y definir los pines con su respectivo bit del puerto como se muestra en la figura.

```
#use rs232(baud=9600,xmit=pin_c6, rcv=pin_c7, bits=8, parity=N)//Declaración de UART
//Declaración de pines del lcd en el puerto D
#define LCD_RS_PIN      PIN_D0
#define LCD_RW_PIN      PIN_D1
#define LCD_ENABLE_PIN  PIN_D2
#define LCD_DATA4       PIN_D4
#define LCD_DATA5       PIN_D5
#define LCD_DATA6       PIN_D6
#define LCD_DATA7       PIN_D7
#include <lcd.c>  //Declaración de libreria
#BYTE portb = 0xF81   //Dirección de Puerto B en memoria.
#BYTE portc = 0xF82   //Dirección de Puerto C en memoria.
#BYTE portd = 0xF83   //Dirección de Puerto D en memoria.
```

Figura 192. Declaración de registro del puerto y librería del LCD -PRÁCTICA 9

16. Posteriormente se declara el método principal del código donde se configura las variables para el dato del UART y una variable numérica que será visualizada en el Lcd, posterior se inicia la pantalla mediante el comando **lcd_init,** posteriormente se obtiene el valor del UART con el comando **getc**, se convierte el valor a decimal y se envía visualiza en el Lcd. y espera de 100 milisegundo como se muestra en la figura.

```
void main(){
   char k;//Variable de recepción del UART
   int t;
   lcd_init();
   while(TRUE)
      {
         k=getc();// se obtiene el valor mediante el UART
         if(k>=48){
         t=k-48;//Se convierte el valor char a numerico
         }
         lcd_gotoxy(1,1);//Se configura el puntero del lcd
         printf(lcd_putc,"Dato: %d",t);//Se escribe en el lcd
         delay_ms(100); //Espera de 100 milisegundos
      }
   }
```

Figura 193. Segmento principal receptor – Práctica 9

17. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura.

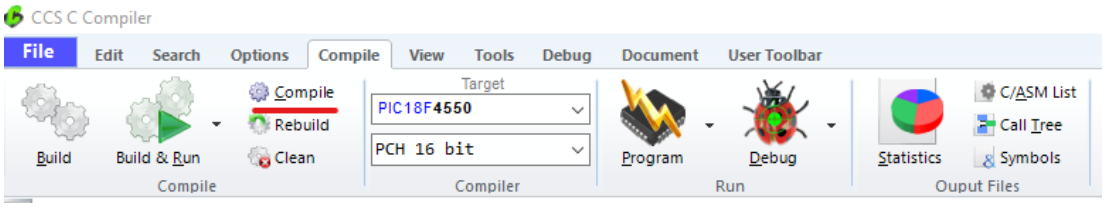


Figura 194. Botón para compilar el programa receptor – Práctica 1

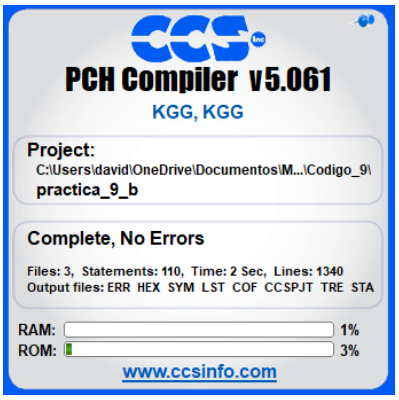


Figura 195. Compilación del programa receptor – Práctica 9

18. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 196. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 8

19. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

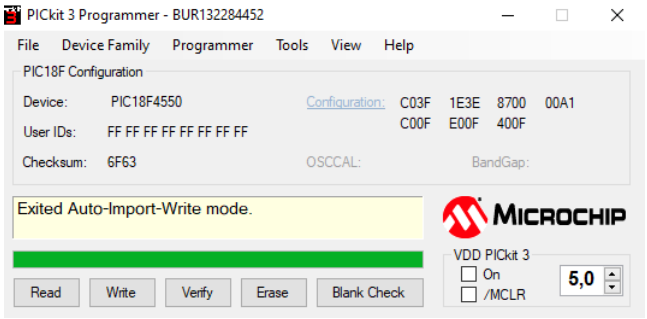


Figura 197. Programación de PIC mediante el Pickit 3 – Práctica 9

D. Registro de Resultados

En la novena práctica se tiene como resultado la comunicación entre dos controladores Pic 18f4550 donde el primero envía los valores de un teclado matricial y el receptor muestra los valores en un LCD.

E. Conexiones físicas de la PRÁCTICA

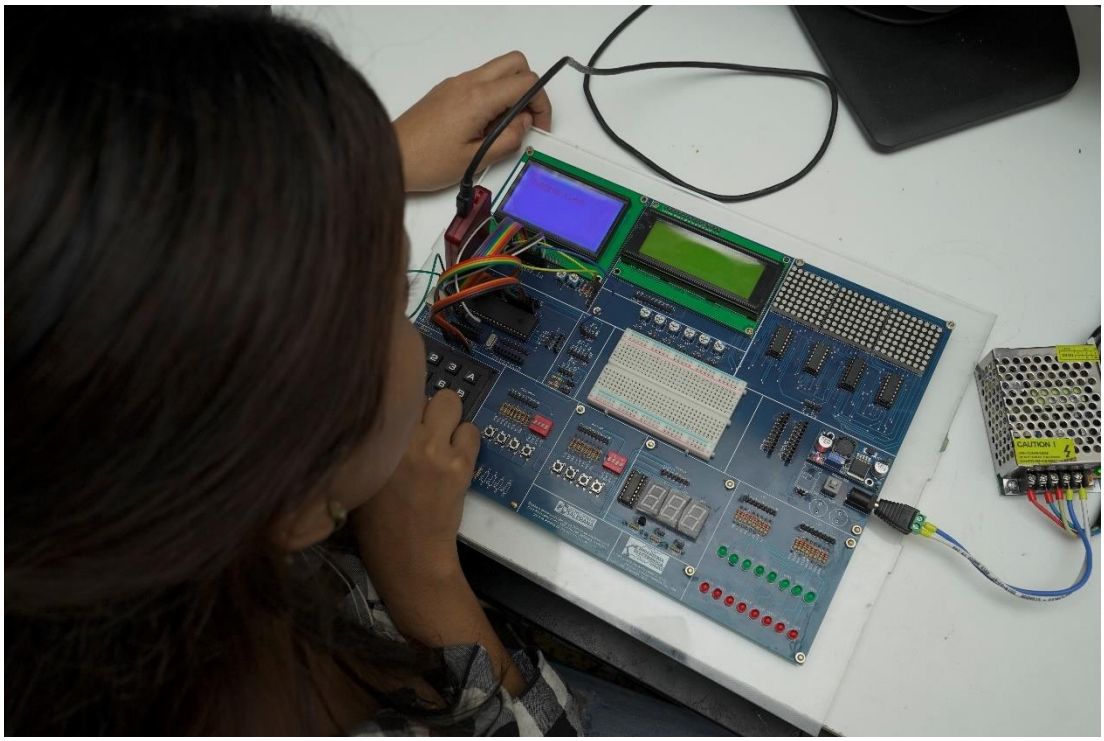


Figura 198. Conexiones físicas - Práctica 9

ANEXO 3.10: RESOLUCIÓN DE PRÁCTICA 10
AUTOMATIZACIÓN INDUSTRIAL

PRÁCTICA 10

NÚMERO DE ESTUDIANTES: 20

DOCENTE

ING. BREMNEN MARINO VELIZ NOBOA MSc.

TIEMPO ESTIMADO: 2 HORAS

TEMA: "ENVÍO Y RECIBO DE DATOS UTILIZANDO LCD 20X4, TECLADO MATRICIAL MEDIANTE EL PIC18F4550 Y LAS INTERRUPCIONES"

A. Objetivo General

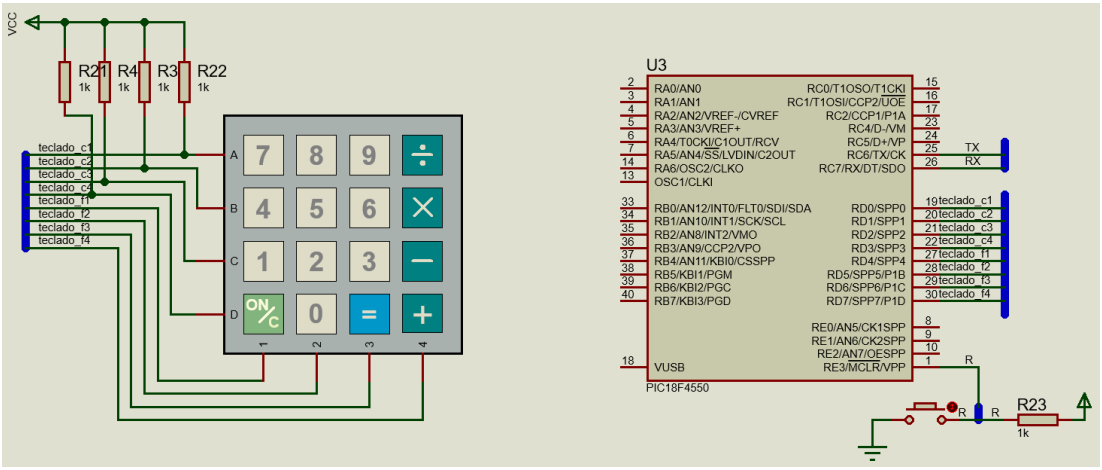
- Implementar un aplicativo para el envío y recepción de datos entre dos microcontroladores 18f4550 utilizando interrupciones.

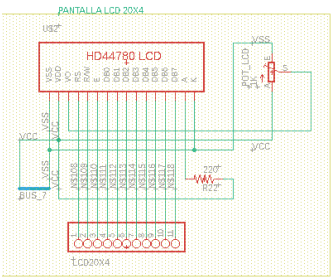
B. Objetivos Específicos

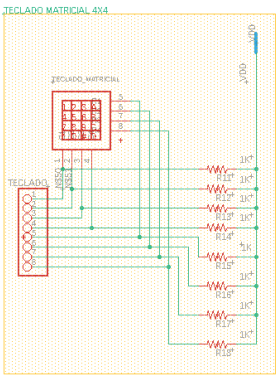
- Implementar un aplicativo para el envío y recepción de datos entre dos microcontroladores 18f4550 por protocolo UART.
- Programar un microcontrolador que envié valores por serial cada vez que se ejecute una interrupción por un tiempo determinado.
- Realizar un programa que reciba valores del puerto serial por una interrupción de comunicación UART.
- Realizar el cableado respectivo entre los dos módulos didácticos.

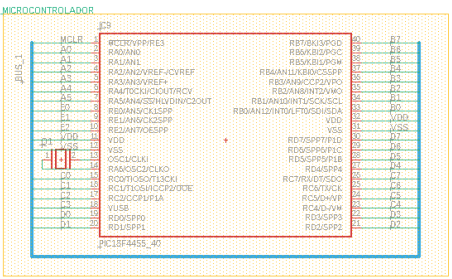
C. Marco Procedimental

1. Se procede a realizar el programa del emisor que envía los valores del teclado matricial 4x4 por una comunicación UART, el esquemático de las conexiones se muestra en la figura.


VCC
R21 1k
R4 1k
R3 1k
R22 1k
teclado_c1
teclado_c2
teclado_c3
teclado_c4
teclado_f1
teclado_f2
teclado_f3
teclado_f4
A
B
C
D
7
8
9
÷
4
5
6
×
1
2
3
−
ON/C
0
=
+
U3
RA0/AN0
RA1/AN1
RA2/AN2/VREF-/CVREF
RA3/AN3/VREF+
RA4/T0CKI/C1OUT/RCV
RA5/AN4/SS/LVDIN/C2OUT
RA6/OSC2/CLKO
OSC1/CLKI
RB0/AN12/INT0/FLT0/SDI/SDA
RB1/AN10/INT1/SCK/SCL
RB2/AN8/INT2/VMO
RB3/AN9/CCP2/VPO
RB4/AN11/KBI0/CSSPP
RB5/KBI1/PGM
RB6/KBI2/PGC
RB7/KBI3/PGD
RC0/T1OSO/T1CKI
RC1/T1OSI/CCP2/UOE
RC2/CCP1/P1A
RC4/D-/VM
RC5/D+/VP
RC6/TX/CK
RC7/RX/DT/SDO
RE0/AN5/CK1SPP
RE1/AN6/CK2SPP
RE2/AN7/OESPP
RE3/MCLR/VPP
VUSB
PIC18F4550
TX
RX
R
R23
1k


PANTALLA LCD 20X4
HD44780 LCD
LCD20X4


TECLADO MATRICIAL 4X4
TECLADO


MICROCONTROLADOR


BLOQUE UART
UART

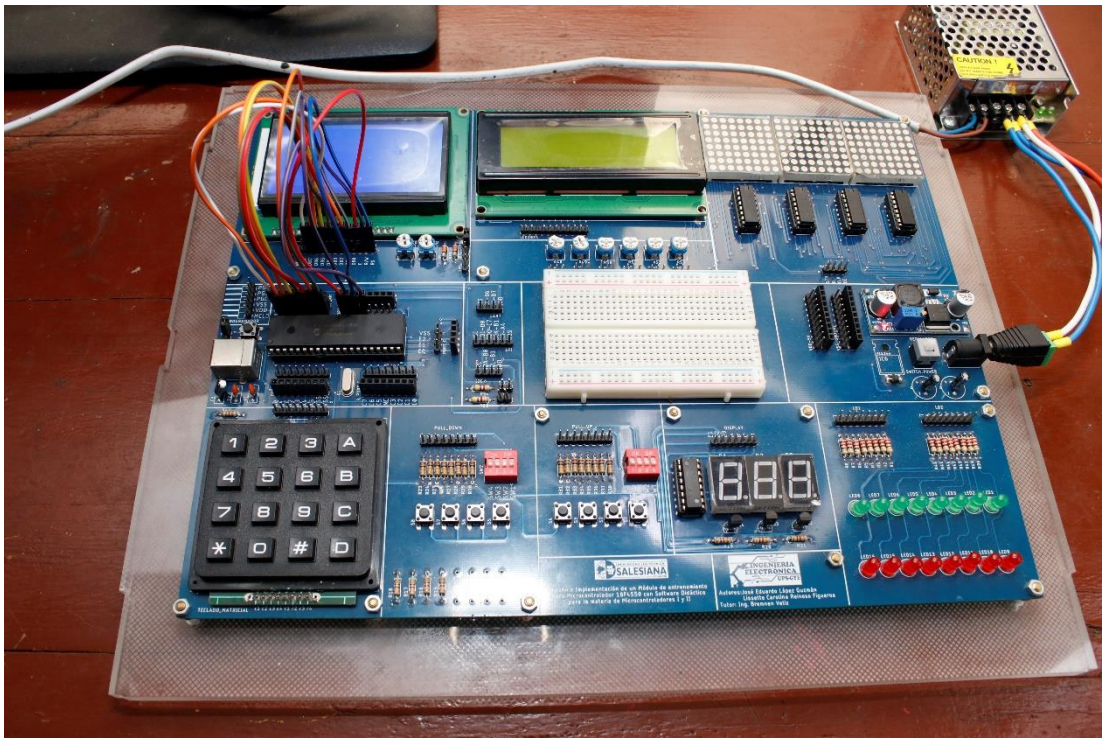


Figura 199. Esquema de conexión del emisor o maestro en PRÁCTICA 10

PIC18F4550 – C7	RX
PIC18F4550 – C6	TX
PIC18F4550 – D7	TECLADO f4
PIC18F4550 – D6	TECLADO f3
PIC18F4550 – D5	TECLADO f2
PIC18F4550 – D4	TECLADO f1
PIC18F4550 – D3	TECLADO c4
PIC18F4550 – D2	TECLADO c3
PIC18F4550 – D1	TECLADO c2
PIC18F4550 – D0	TECLADO c1

Tabla 13: Conexiones de PRÁCTICA 10

2. Ejecutar el programa PIC Compiler V 5 de la familia de microchip basado en el lenguaje C como se muestra en la figura.

Figura 200. PIC Compiler V 5 – PRÁCTICA 10

3. Al ejecutar el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_10_a** como se muestra en la figura-

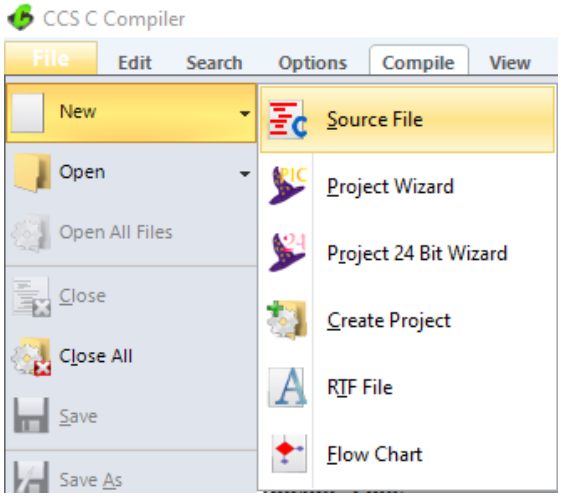


Figura 201. Creación de archivo de código en lenguaje C – PRÁCTICA 10

4. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

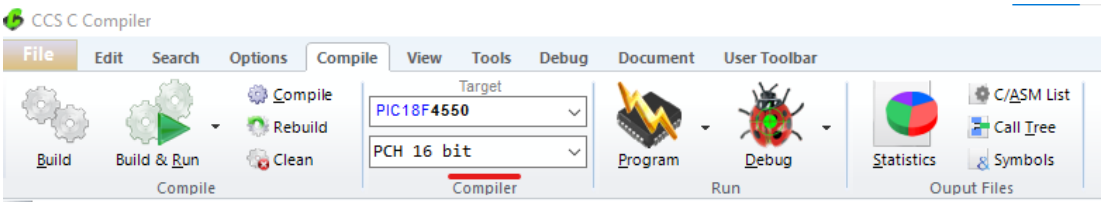


Figura 202. Configuración del microcontrolador en el compilador – PRÁCTICA 10

5. Se procede a llamar a la librería del microcontrolador, definir los fusibles, velocidad del cristal donde de define la velocidad y los puertos a utilizar como se muestra en la figura.

```
#include <18F4550.h>            //No Watch Dog Timer
#fuses INTRC_IO                 //clock fusible
#FUSES NOWDT                    //No Watch Dog Timer
#FUSES HS                       //High speed Osc (> 4mhz)
#FUSES PUT                      //Power Up Timer
#FUSES NOPROTECT                //Code not protected from reading
#FUSES NOBROWNOUT               //No brownout reset
#FUSES NOMCLR                   //Master Clear pin enabled
#FUSES NOLVP                    //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD
#use delay(clock = 20000000)
```

Figura 203. Declaración de librerías y registros -PRÁCTICA 10

6. Declarar el comando para utilizar el UART y declarar los pines y velocidad a utilizar y los puertos con su respectivo registro como se muestra en la figura.

```
#use rs232(baud=9600,xmit=pin_c6, rcv=pin_c7, bits=8, parity=N)//Declaracion UART
#include <kbd.c>      //Libreria del teclado
#BYTE portb = 0xF81   //Dirección de Puerto B en memoria.
#BYTE portc = 0xF82   //Dirección de Puerto C en memoria.
#BYTE portd = 0xF83   //Dirección de Puerto D en memoria.
int16 Ts_cont=0;           //Variable para interrupcion
char k;                    //Variable para dato del teclado
int t;                     //Variable numerica para envio por UART
```

Figura 204. Declaración de UART, registro del puerto y variables - PRÁCTICA 10

7. Para la interrupción se tiene que tomar el valor del tiempo en segundos a configurar en la interrupción con la ganancia del cristal de cuarzo sobre el valor del pre escalar con una ganancia de 4 como se muestra en la figura.

$$\frac{\text{Tiempo(s)} \cdot \text{F}_{\text{osc}}}{4 \cdot \text{Preescaler}} =$$

Figura 205. Fórmula para el timer1-PRÁCTICA 10

8. Declarar el sub-método que se ejecutara al momento de realizar la interrupción como el valor obtenido de la formula supera el valor máximo del Timer 65635(limite), valor obtenido es 625000, se toma en cuenta 10 ciclos de repetición para disminuir el valor a 62500, dejando el valor del Timer de la diferencia entre 65635 y 62500 enviando el valor de la tecla presionada por el serial cada segundo que se ejecute el Timer como se muestra en la figura.

```
#int_timer1  //TIMER1
void sampling_time(void){

   Ts_cont--;          //Se decrementa hasta llegar a cero
   set_timer1(3036);//Carga de nuevo el timer1
   if (Ts_cont<=0) // Si llega a cero, se cumplió el periodo de muestreo
   {
     printf("%d\r\n",t);  //Envio de dato numerico por UART
      Ts_cont=10;  // Inicializa el contador para el próximo periodo
   }
}
```

Figura 206. Sub método del timer1-PRÁCTICA 10

9. Posteriormente se declara el método principal del código donde se configura el inicio de la librería del teclado mediante el comando **kbd_init,**

la interrupción del Timer 1 con su pre escalar, el periodo del ciclo para los 10 segundos, el timer1 que es la diferencia entre el valor obtenido con el valor máximo y el lazo principal se obtiene el valor de la tecla con el comando **kbd_get**, se convierte el valor a decimal y posterior reinicio de variable k como se muestra en la figura.

```
void main()
   {
   Kbd_init();//Inicio de libreria del teclado
   lcd_init();// inicio de libreria LCD
   setup_timer_1 ( T1_INTERNAL | T1_DIV_BY_8 );// Configura el Timer 1
   Ts_cont=5;        //Carga Contrador para Calcular el Periodo de Muestreo
   set_timer1(3036); // Inicializa el Timer 1 para calcular 1 Segundos de Ts
   enable_interrupts(int_timer1);   // Habilitar las interrupciones del PIC TIMER1
   enable_interrupts(GLOBAL);   // Habilitar las interrupciones del PIC

         printf(lcd_putc,"\f EMISOR");//Se escribe en el lcd
         delay_ms(1000);
         printf(lcd_putc,"\f");//Se escribe en el lcd
   while(true)
   {
       do{ //espera hasta que se presione una tecla
   k=kbd_getc(); //Captura valor del teclado
   }
   while(k=='\n');
   t=k-48;
     lcd_gotoxy(1,1);//Se configura el puntero del lcd
         printf(lcd_putc,"Dato: %d",t);//Se escribe en el lcd
   }
}
```

Figura 207. Segmento principal emisor – Práctica 10

10. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura 180 Y 181.

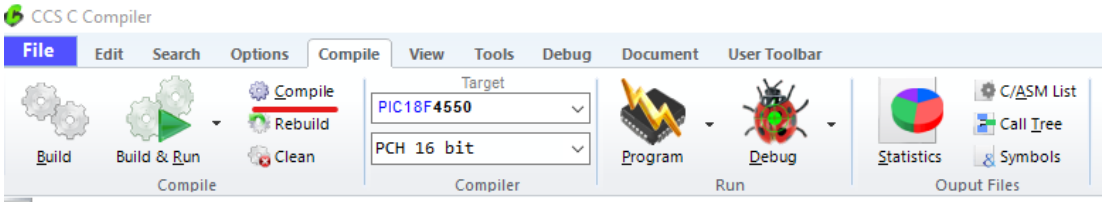


Figura 208. Botón para compilar el programa emisor – Práctica 10

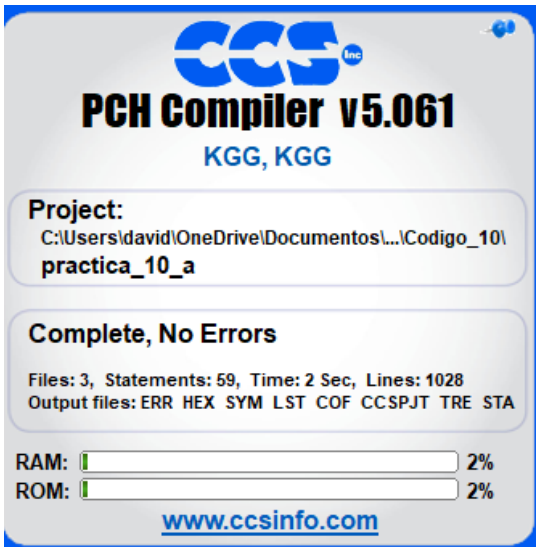


Figura 209. Compilación del programa emisor – Práctica 10

11. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 210. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 10

12. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

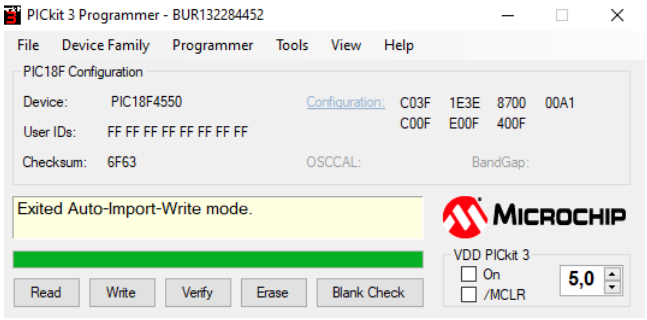


Figura 211. programación de PIC mediante el Pickit 3 – Práctica 10

13. Se procede a realizar el programa receptor que recibe el valor mediante una interrupción de comunicación del otro microcontrolador y se visualizara en una pantalla Lcd, el esquemático de las conexiones se muestra en la figura.

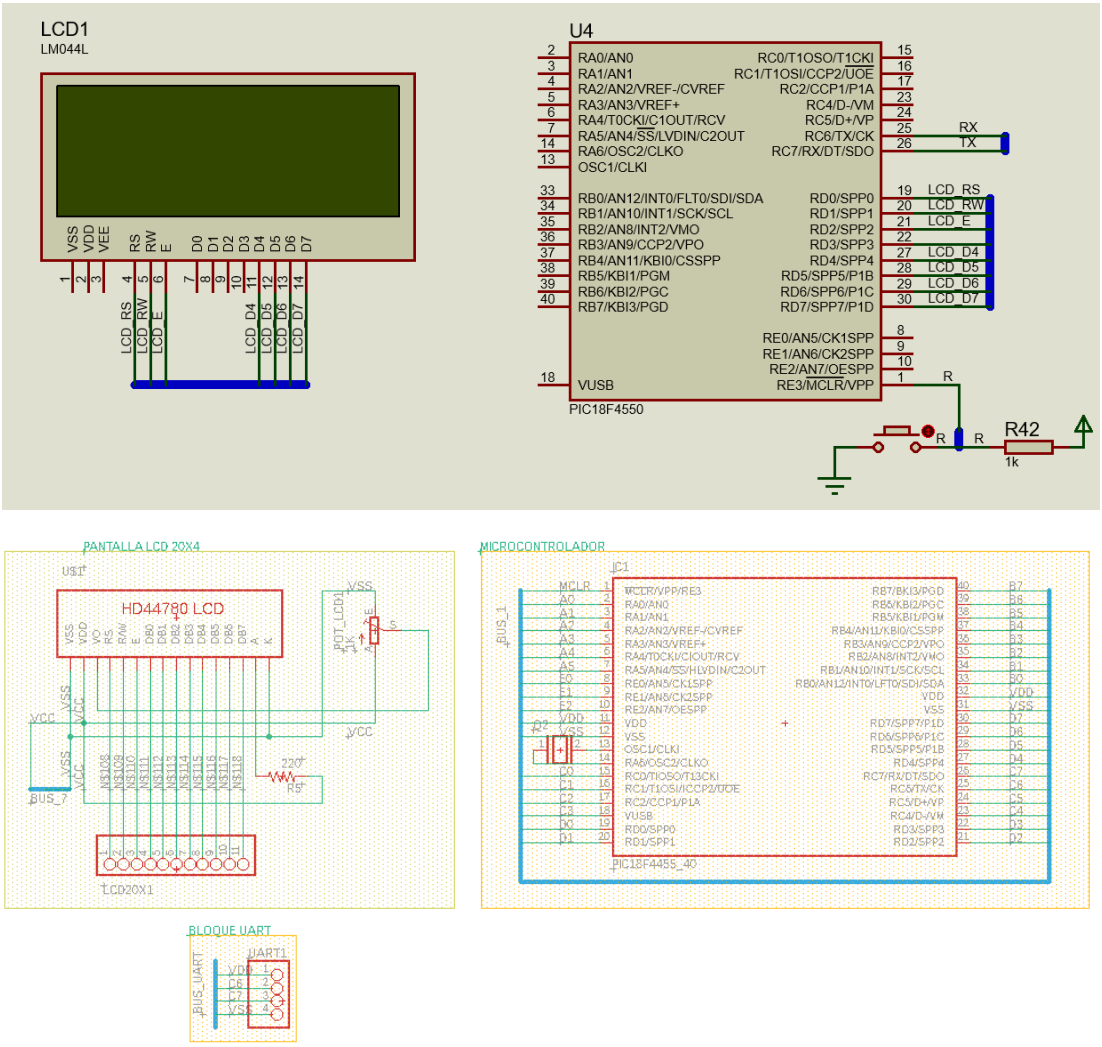


Figura 212. Esquema de conexión del receptor en PRÁCTICA 10

PIC18F4550 – B7	LCD D7
PIC18F4550 – B6	LCD D6
PIC18F4550 – B5	LCD D5
PIC18F4550 – B4	LCD D4
PIC18F4550 – B2	LCD E
PIC18F4550 – B1	LCD RW
PIC18F4550 – B0	LCD RS
PIC18F4550 – C6	RX
PIC18F4550 – C7	TX

Tabla 14: Conexiones de PRÁCTICA 10

14. Se procede a crear un archivo nuevo en el pic c Compiler para programar el código del receptor, en el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_10_b** como se muestra en la figura-

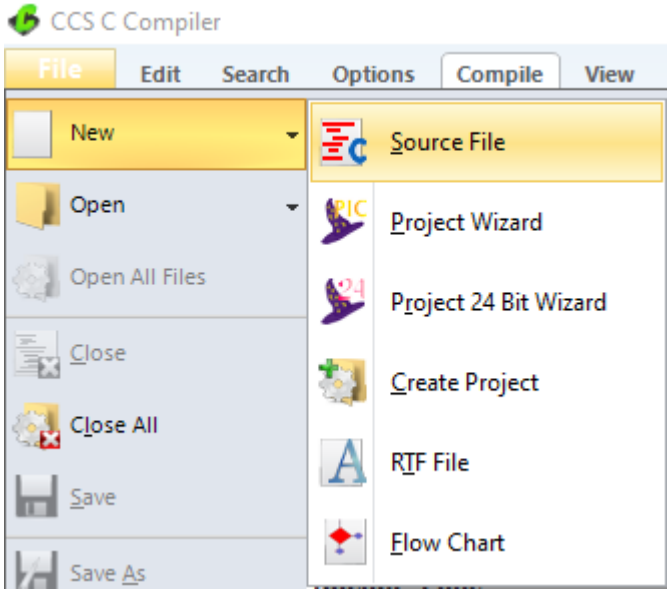


Figura 213. Creación de archivo de código receptor en lenguaje C – PRÁCTICA 10

15. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

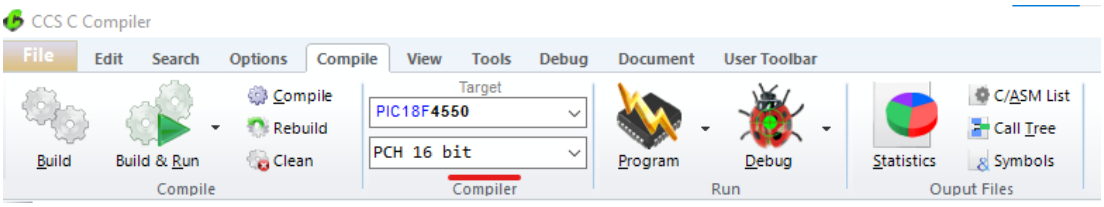


Figura 214. Configuración del microcontrolador en el compilador – PRÁCTICA 8

16. Se procede a llamar a la librería del microcontrolador, definir los fusibles, velocidad del cristal donde de define la velocidad, los puertos a utilizar como se muestra en la figura.

```
#include <18f4550.h>            //Declaración de libreria PIC18F4550
#use delay(clock=20Mhz)          //Declaracion del cristal de 20 Mhz
#FUSES NOWDT                     //No Watch Dog Timer
#FUSES HS                        //High speed Osc (> 4mhz)
#FUSES PUT                       //Power Up Timer
#FUSES NOPROTECT                 //Code not protected from reading
#FUSES NOBROWNOUT                //No brownout reset
#FUSES NOMCLR                    //Master Clear pin enabled
#FUSES NOLVP                     //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD
```

Figura 215. Declaración de librerías y registros -PRÁCTICA 10

17. Declarar el comando para utilizar el UART y declarar los pines y velocidad a utilizar, declara la librería del Lcd y definir los pines con su respectivo bit del puerto como se muestra en la figura.

```
#use rs232(baud=9600,xmit=pin_c6, rcv=pin_c7, bits=8, parity=N)//Declaración de UART
//Declaración de pines del lcd en el puerto D
#define LCD_RS_PIN      PIN_D0
#define LCD_RW_PIN      PIN_D1
#define LCD_ENABLE_PIN  PIN_D2
#define LCD_DATA4       PIN_D4
#define LCD_DATA5       PIN_D5
#define LCD_DATA6       PIN_D6
#define LCD_DATA7       PIN_D7
#include <lcd.c>  //Declaración de libreria
#BYTE portb = 0xF81   //Dirección de Puerto B en memoria.
#BYTE portc = 0xF82   //Dirección de Puerto C en memoria.
#BYTE portd = 0xF83   //Dirección de Puerto D en memoria.
```

Figura 216. Declaración de registro del puerto y librería del LCD -PRÁCTICA 10

18. Declarar el sub-método que se ejecuta cuando el serial reciba un dato y le guarda en una variable como se muestra en la figura

```
#int_RDA
RDA_isr()
{
        k=getc();// se obtiene el valor mediante el UART
        if(t>0)
        t=k-48;//Se convierte el valor char a numerico

}
```

Figura 217. Declaración de método de interrupción serial -PRÁCTICA 10

19. Posteriormente se declara el método principal del código donde se configura las variables para el dato del UART y una variable numérica que será visualizada en el Lcd, posterior se inicia la pantalla mediante el comando **lcd_init,** posteriormente se obtiene el valor de la interrupción del UART y se visualiza en el Lcd. y espera de 100 milisegundo como se muestra en la figura.

```
void main(){

   lcd_init();// inicio de libreria LCD
   enable_interrupts(INT_RDA);//INTERRUPCION DE COMUNICACION
   enable_interrupts(GLOBAL);//INTERRUPCION DE COMUNICACION
   while(TRUE)
      {

         lcd_gotoxy(1,1);//Se configura el puntero del lcd
         printf(lcd_putc,"Dato: %d",t);//Se escribe en el lcd
         delay_ms(100); //Espera de 100 milisegundos
      }
}
```

Figura 218. Segmento principal receptor – Práctica 10

20. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura.

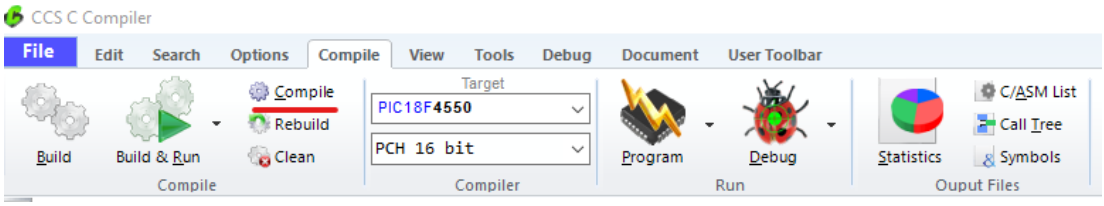


Figura 219. Botón para compilar el programa receptor – Práctica 10

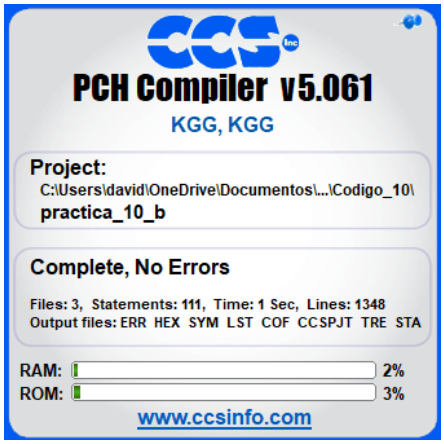


Figura 220. Compilación del programa receptor – Práctica 9

21. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 221. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 10

22. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

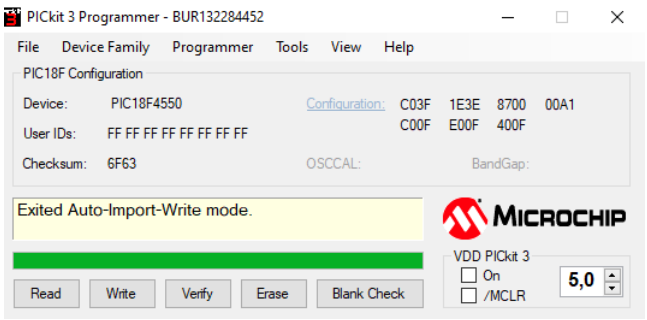


Figura 222. Programación de PIC mediante el Pickit 3 – Práctica 10

D. Registro de Resultados

En la décima práctica se tiene como resultado la comunicación entre dos controladores Pic 18f4550 utilizando dos interrupciones una para el emisor una interrupción por tiempo y otra para el receptor que es para la comunicación serial donde el primero envía los valores de un teclado matricial y el receptor muestra los valores en un LCD.

E. Conexiones físicas de la PRÁCTICA

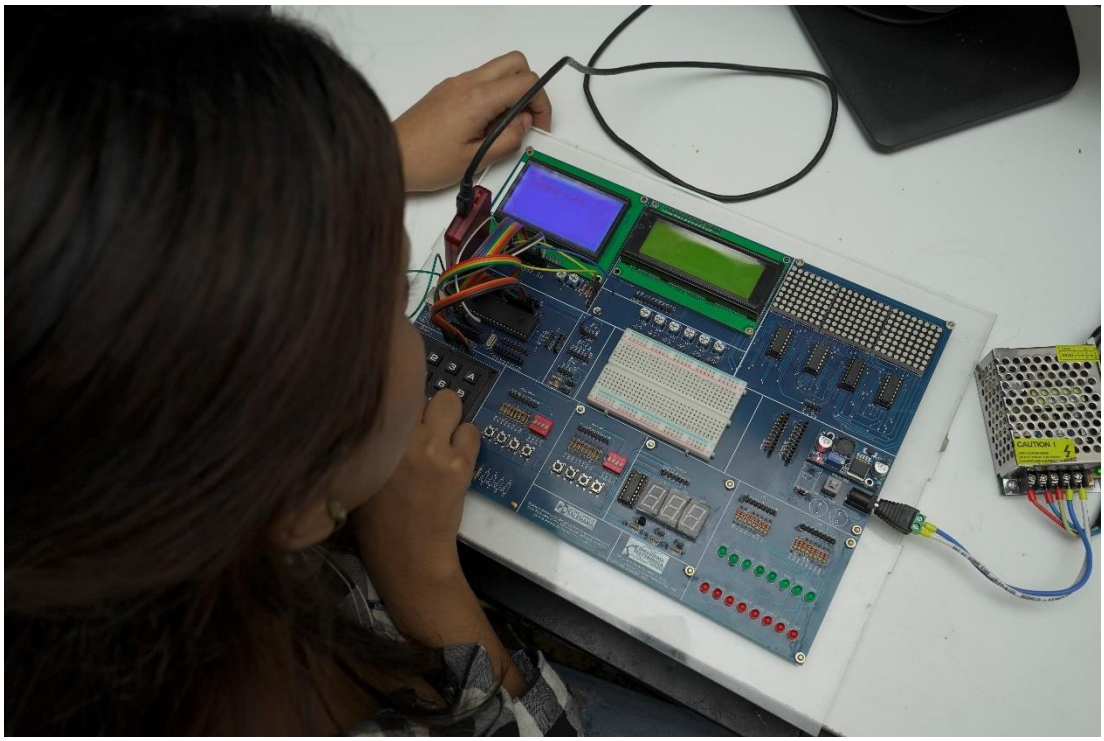


Figura 223. Conexiones físicas - Práctica 10

ANEXO 3.11: RESOLUCIÓN DE PRÁCTICA 11
AUTOMATIZACIÓN INDUSTRIAL

PRÁCTICA 11

NÚMERO DE ESTUDIANTES: 20

DOCENTE

ING. BREMNEN MARINO VELIZ NOBOA MSc.

TIEMPO ESTIMADO: 2 HORAS

TEMA: “GRAFICAS MEDIANTE GLCD UTILIZANDO EL PIC18F4550”

A. Objetivo General

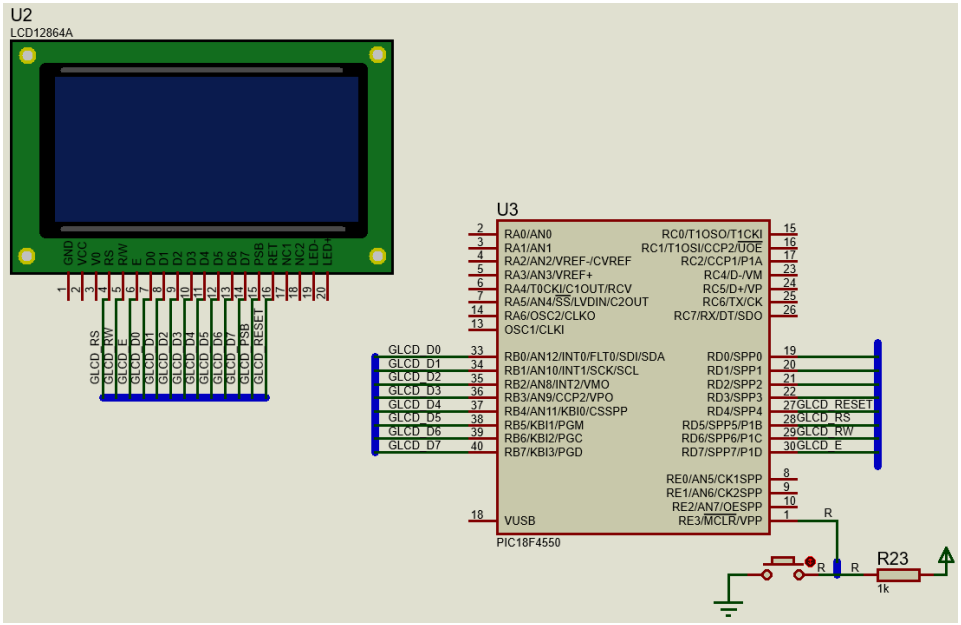
- Implementar una guía para el uso de un GLDC de 128x64 para realizar grafica con un microcontrolador 18f4550.

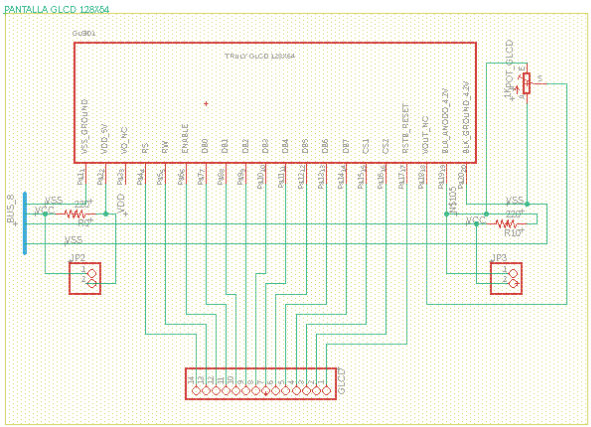
B. Objetivos Específicos

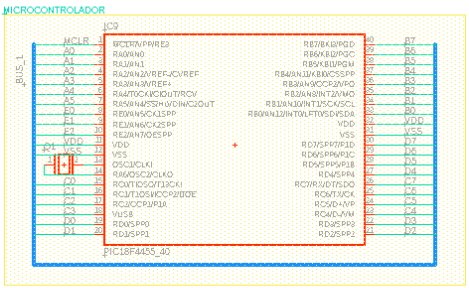
- Implementar una guía para el uso de un GLDC de 128x64 para realizar grafica con un microcontrolador 18f4550.
- Implementar la visualización de graficas o figuras geométricas básicas en un GLCD.
- Realizar el cableado respectivo entre los dos módulos didácticos

C. Marco Procedimental

1. Se procede a realizar el programa del que dibujara mediante comandos en el GLCD se tiene como esquemático de las conexiones la figura.







Elaborado por:	**Revisado por:**	**Aprobado por:** Ing. Orlando Barcia Ayala
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

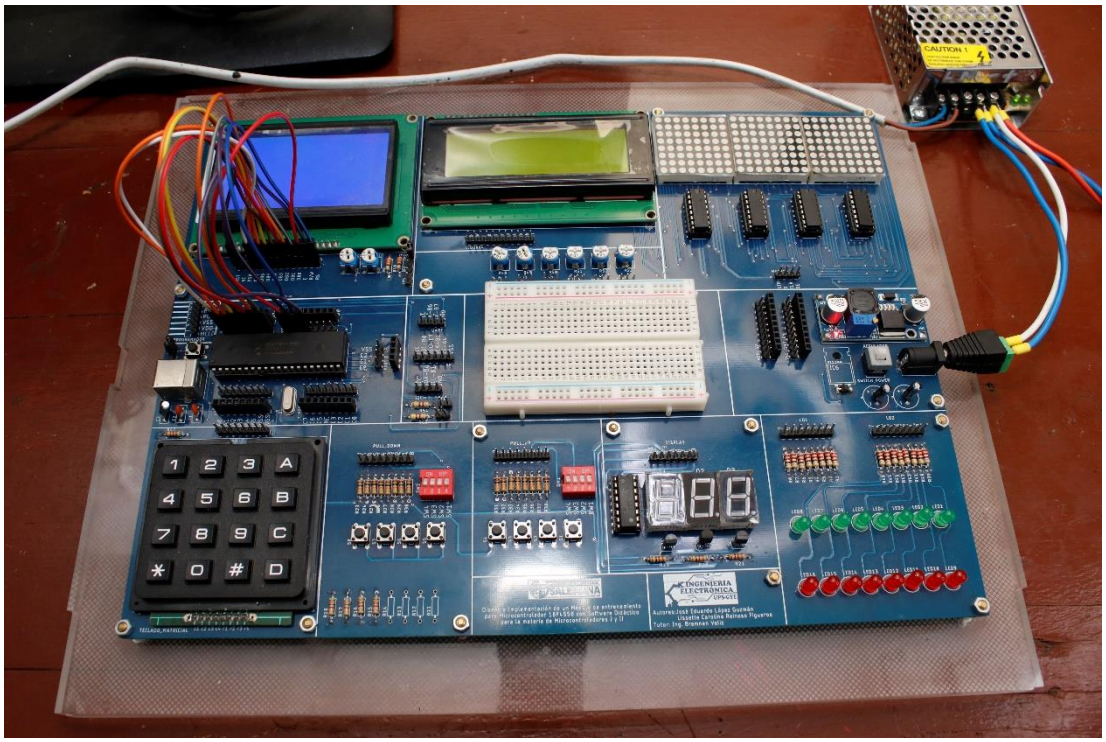


Figura 224. Esquema de conexión PRÁCTICA 11

PIC18F4550 – B7	GLCD D7
PIC18F4550 – B6	GLCD D6
PIC18F4550 – B5	GLCD D5
PIC18F4550 – B4	GLCD D4
PIC18F4550 – B3	GLCD D3
PIC18F4550 – B2	GLCD D2
PIC18F4550 – B1	GLCD D1
PIC18F4550 – B0	GLCD D0
PIC18F4550 – D4	GLCD_RESET
PIC18F4550 – D5	GLCD_RS
PIC18F4550 – D6	GLCD_RW
PIC18F4550 – D7	GLCD_E

Tabla 15: Conexiones de PRÁCTICA 11

2. Ejecutar el programa PIC Compiler V 5 de la familia de microchip basado en el lenguaje C como se muestra en la figura.

Figura 225. PIC Compiler V 5 – PRÁCTICA 11

3. Al ejecutar el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_11** como se muestra en la figura-

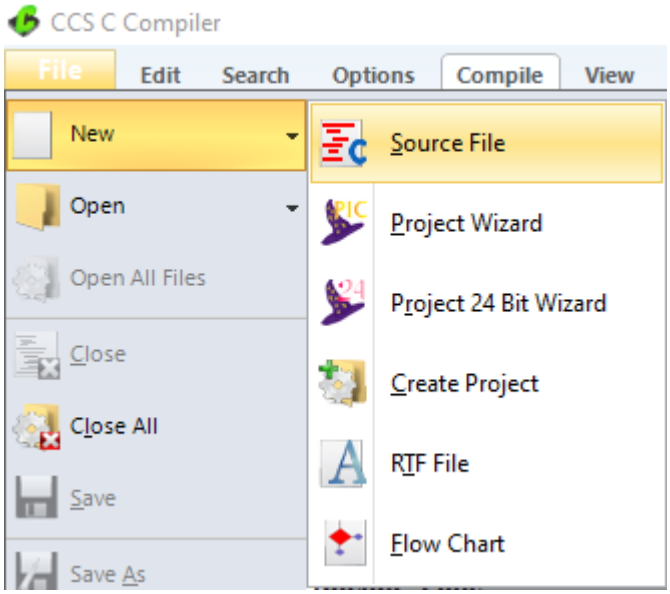


Figura 226. Creación de archivo de código en lenguaje C – PRÁCTICA 11

4. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

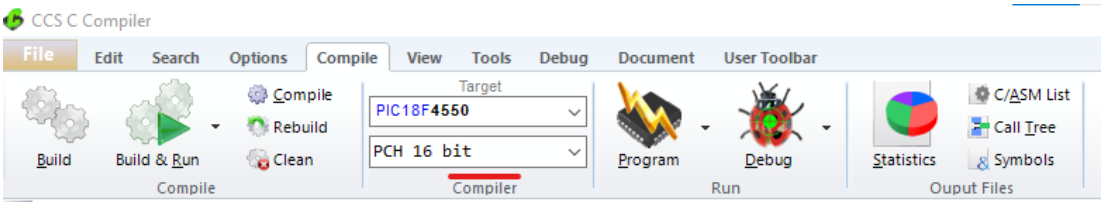


Figura 227. Configuración del microcontrolador en el compilador – PRÁCTICA 11

5. Se procede a llamar a la librería del microcontrolador, definir los fusibles, velocidad del cristal donde de define la velocidad y los puertos a utilizar como se muestra en la figura.

```
#include <18F4550.h>
#fuses INTRC_IO                  //clock fusible
#FUSES NOWDT                     //No Watch Dog Timer
#FUSES HS                        //High speed Osc (> 4mhz)
#FUSES PUT                       //Power Up Timer
#FUSES NOPROTECT                 //Code not protected from reading
#FUSES NOBROWNOUT                //No brownout reset
#FUSES NOMCLR                    //Master Clear pin enabled
#FUSES NOLVP                     //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#use delay(clock=20Mhz)    //Declaración de la velocidad del cristal de quarzo
```

Figura 228. Declaración de librería del pic y registros -PRÁCTICA 11

6. Declarar las librerías a utilizar para el Glcd , comandos del Glcd, y variables a utilizar como se muestra en la figura.

```
#include <stdlib.h>        //libreria para transformar datos enteros en tipo char (texto)
#include <math.h>          //libreria para calculo
#include "sT7920.h"        //libreria del GLCD
#include "GRAPHICS.C"      //libreria de los comandos del GLCD
//MENSAJES A PUBLICAR EN EL GLCD
char men1[ ]="PRACTICA 11";
char men2[ ]="IMPLEMENTACION DE    GRAFICAS MEDIANTE    GLCD UTILIZANDO EL   PIC18F4550";
```

Figura 229. Declaración de librerías y variables -PRÁCTICA 11

7. Posteriormente se declara el método principal del código donde se configura el inicio de la librería del GLCD mediante el comando **glcd_init_graph,** se limpia valores en la pantalla con el comando **glcd_fillscreen** y se actualiza la pantalla con el comando **glcd_update**, en el lazo de repetición se procede a mostrar los mensajes en el Glcd, dibujar con el comando **lcd_line** , dibujar figuras circulares y rectangulares como se muestra en la figura.

```
void main(){
glcd_init_graph();//INICIO DE GLCD
glcd_fillscreen(OFF);//Limpia el GLCD
glcd_update();//Actualiza la pantalla para visualizar comandos
   while(1)
   {
      //Se publica el mensaje 1 y mensaje 2 mediante el comando text57
      glcd_text57(1,1,men1,1,1); //Comando para escriturar de un string
      glcd_text57(1,10,men2,1,1);//Comando para escriturar de un string
      glcd_update();//Actualiza la pantalla para visualizar comandos
      delay_ms(2000);//espera de 2 segundos
      glcd_fillscreen(OFF);//borra la pantalla
      glcd_update();//Actualiza la pantalla para visualizar comandos
      for(int x=0;x<=63;x++){//dibuja una linea
         lcd_line(1,1,x,x,1); // comando para dibujar una linea
         glcd_update();//Actualiza la pantalla para visualizar comandos
         delay_ms(1);
      }
      for(int y=0;y<=63;y++){
         lcd_line(63,63,63+y,63-y,1);
         glcd_update();//Actualiza la pantalla para visualizar comandos
         delay_ms(1);
      }
      delay_ms(2000);//espera de 2 segundos
      glcd_fillscreen(OFF);//borra la pantalla
      glcd_update();//Actualiza la pantalla para visualizar comandos
      for(int z=0;z<=30;z++){
         glcd_circle(30,30,z,1,1);//comando para dibujar un circulo
         glcd_update();//Actualiza la pantalla para visualizar comandos
         delay_ms(1);
      }
      delay_ms(2000);//espera de 2 segundos
      glcd_fillscreen(OFF);//borra la pantalla
      glcd_update();//Actualiza la pantalla para visualizar comandos
      for(int z=0;z<=30;z++){
         glcd_rect(1,1,1,z,1,1);//comando para dibujar un rectangulo
         glcd_update();//Actualiza la pantalla para visualizar comandos
         delay_ms(1);
      }
      delay_ms(2000);//espera de 2 segundos
   }
}
```

Figura 230. Segmento principal – Práctica 11

8. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura 180 Y 181.

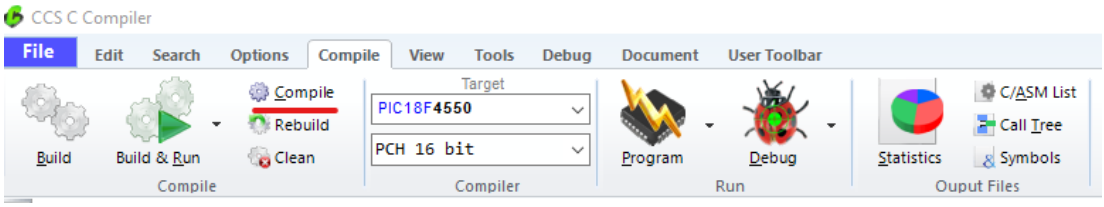


Figura 231. Botón para compilar el programa emisor – Práctica 10

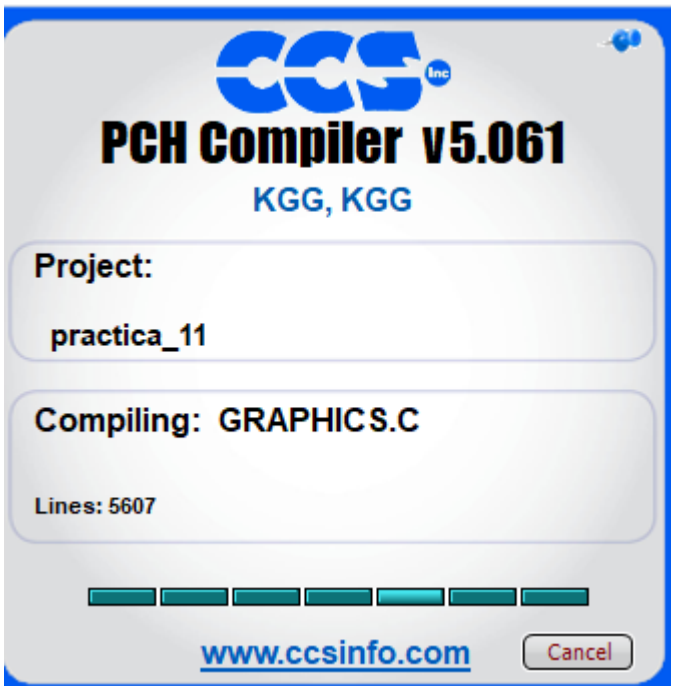


Figura 232. Compilación del programa emisor – Práctica 10

9. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 233. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 10

10. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

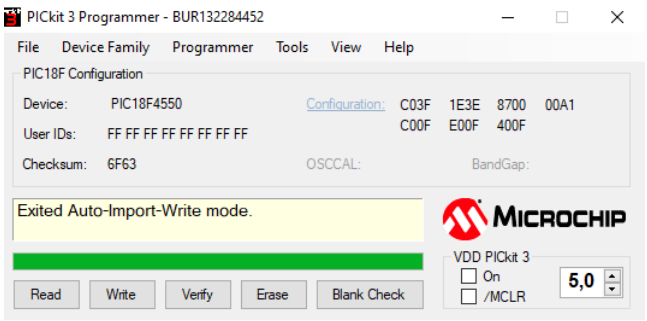


Figura 234. programación de PIC mediante el Pickit 3 – Práctica 10

D. Registro de Resultados

En la onceava práctica se tiene como resultado él envió de comandos desde el microcontrolador al GLCD para lograr el entendimiento del uso de las librerías ST7920 del GLCD y un PIC 18f4550

E. Resultados de PRÁCTICA

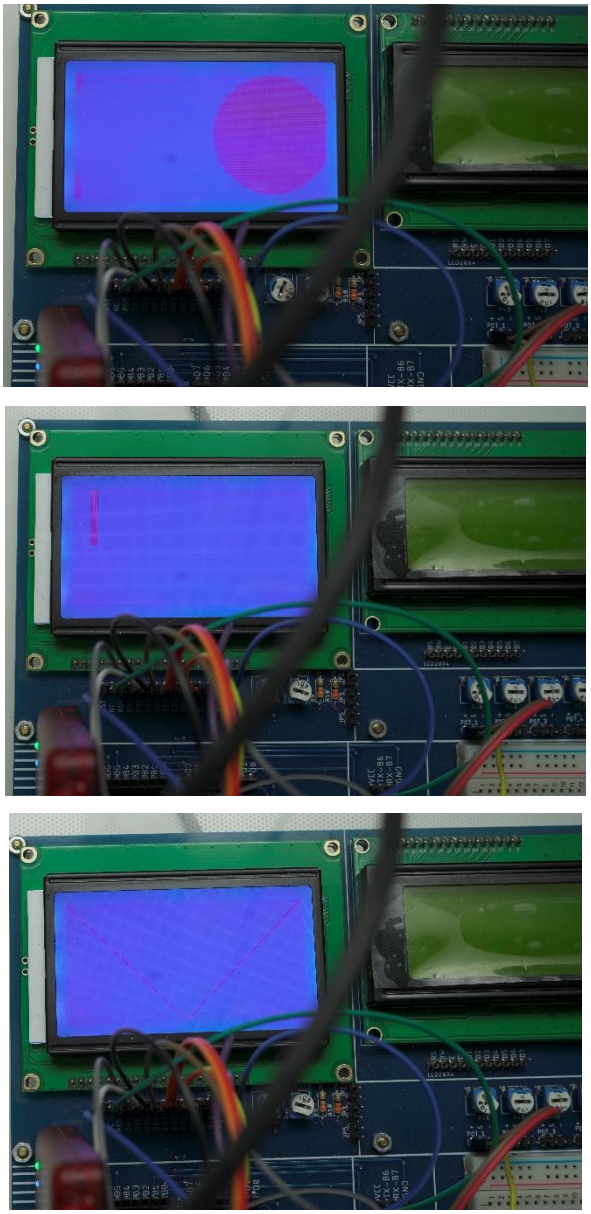


Figura 235. Resultados – Práctica 11

ANEXO 3.12: RESOLUCIÓN DE PRÁCTICA 12
AUTOMATIZACIÓN INDUSTRIAL

PRÁCTICA 12

NÚMERO DE ESTUDIANTES: 20

DOCENTE

ING. BREMNEN MARINO VELIZ NOBOA MSc.

TIEMPO ESTIMADO: 2 HORAS

TEMA: "GRAFICAS MEDIANTE GLCD Y POTENCIÓMETRO UTILIZANDO EL PIC18F4550."

Elaborado por:	**Revisado por:**	**Aprobado por:** Ing. Orlando Barcia Ayala
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

A. Objetivo General

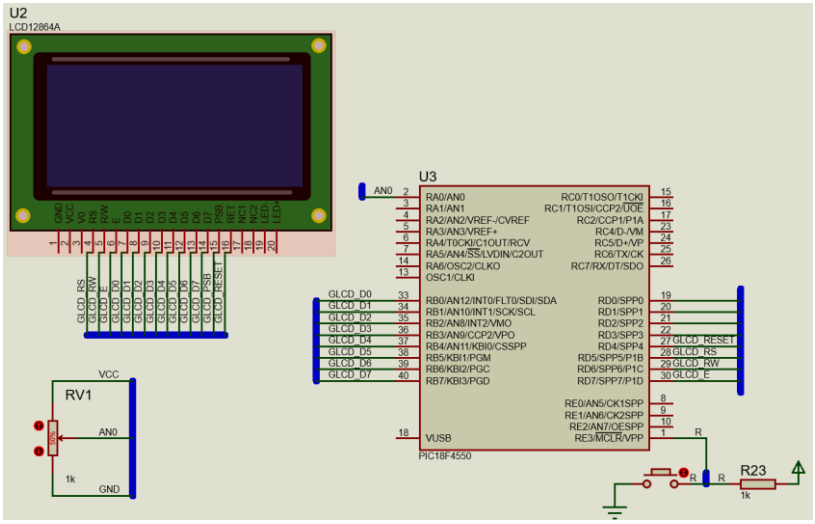
- Implementar una gráfica mediante GLCD y Potenciómetro utilizando el PIC18F4550.

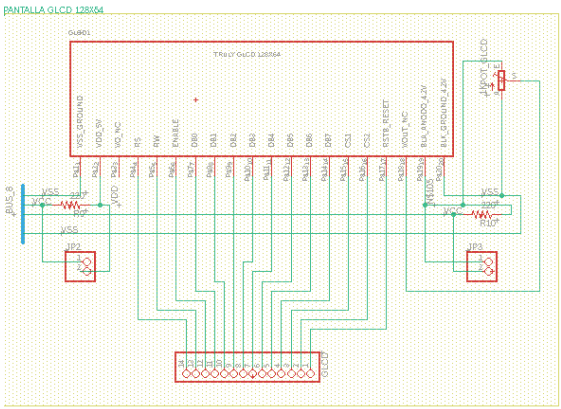
B. Objetivos Específicos

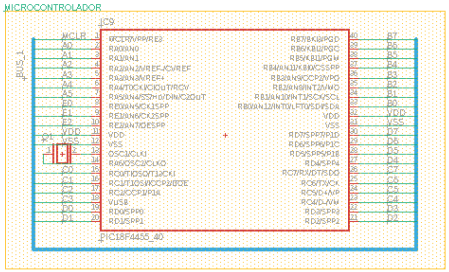
- Implementar una guía para el uso de un GLDC de 128x64 para realizar grafica con un microcontrolador 18f4550.
- Implementar la visualización de una figura geométrica circular controlando el radio mediante un valor obtenido por conversión análoga a digital
- Convertir un valor numérico a string utilizando el comando itoa.

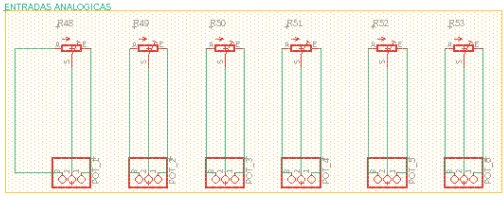
C. Marco Procedimental

1. Se procede a realizar el programa del Glcd donde se tendrá una señal analógica conectada al microcontrolador que sirve de señal para indicar el radio de la figura circular a mostrarse en el GLCD, el esquemático de las conexiones se muestra en la figura.


U2
LCD12864A
U3
PIC18F4550
RV1
R23
1k
GND
VCC


PANTALLA GLCD 128X64


MICROCONTROLADOR


ENTRADAS ANALOGICAS

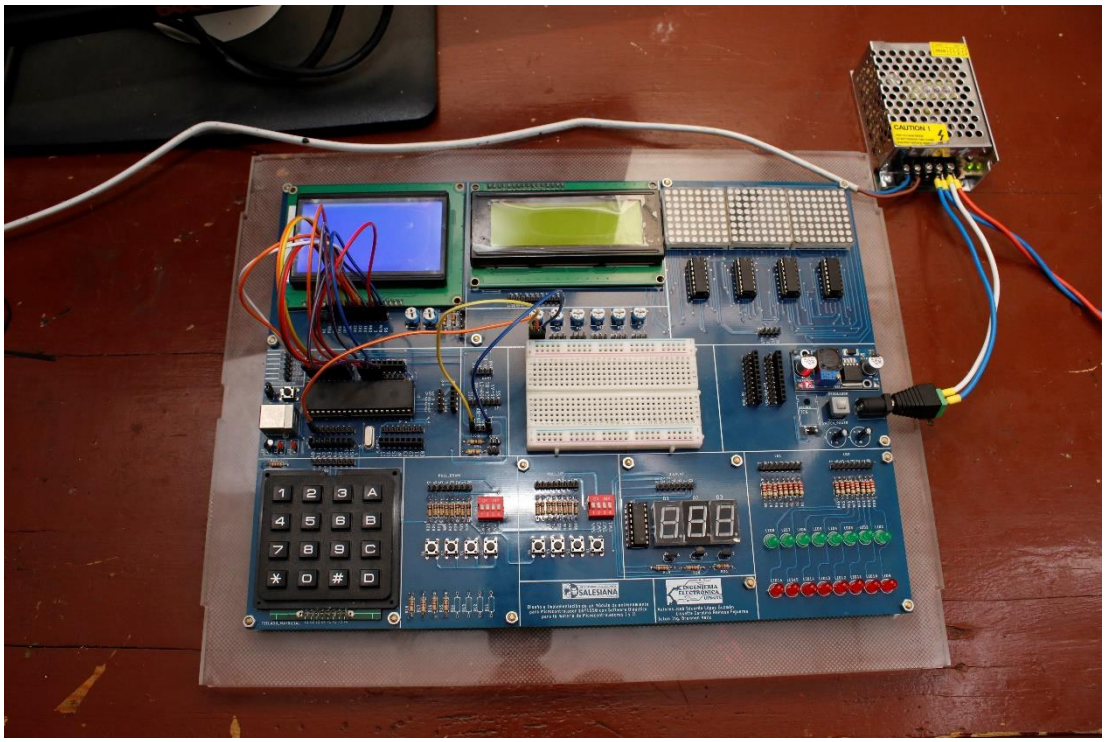


Figura 236. Esquema de conexión en PRÁCTICA 12

PIC18F4550 – B7	GLCD D7
PIC18F4550 – B6	GLCD D6
PIC18F4550 – B5	GLCD D5
PIC18F4550 – B4	GLCD D4
PIC18F4550 – B3	GLCD D3
PIC18F4550 – B2	GLCD D2
PIC18F4550 – B1	GLCD D1
PIC18F4550 – B0	GLCD D0
PIC18F4550 – D4	GLCD_RESET
PIC18F4550 – D5	GLCD_RS
PIC18F4550 – D6	GLCD_RW
PIC18F4550 – D7	GLCD_E
PIC18F4550 –A0	POT S

PIC18F4550 –VCC	POT VCC
PIC18F4550 –GND	POT GND

Tabla 16: Conexiones de PRÁCTICA 12

2. Ejecutar el programa PIC Compiler V 5 de la familia de microchip basado en el lenguaje C como se muestra en la figura.

Figura 237. PIC Compiler V 5 – PRÁCTICA 12

3. Al ejecutar el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_12** como se muestra en la figura-

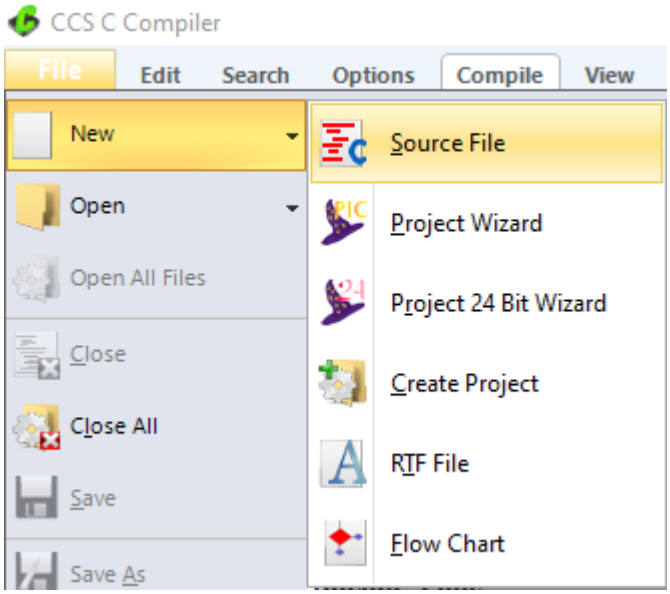


Figura 238. Creación de archivo de código en lenguaje C – PRÁCTICA 12

4. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

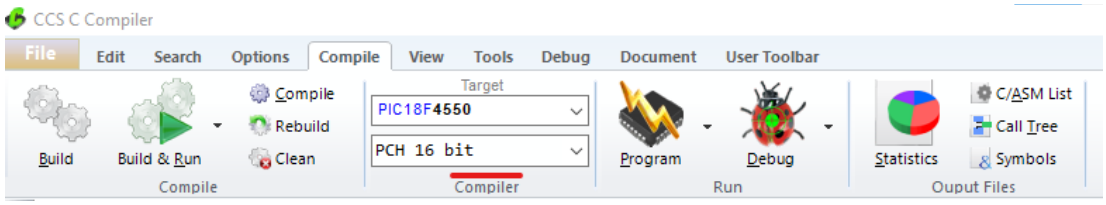


Figura 239. Configuración del microcontrolador en el compilador – PRÁCTICA 12

5. Se procede a llamar a la librería del microcontrolador, definir los fusibles, velocidad del cristal donde de define la velocidad, el tamaño de los bits del conversor analógico como se muestra en la figura.

```
#include <18F4550.h>
#DEVICE ADC=8                    //Definir el bit del conversor adc
#fuses INTRC_IO                  //clock fusible
#FUSES NOWDT                     //No Watch Dog Timer
#FUSES HS                        //High speed Osc (> 4mhz)
#FUSES PUT                       //Power Up Timer
#FUSES NOPROTECT                 //Code not protected from reading
#FUSES NOBROWNOUT                //No brownout reset
#FUSES NOMCLR                    //Master Clear pin enabled
#FUSES NOLVP                     //No low voltage prgming, B3(PIC16)
#use delay(clock=20Mhz)    //Declaración de la velocidad del cristal
```

Figura 240. Declaración de librerías y registros -PRÁCTICA 12

6. Declarar las librerías del conversor de decimal a texto, la librería del Glcd, la librería de comandos del Glcd y las variables a utilizar como se muestra en la figura.

```
#include <stdlib.h>          //libreria para transformar
#include <math.h>            //libreria para calculo
#include "sT7920.h"          //libreria del GLCD
#include "GRAPHICS.C"        //libreria de los comandos
//MENSAJES A PUBLICAR EN EL GLCD
char men1[ ]="Radio:";
char V[3];
unsigned int valor;
```

Figura 241. Declaración de librerías y variables -PRÁCTICA 12

7. Posteriormente se declara el método principal del código donde se configura el inicio de la librería del GLCD mediante el comando **glcd_init_graph,** se tiene que realizar una limpieza con el comando **Glcd_update** , proceder a configurar el reloj del conversor análogo a digital con el comando **a setup_adc(ADC_CLOCK_INTERNAL),** y a configurar el bit donde realizar la conversión con el comando **setup_adc_ports(AN0)**, en el lazo de repetición se procede a adquirir el

valor del bit AN0 y realizar su escalamiento y se convierte el valor de numérico a texto y se dibuja en el Glcd como se muestra en la figura.

```
void main(){
glcd_init_graph();//INICIO DE GLCD
glcd_fillscreen(OFF);//Limpia el GLCD
glcd_update();//Actualiza la pantalla para visualizar comandos
setup_adc(ADC_CLOCK_INTERNAL);//configuración del reloj del conversor
setup_adc_ports(AN0);//configuración del AN0
   while(1)
   {
         set_adc_channel(0);//seleccionamos el conversor en el AN0
         delay_us(10);//ESPERA
         valor=read_adc();//realizamos adquisición
         valor=valor*0.1;//escalamiento
         itoa(valor,10,V);//conversión de numero a texto
         glcd_circle(30,30,valor,1,1);//comando para dibujar un circulo
         glcd_text57(97,10,men1,1,1);//comando para escribir mensaje 1
         glcd_text57(100,30,V,1,1);//comando para escribir el valor
         glcd_update();//Actualiza la pantalla para visualizar comandos
         delay_ms(500);
         glcd_fillscreen(OFF);//Limpia el GLCD
      }
   }
```

Figura 242. Segmento principal – Práctica 12

8. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura 180 Y 181.

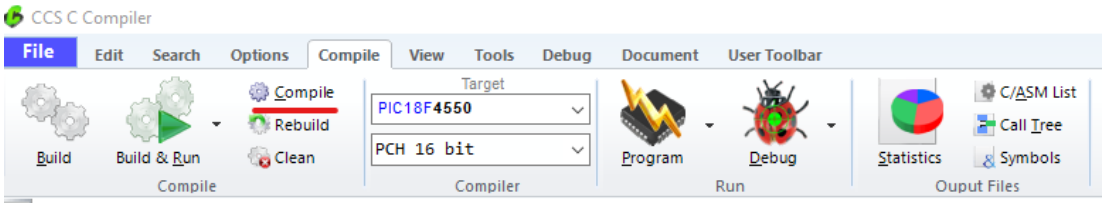


Figura 243. Botón para compilar el programa – Práctica 12

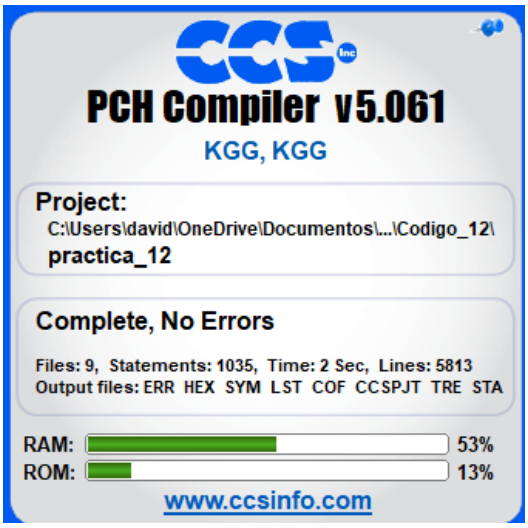


Figura 244. Compilación del programa – Práctica 12

9. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 245. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 12

10. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

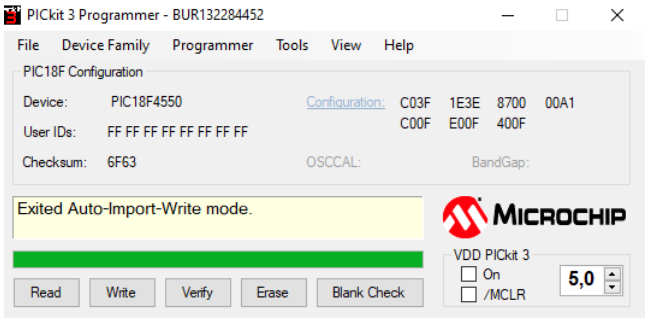


Figura 246. programación de PIC mediante el Pickit 3 – Práctica 12

D. Registro de Resultados

En la décimo segunda práctica se tiene como resultado la comunicación entre un microcontrolador Pic 18f4550 con un periférico GLCD donde el micro realizara una conversión análoga a digital y controla una figura en la pantalla grafica mostrando el valor de la conversión y el radio a dibujarse de un círculo.

E. Conexiones físicas de la PRÁCTICA

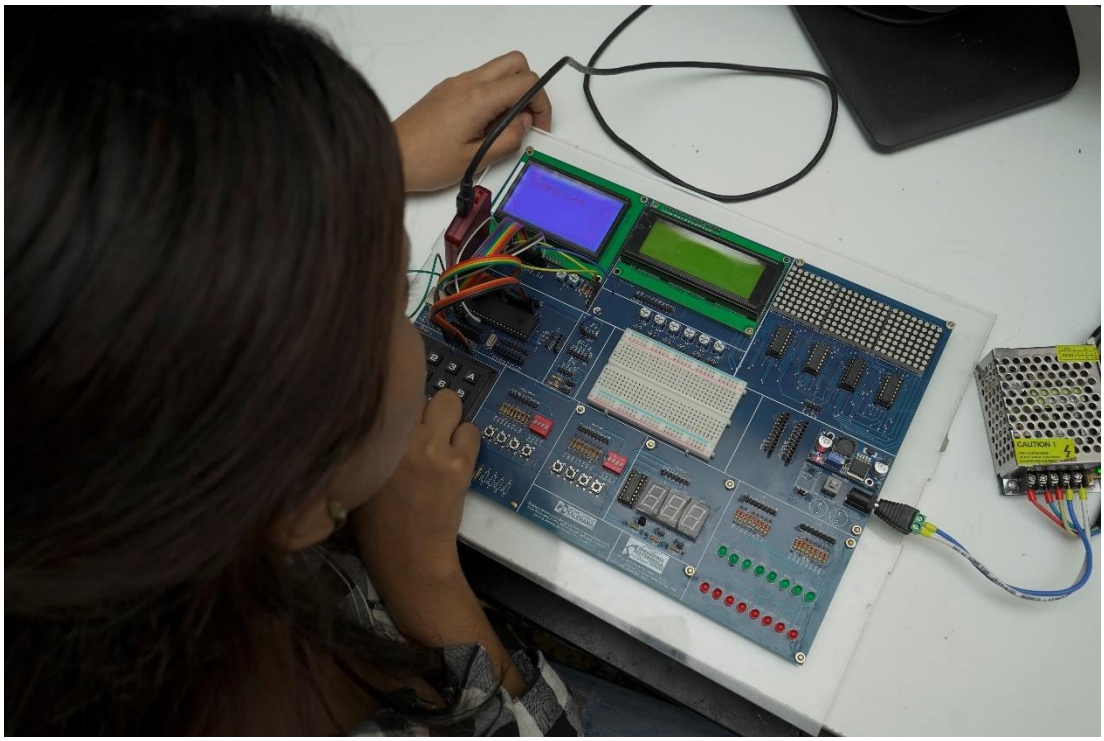


Figura 247. Conexiones físicas - Práctica 12

ANEXO 3.13: RESOLUCIÓN DE PRÁCTICA 13
AUTOMATIZACIÓN INDUSTRIAL

PRÁCTICA 13

NÚMERO DE ESTUDIANTES: 20

DOCENTE

ING. BREMNEN MARINO VELIZ NOBOA MSc.

TIEMPO ESTIMADO: 2 HORAS

TEMA: "SISTEMA DE CAJA FUERTE UTILIZANDO GLCD, TECLADO MATRICIAL, BUZZER Y PIC18F4550."

Elaborado por:	**Revisado por:**	**Aprobado por:** Ing. Orlando Barcia Ayala
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

A. Objetivo General

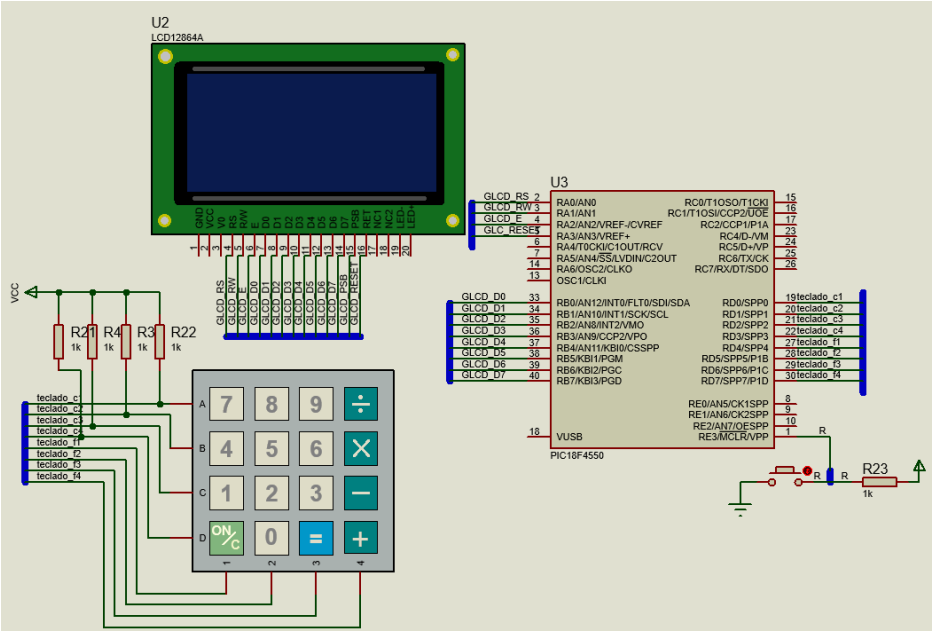
- Implementar un aplicativo para el sistema de caja fuerte utilizando GLCD, Teclado Matricial, Buzzer y PIC18f4550.

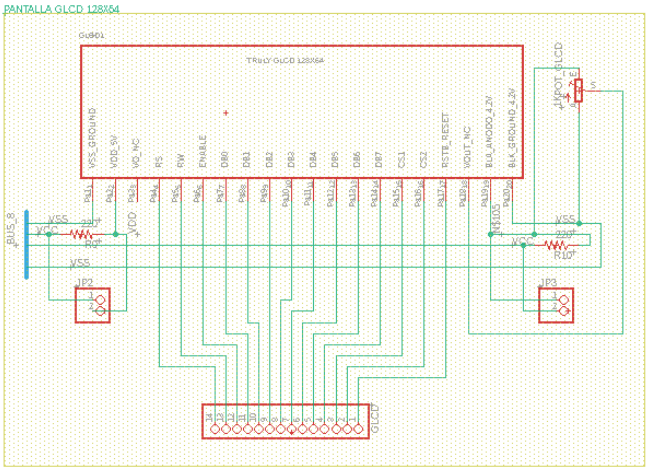
B. Objetivos Específicos

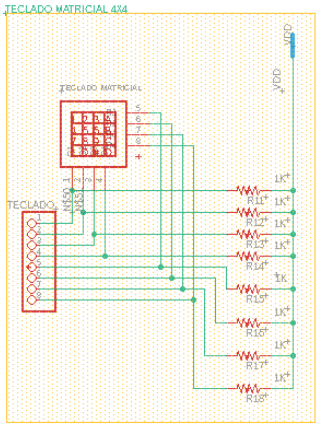
- Implementar: un aplicativo en el módulo didáctico que simule un sistema de acceso a una caja fuerte.
- Diseñar un programa que mediante el uso de librerías logre enviar los datos al GLCD y al teclado separando el nivel de cómputo.

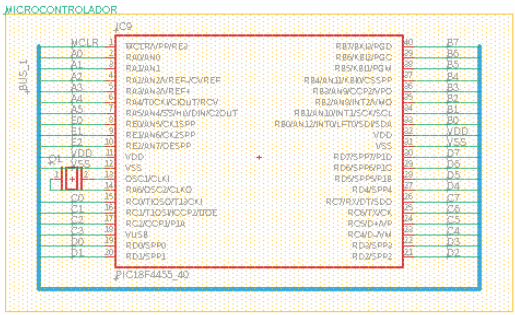
C. Marco Procedimental

1. Se procede a realizar el programa del sistema de caja fuerte, el esquemático de las conexiones se muestra en la figura.









Elaborado por:	**Revisado por:**	**Aprobado por:** Ing. Orlando Barcia Ayala
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

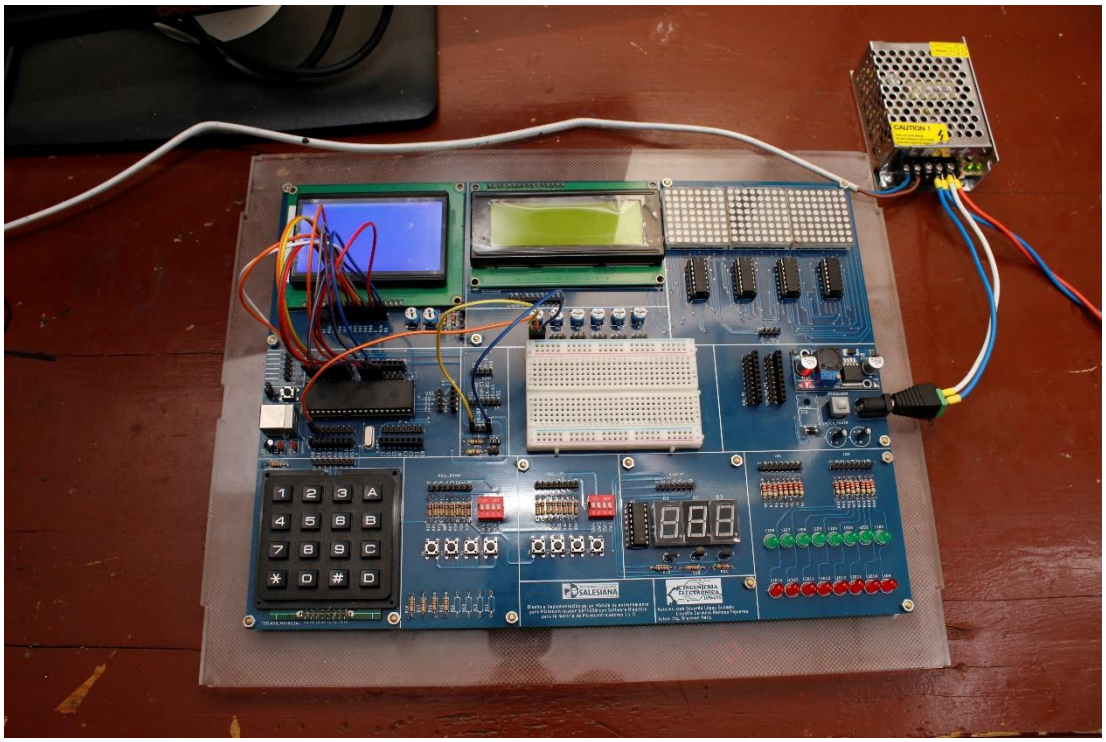


Figura 248. Esquema de conexión en PRÁCTICA 13

PIC18F4550 – B7	GLCD D7
PIC18F4550 – B6	GLCD D6
PIC18F4550 – B5	GLCD D5
PIC18F4550 – B4	GLCD D4
PIC18F4550 – B3	GLCD D3
PIC18F4550 – B2	GLCD D2
PIC18F4550 – B1	GLCD D1
PIC18F4550 – B0	GLCD D0
PIC18F4550 – A3	GLCD RESET
PIC18F4550 – A2	GLCD E
PIC18F4550 – A1	GLCD RW
PIC18F4550 – A0	GLCD RS
PIC18F4550 – D0	TECLADO C1

PIC18F4550 – D1	TECLADO C2
PIC18F4550 – D2	TECLADO C3
PIC18F4550 – D3	TECLADO C4
PIC18F4550 – D4	TECLADO F1
PIC18F4550 – D5	TECLADO F2
PIC18F4550 – D6	TECLADO F3
PIC18F4550 – D7	TECLADO F4

Tabla 17: Conexiones de PRÁCTICA 13

2. Ejecutar el programa PIC Compiler V 5 de la familia de microchip basado en el lenguaje C como se muestra en la figura.

Figura 249. PIC Compiler V 5 – PRÁCTICA 13

3. Al ejecutar el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_13** como se muestra en la figura-

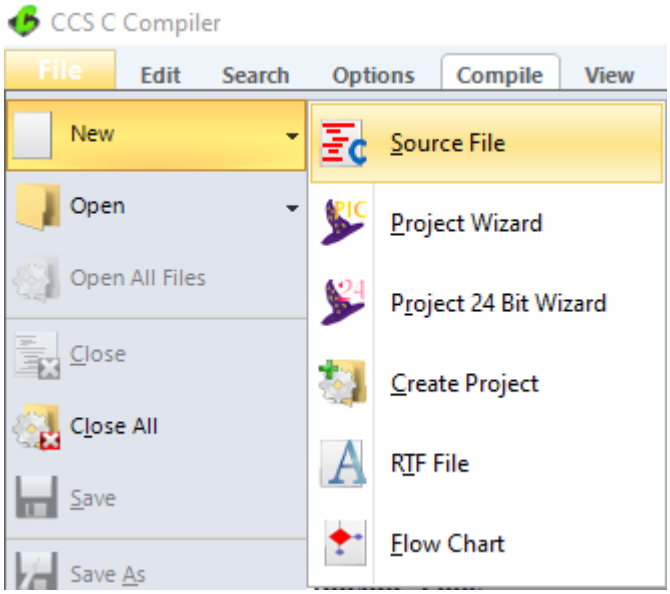


Figura 250. Creación de archivo de código en lenguaje C – PRÁCTICA 13

4. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

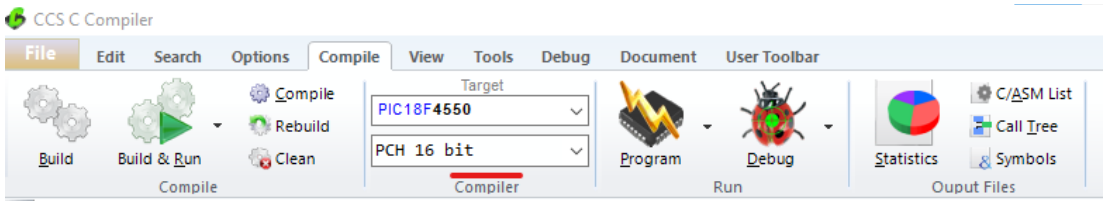


Figura 251. Configuración del microcontrolador en el compilador – PRÁCTICA 13

5. Se procede a llamar a la librería del microcontrolador, definir los fusibles, velocidad del cristal donde de define la velocidad y los puertos a utilizar como se muestra en la figura.

```
#include <18F4550.h>
#FUSES NOWDT                    //No Watch Dog Timer
#FUSES HS                       //High speed Osc (> 4mhz)
#FUSES PUT                      //Power Up Timer
#FUSES NOPROTECT                //Code not protected from reading
#use delay(clock=20Mhz)     //Declaración de la velocidad del cristal de quarzo
#include <stdlib.h>        //libreria para transformar datos enteros en tipo char (texto)
#include <math.h>          //libreria para calculo
#include <kbd.c>           //Libreria del teclado
#include "sT7920.h"        //libreria del GLCD
#include "GRAPHICS.C"      //libreria de los comandos del GLCD
```

Figura 252. Declaración de librerías -PRÁCTICA 13

6. Declarar el comando para utilizar el UART y declarar los pines y velocidad a utilizar y los puertos con su respectivo registro como se muestra en la figura.

```
#byte porta=0x0f80//bit del puerto a
#byte portb=0x0f81//bit del puerto b
#byte portc=0x0f82//bit del puerto c
#byte portd=0x0f83//bit del puerto d
//variables del programa
char k='\n';
int t=0;// variable de tecla
char v[5];// arreglo V
long clave=1586;//variable de clave
long temporal;//variable temporal para clave
int a,b,c,d;//variable de digitos
//MENSAJES A PUBLICAR EN EL GLCD
char mens1[ ]="Bienvenido";
char mens2[ ]="INGRESE CLAVE";
char mens3[ ]="INGRESO CORRECTO";
char mens4[ ]="INGRESO INCORRECTO";
char m[ ]="*";
```

Figura 253. Declaración de registro del puerto y variables del proceso y del GLCD -PRÁCTICA 13

7. Para el ingreso de las teclas es necesario declarar un sub-método el cual retorne un valor cuando se presione una tecla como se muestra en la figura.

```
char tecla(void)
{
char c;
do{ //espera hasta que se presione una tecla
c=kbd_getc(); //Captura valor del teclado
}
while(c=='\n');
return(c);
}
```

Figura 254. Sub método para tecla-PRÁCTICA 13

8. En el bloque main se comienza activando las librerías del teclado y del Glcd, posteriormente se muestra un mensaje de bienvenida durante 1 segundo y se limpia la pantalla como se muestra en la figura.

```
void main(){

kbd_init() ;//INICIO DE teclado
glcd_init_graph();//INICIO DE GLCD
glcd_fillscreen(OFF);//Limpia el GLC
glcd_update();//Actualiza la pantalla para visualizar comandos D
glcd_text57(1,20,mens1,2,1);//comando para escribir mensaje 1
glcd_update();//Actualiza la pantalla para visualizar comandos
 delay_ms(1000);//espera de 1 segundo
 glcd_fillscreen(OFF);//Limpia el GLCD
glcd_update();//Actualiza la pantalla para visualizar comandos
```

Figura 255. Lazo principal parte 1-PRÁCTICA 13

9. Posteriormente se declara un lazo de repetición donde se toma los valores de cada digito mediante el método tecla, se convierte a un valor numérico y se muestre an el Glcd, al tener los 4 digitos presionado se realiza una comparación entre los valores ingresados y el valor guardado en la memoria que pertenece a la clave del sistema, donde si la clave es incorrecta se mostrara en el GLCD un mensaje indicado ese estado y el sistema se quedara bloqueado de manera que sea necesario un reset físico al microcontrolador, caso contrario mostrara un pantalla en el GLCD de acceso concedido permitiendo el ingreso como se muestra en la figura.

```
void main(){
kbd_init() ;//INICIO DE teclado
glcd_init_graph();//INICIO DE GLCD
glcd_fillscreen(OFF);//Limpia el GLC
glcd_update();//Actualiza la pantalla para visualizar comandos D
glcd_text57(1,20,mens1,2,1);//comando para escribir mensaje 1
glcd_update();//Actualiza la pantalla para visualizar comandos
 delay_ms(1000);//espera de 1 segundo
 glcd_fillscreen(OFF);//Limpia el GLCD
glcd_update();//Actualiza la pantalla para visualizar comandos
   while(true)
   {
      glcd_fillscreen(OFF);//Limpia el GLCD
      glcd_text57(10,10,mens2,1,1);//comando para escribir mensaje 2
      glcd_update();//Actualiza la pantalla para visualizar comandos
      delay_ms(10);
      k=tecla();//sub metodo para tecla
      a=k-48;//conversión de texto a numero
      itoa(a,10,v);//conversión de numero a texto
      glcd_text57(40,30,v,2,1);//comando para escribir variable a
      glcd_update();//Actualiza la pantalla para visualizar comandos
       delay_ms(10);
      k=tecla();//sub metodo para tecla
      b=k-48;
      itoa(b,10,v);//conversión de numero a texto
      glcd_text57(50,30,v,2,1);//comando para escribir variable b
      glcd_update();//Actualiza la pantalla para visualizar comandos
       delay_ms(10);
      k=tecla();//sub metodo para tecla
      c=k-48;//conversión de texto a numero
      itoa(c,10,v);//conversión de numero a texto
      glcd_text57(60,30,v,2,1);//comando para escribir variable c
      glcd_update();//Actualiza la pantalla para visualizar comandos
       delay_ms(10);
      k=tecla();//sub metodo para tecla
      d=k-48;//conversión de texto a numero
      itoa(d,10,v);//conversión de numero a texto
      glcd_text57(70,30,v,2,1);//comando para escribir variable d
      glcd_update();//Actualiza la pantalla para visualizar comandos
      delay_ms(10);
      temporal=a*1000+b*100+c*10+d;//Clave temporal
      if(clave==temporal)//comparación entre clave temporal y clave en memo
      {
      glcd_fillscreen(OFF);//Limpia el GLCD
      glcd_text57(10,30,mens3,1,1);//comando para escribir mensaje 3
      glcd_update();//Actualiza la pantalla para visualizar comandos
      delay_ms(5000);
      }
      else
      {
      glcd_fillscreen(OFF);//Limpia el GLCD
      glcd_text57(10,30,mens4,1,1);//comando para escribir mensaje 4
      glcd_update();//Actualiza la pantalla para visualizar comandos
      delay_ms(5000);
      }
   }
}
```

Figura 256. Segmento principal – Práctica 13

10. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura 180 Y 181.

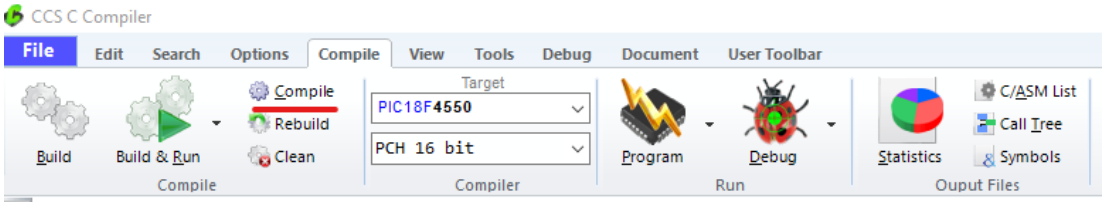


Figura 257. Botón para compilar el programa emisor – Práctica 13

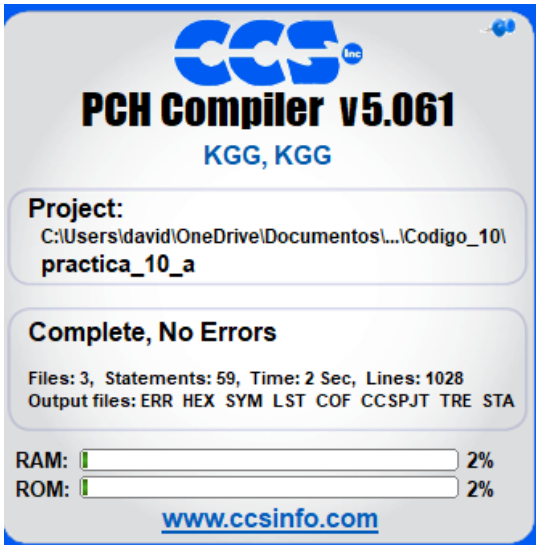


Figura 258. Compilación del programa emisor – Práctica 13

11. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 259. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 13

12. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

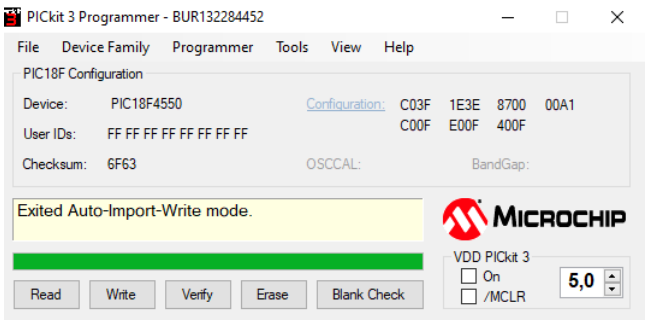


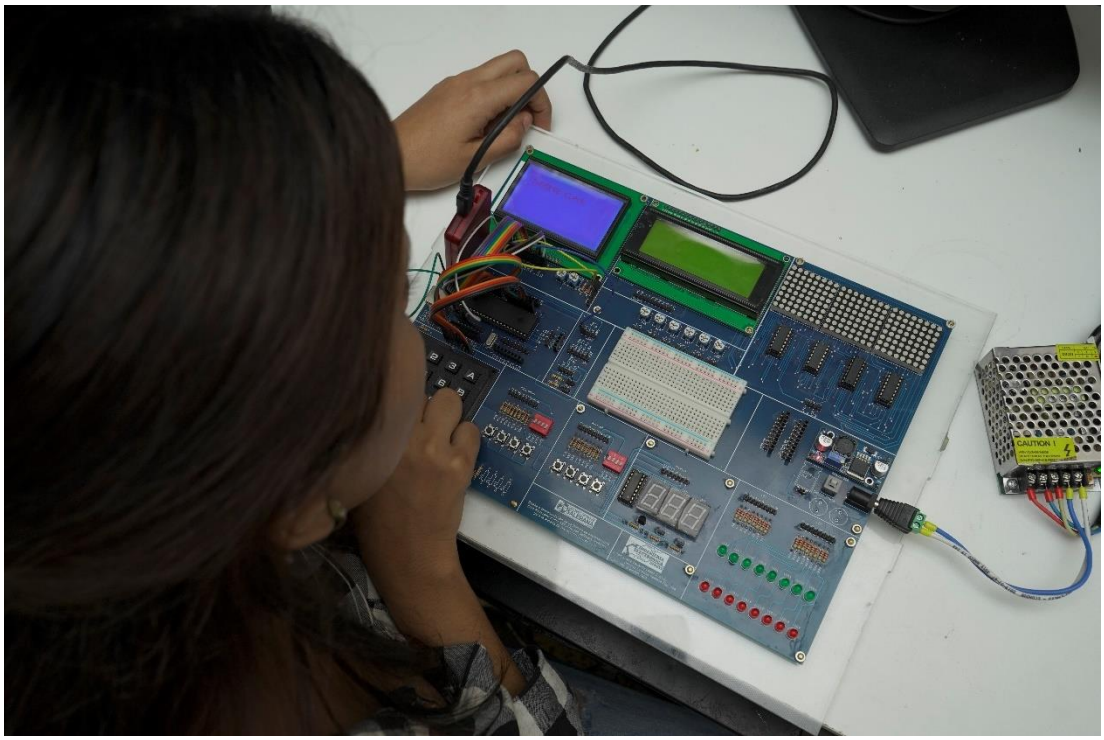
Figura 260. programación de PIC mediante el Pickit 3 – Práctica 13

D. Registro de Resultados

En la décima tercera práctica se diseñó un aplicativo con el módulo didactico el cual simula el sistema de una caja fuerte con una pantalla grafica GLD, un

teclado matricial y un controlador Pic 18f4550, de manera que se consulta una clave de acceso al usuario.

E. Conexiones físicas de la PRÁCTICA



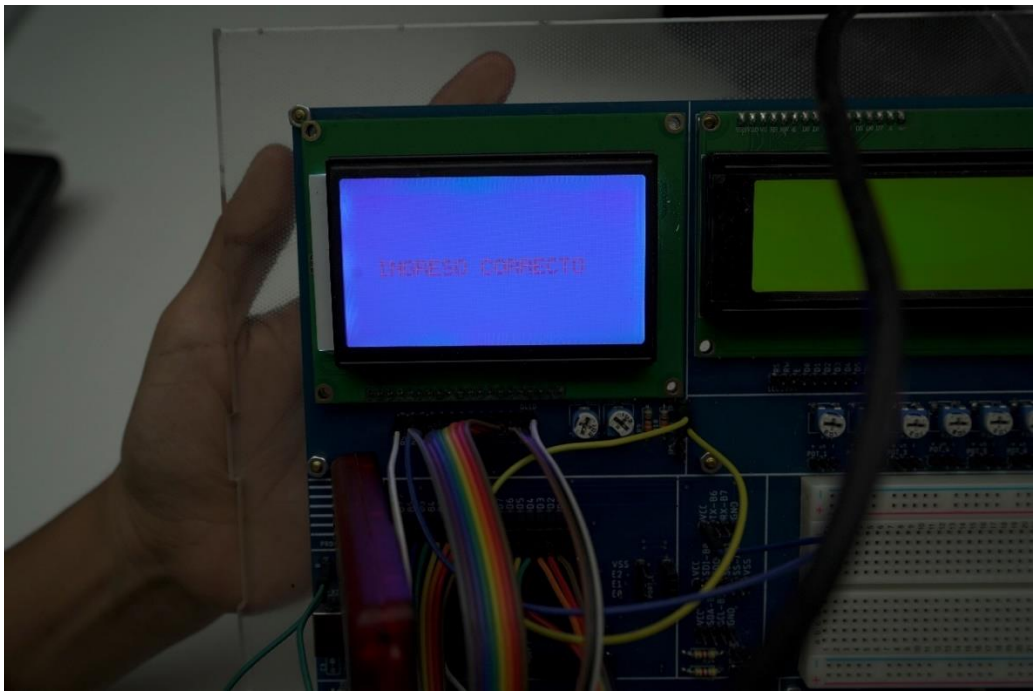


Figura 261. Conexiones físicas - Práctica 13

ANEXO 3.14: RESOLUCIÓN DE PRÁCTICA 14
AUTOMATIZACIÓN INDUSTRIAL

PRÁCTICA 14

NÚMERO DE ESTUDIANTES: 20

DOCENTE

ING. BREMNEN MARINO VELIZ NOBOA MSc.

TIEMPO ESTIMADO: 2 HORAS

TEMA: "MATRICES LED CREANDO PALABRAS, NÚMEROS MEDIANTE EL PIC18F4550."

Elaborado por:	**Revisado por:**	**Aprobado por:** Ing. Orlando Barcia Ayala
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

A. Objetivo General

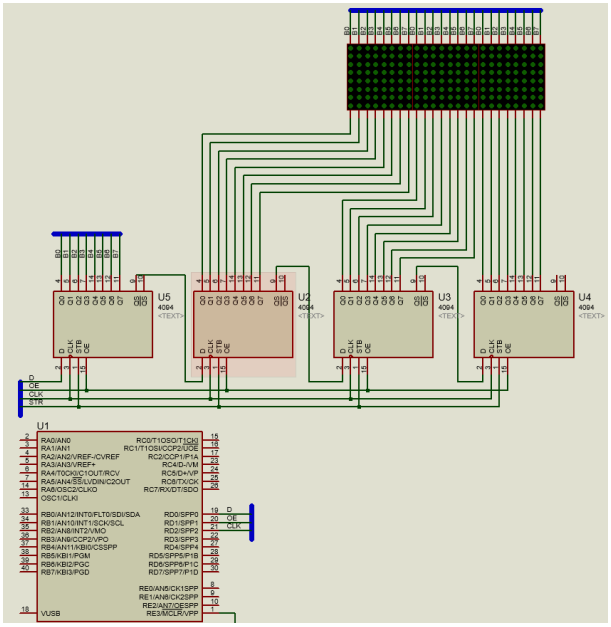
- Implementar una guía para el uso de matrices led creando palabras, números mediante el pic18f4550.

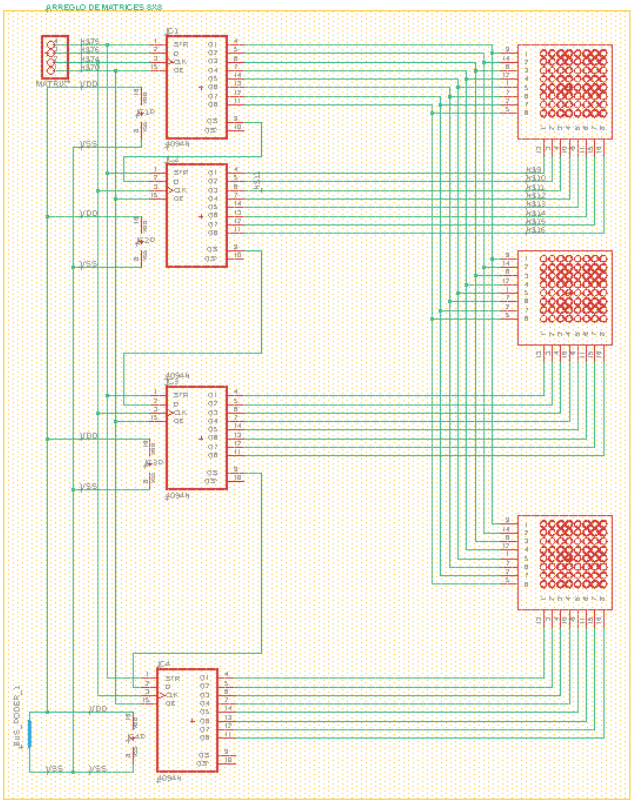
B. Objetivos Específicos

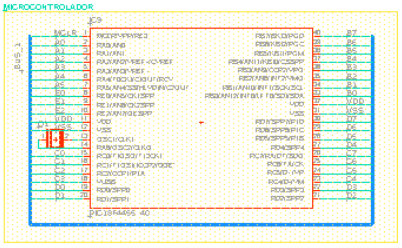
- Explicar en detalle la creación de carácter y números con el programa xlr8 para su uso en una matriz led.
- Programar con un desplazamiento de registro los valores a enviar a la matriz led.
- Realizar el cableado respectivo entre el módulo didáctico y la matriz led.

C. Marco Procedimental

1. Se procede a realizar la conexión entre el microcontrolador y las matrices led como se muestra en el esquema de la figura







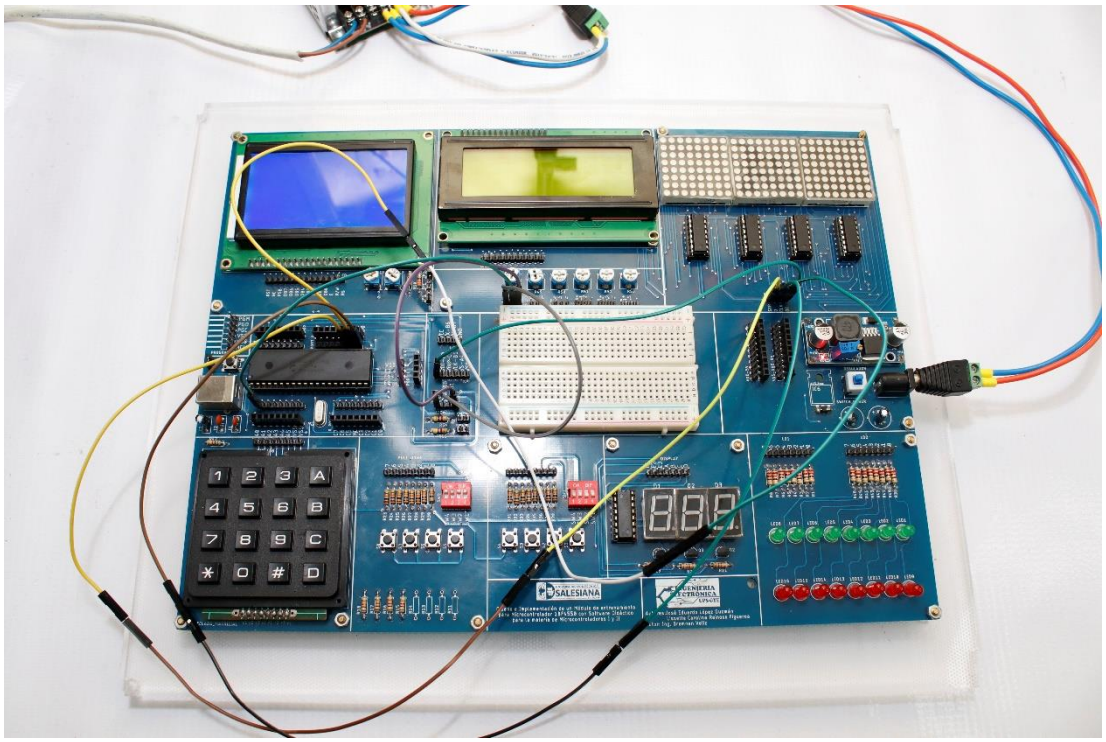


Figura 262. Esquema de conexión en PRÁCTICA 14

PIC18F4550 –VCC	OE
PIC18F4550 – D2	STR
PIC18F4550 – D1	CLK
PIC18F4550 – D0	D

Tabla 18: Conexiones de PRÁCTICA 14

2. Ejecutar el programa PIC Compiler V 5 de la familia de microchip basado en el lenguaje C como se muestra en la figura.

Figura 263. PIC Compiler V 5 – PRÁCTICA 14

3. Al ejecutar el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_14** como se muestra en la figura-

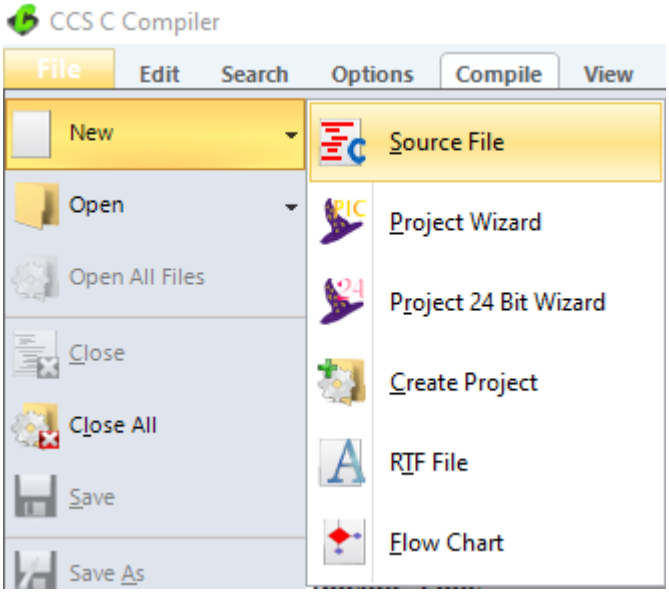


Figura 264. Creación de archivo de código en lenguaje C – PRÁCTICA 14

4. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

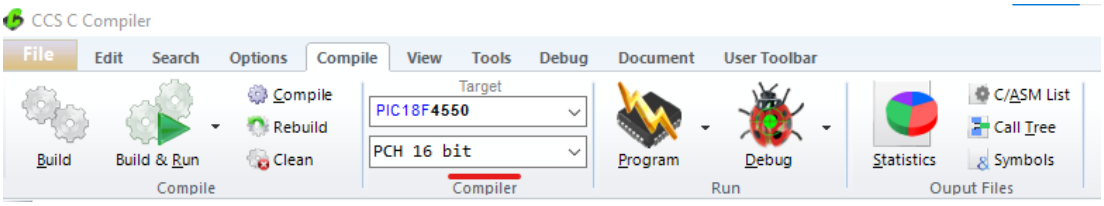


Figura 265. Configuración del microcontrolador en el compilador – PRÁCTICA 14

5. Se procede en a ejecutar un navegador web y ubicar la siguiente dirección http://xlr8.at/8x8hexbin/ donde al momento de acceder se visualiza un aplicativo donde se podrá dibujar los caracteres que se deseen mostrar en la matriz led como se muestra en la figura.

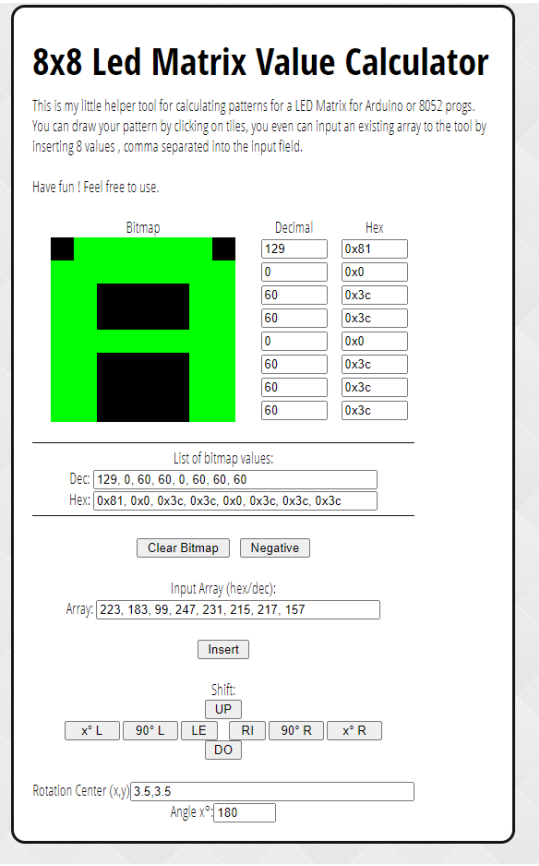


Figura 266. Aplicativo 8x8 led matriz led PRÁCTICA 14

6. Se procede a guardar los valores en un arreglo entero de 8 bits con tamaño de 8 espacios en la siguiente tabla se muestran los valores de cada letra y numero.

CARACTER	Arreglo de 8 espacios
A	129, 0, 60, 60, 0, 60, 60, 60
B	1, 0, 60, 60, 0, 60, 60, 3
C	0, 0, 63, 63, 63, 63, 0, 0
D	3, 1, 60, 60, 60, 60, 1, 3
E	0, 0, 63, 1, 1, 63, 0, 0
F	0, 0, 63, 1, 1, 63, 63, 63
G	0, 63, 63, 48, 48, 60, 60, 0
H	60, 60, 60, 0, 0, 60, 60, 60
I	0, 0, 231, 231, 231, 231, 0, 0
J	0, 0, 231, 231, 231, 231, 7, 7
K	60, 57, 51, 7, 7, 51, 57, 62
L	63, 63, 63, 63, 63, 63, 0, 0
M	126, 60, 90, 102, 126, 126, 126, 126
N	126, 62, 94, 110, 118, 122, 124, 126
O	0, 60, 126, 126, 126, 126, 60, 0
P	0, 60, 60, 0, 63, 63, 63, 63
Q	131, 51, 123, 123, 3, 123, 51, 132
R	1, 125, 125, 1, 3, 119, 115, 121
S	1, 1, 127, 127, 1, 253, 253, 1
T	0, 0, 231, 231, 231, 231, 231, 231
U	255, 60, 60, 60, 60, 60, 0, 0
V	126, 126, 126, 126, 126, 189, 219, 231
W	102, 102, 102, 102, 102, 165, 219, 255

X	126, 189, 219, 231, 231, 219, 189, 126
Y	126, 189, 219, 231, 231, 231, 231, 231
Z	128, 254, 253, 251, 247, 239, 223, 128
0	195, 189, 189, 189, 189, 189, 189, 195
1	231, 199, 167, 103, 231, 231, 231, 0
2	0, 253, 251, 247, 239, 223, 191, 0
3	129, 253, 253, 129, 129, 253, 253, 129
4	189, 189, 189, 129, 253, 253, 253, 253
5	129, 191, 191, 129, 253, 253, 129, 255
6	129, 191, 191, 129, 189, 189, 129, 255
7	131, 251, 251, 251, 251, 251, 251, 255
8	129, 189, 189, 129, 189, 189, 129, 255
9	129, 189, 189, 129, 253, 253, 129, 255

Tabla 19: Valores de los arreglos para la matriz led

7. Se procede a llamar a la librería del microcontrolador, definir los fusibles, velocidad del cristal donde de define la velocidad y los puertos a utilizar como se muestra en la figura.

```
#include<18f4550.h>              //Libreria del pic
#FUSES HS                       //Fusible para cristal
#use delay(crystal=20000000)
#USE SPI (MASTER, CLK=PIN_D1, DI=PIN_A0, DO=PIN_D0, ENABLE=PIN_D2, BAUD=200000, MODE=0, BITS=32, LSB_FIRST)
```

Figura 267. Declaración de librerías y registros -PRÁCTICA 14

8. Se procede a declarar el método principal, las variables que contendrán los valores indicados a la tabla previamente declarada y un lazo while donde se tendrá un arreglo de 32 bits enviados por el spi mediante el comando **spi_fer** que es el tamaño de los 4 registros de desplazamiento tomando el primer valor como el de multiplexación y los otros tres valores como datos que luego se unen en un solo dato de 32 bits mediante el

comando **make32** pertenecientes a los valores de los caracteres a mostrarse en las matrices led como se muestra en la figura.

```
void main()
{

int8 m0[8]={128,64,32,16,8,4,2,1};//Dato 1 de multiplexacion
int8 m1[8]={0, 60, 126, 126, 126, 126, 60, 0};//dato de caracter 0
int8 m2[8]={0, 0, 231, 231, 231, 231, 0, 0};//dato de caracter K
int8 m3[8]={189, 189, 189, 129, 253, 253, 253, 253};//dato de caracter 4
int32 x;// dato a enviar por el spi
int8 aa=0;//bit para la multiplexacion
while(true)
{
   x=make32(m0[aa],m1[aa],m2[aa],m3[aa]);//Se une los datos
   spi_xfer(x);//se envia x el buffer
   delay_us(10);//espera
aa++;//aumento para cambio de multiplexacion
if (aa>=8){//cuando el bit de multiplexacion pasa 8 vuelve a 0
aa=0;
}
    }
}
```

Figura 268. Método principal PRÁCTICA 14

9. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura 180 Y 181.

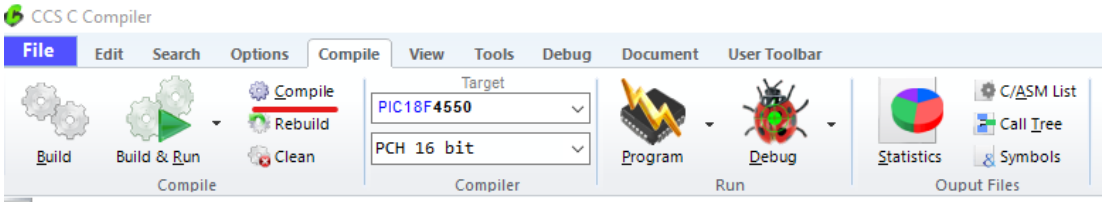


Figura 269. Botón para compilar el programa – Práctica 14

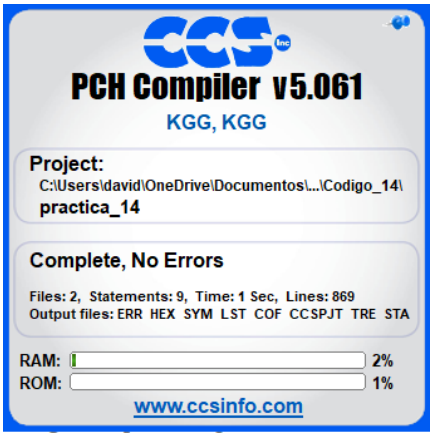


Figura 270. Compilación del programa – Práctica 14

10. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 271. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 14

11. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

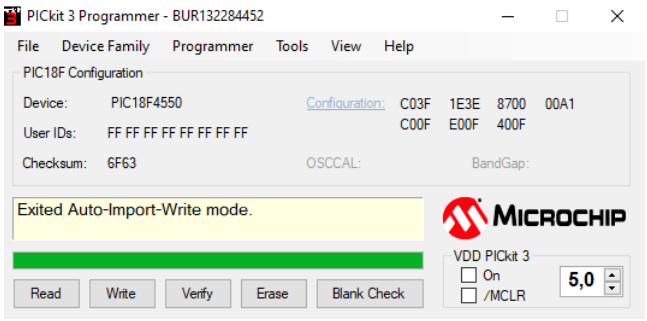
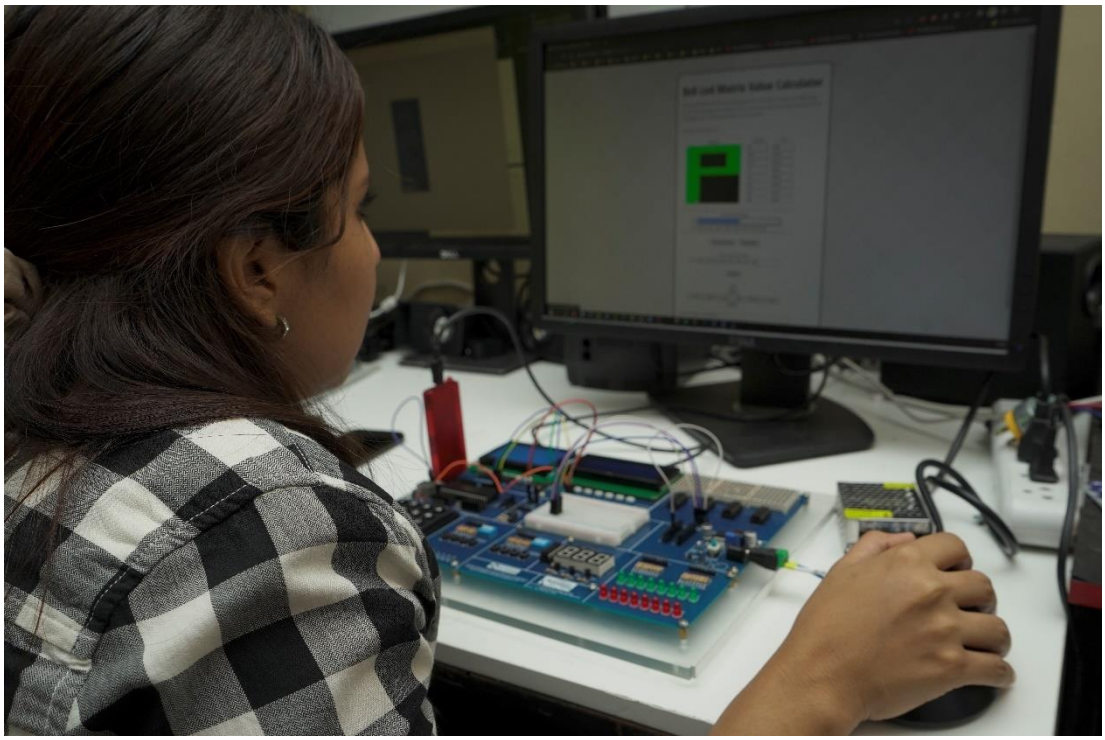


Figura 272. programación de PIC mediante el Pickit 3 – Práctica 14

D. Registro de Resultados

En la décima cuarta práctica se una guía del uso de las herramientas 8pixel para la creación de caracteres y visualización en tres matrices led comunicándose con un controlador Pic 18f4550 .

E. Resultados de PRÁCTICA



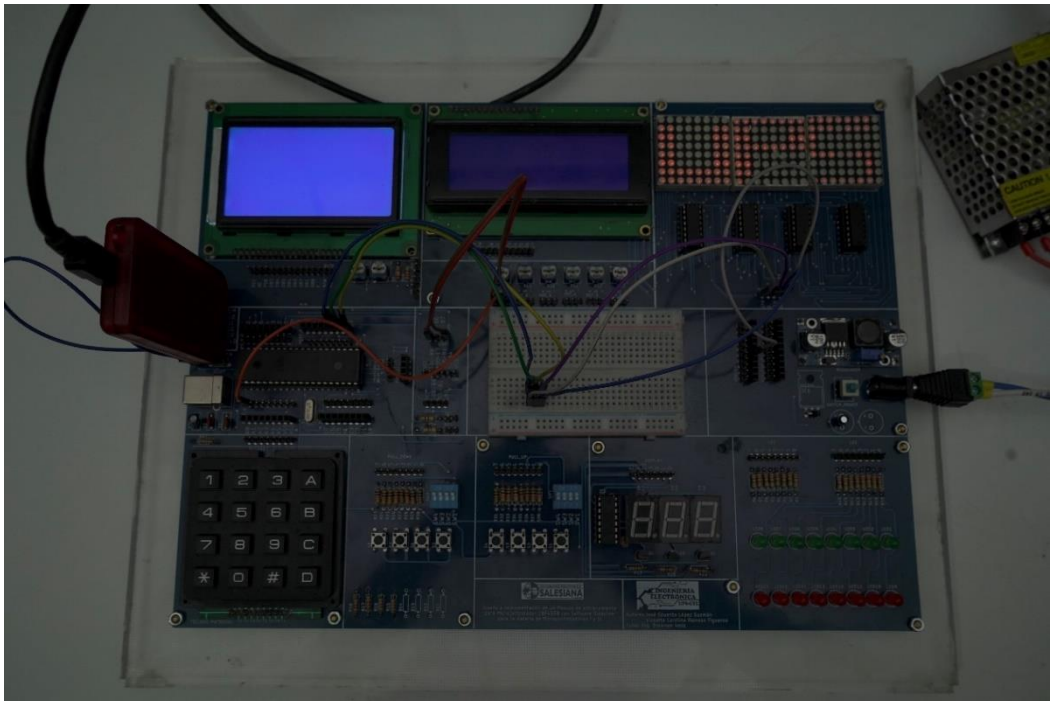


Figura 273. Resultados - Práctica 15

ANEXO 3.15: RESOLUCIÓN DE PRÁCTICA 15
AUTOMATIZACIÓN INDUSTRIAL

PRÁCTICA 15

NÚMERO DE ESTUDIANTES: 20

DOCENTE

ING. BREMNEN MARINO VELIZ NOBOA MSc.

TIEMPO ESTIMADO: 2 HORAS

TEMA: “ENVÍO Y RECIBO DE DATOS UTILIZANDO LCD 20X4, TECLADO MATRICIAL MEDIANTE EL PIC18F4550 Y LAS INTERRUPCIONES”

A. Objetivo General

- Implementar un aplicativo que realice la conversión análoga a digital y visualización en una matriz led por registros de desplazamientos.

B. Objetivos Específicos

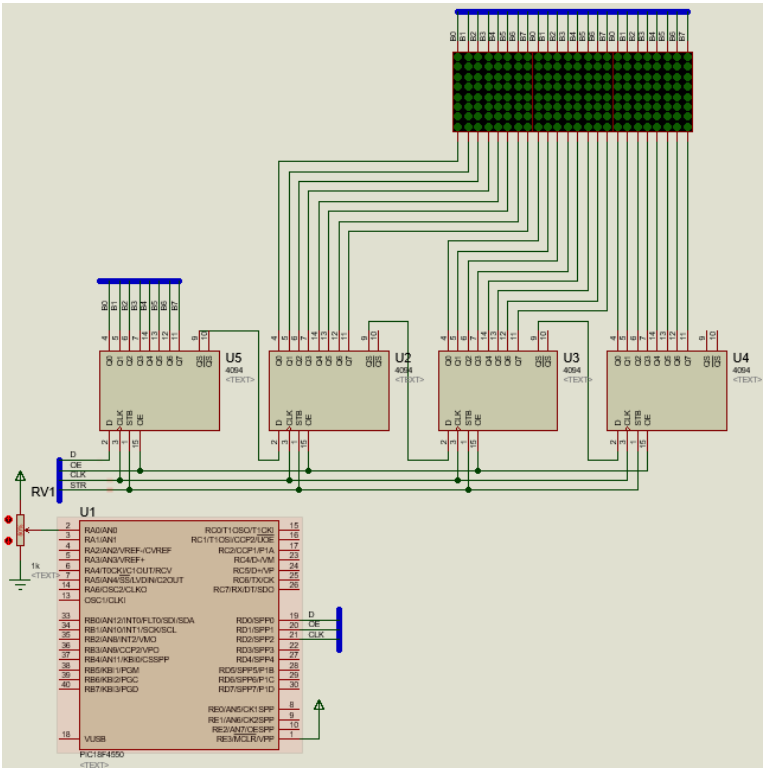
- Explicar en detalle la creación de carácter y números con el programa xlr8 para su uso en una matriz led.
- Programar mediante un controlador pic una conversión análoga a digital y mostrar en una matriz led mediante un desplazamiento de registro los valores.
- Realizar el cableado respectivo entre el módulo didáctico y la matriz led.

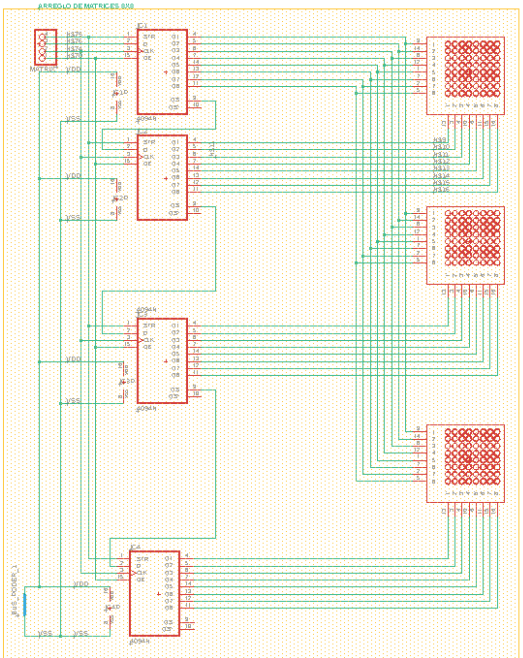
C. Marco Procedimental

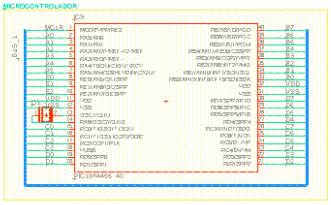
1. Se procede a realizar la conexión entre el microcontrolador, el potenciómetro y las matrices led como se muestra en el esquema de la figura

PIC18F4550 –VCC	OE
PIC18F4550 – D2	STR
PIC18F4550 – D1	CLK
PIC18F4550 – D0	D
PIC18F4550 – VCC	POT VCC
PIC18F4550 – GND	POT GND
PIC18F4550 – A0	S

Tabla 20: Conexiones de PRÁCTICA 15


U5
U2
U3
U4
4094
RV1
U1
PIC18F4550





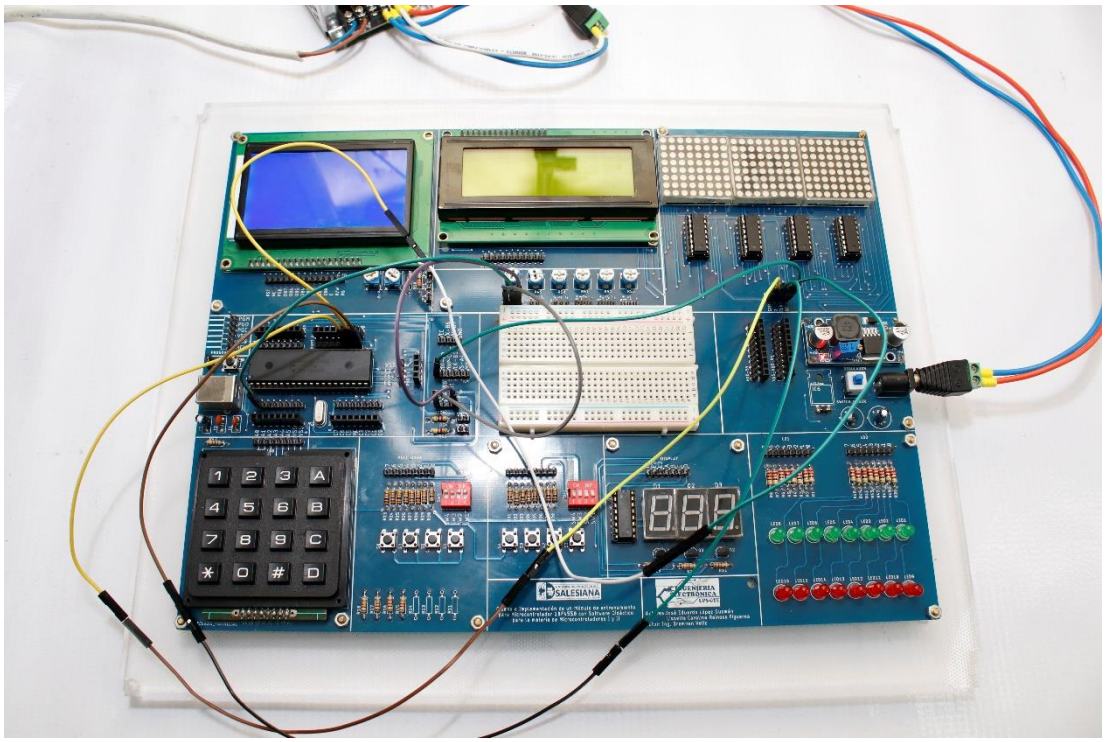


Figura 274. Esquema de conexión en PRÁCTICA 15

2. Ejecutar el programa PIC Compiler V 5 de la familia de microchip basado en el lenguaje C como se muestra en la figura.

Figura 275. PIC Compiler V 5 – PRÁCTICA 15

3. Al ejecutar el programa dar click en File >> New>> Source File, de esta manera se procede a crear el archivo denominado **PRÁCTICA_15** como se muestra en la figura-

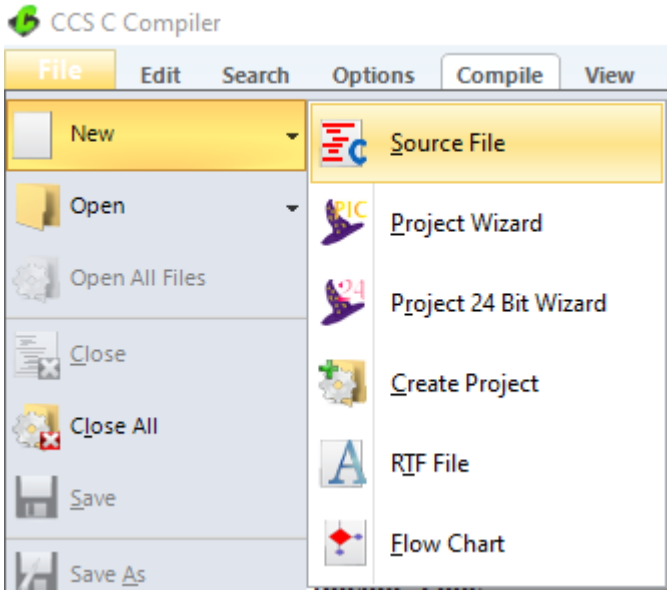


Figura 276. Creación de archivo de código en lenguaje C – PRÁCTICA 15

4. Se procede a configurar la tarjeta el microcontrolador a utilizar **PIC18F4550** y los bits **16** bit como se muestra en la figura

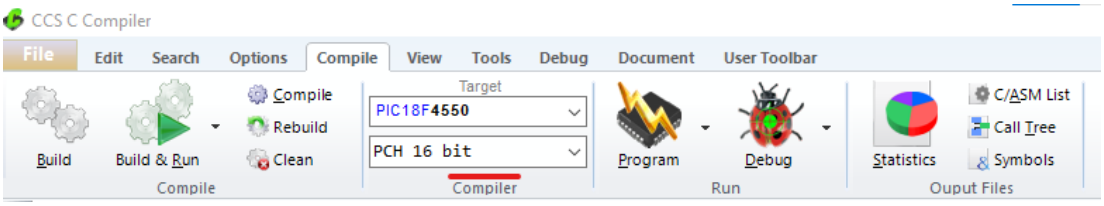


Figura 277. Configuración del microcontrolador en el compilador – PRÁCTICA 15

5. Se procede en a ejecutar un navegador web y ubicar la siguiente dirección http://xlr8.at/8x8hexbin/ donde al momento de acceder se visualiza un

aplicativo donde se podrá dibujar los caracteres que se deseen mostrar en la matriz led como se muestra en la figura.

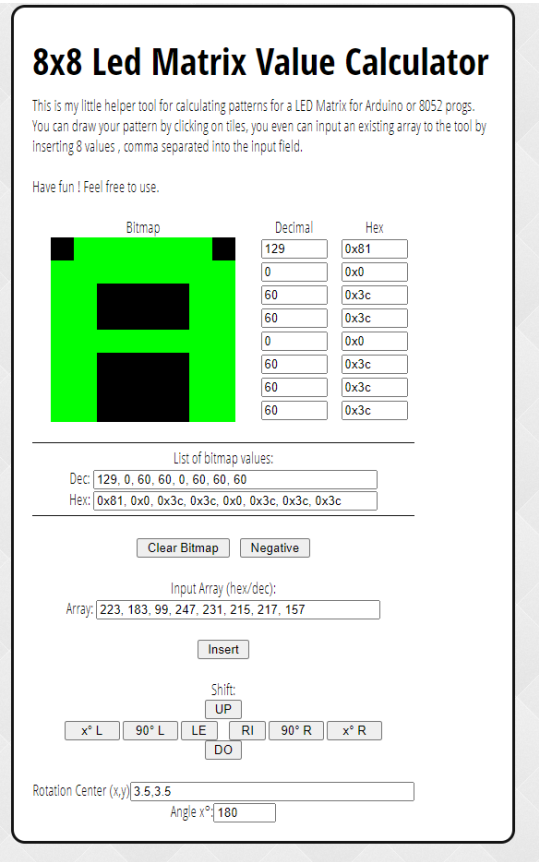


Figura 278. Aplicativo 8x8 led matriz led PRÁCTICA 15

6. Se procede a guardar los valores en un arreglo entero de 8 bits con tamaño de 8 espacios en la siguiente tabla se muestran los valores de cada letra y numero.

CARACTER	Arreglo de 8 espacios
A	129, 0, 60, 60, 0, 60, 60, 60
B	1, 0, 60, 60, 0, 60, 60, 3
C	0, 0, 63, 63, 63, 63, 0, 0
D	3, 1, 60, 60, 60, 60, 1, 3
E	0, 0, 63, 1, 1, 63, 0, 0

F	0, 0, 63, 1, 1, 63, 63, 63
G	0, 63, 63, 48, 48, 60, 60, 0
H	60, 60, 60, 0, 0, 60, 60, 60
I	0, 0, 231, 231, 231, 231, 0, 0
J	0, 0, 231, 231, 231, 231, 7, 7
K	60, 57, 51, 7, 7, 51, 57, 62
L	63, 63, 63, 63, 63, 63, 0, 0
M	126, 60, 90, 102, 126, 126, 126, 126
N	126, 62, 94, 110, 118, 122, 124, 126
O	0, 60, 126, 126, 126, 126, 60, 0
P	3, 125, 125, 125, 3, 127, 127, 127
Q	131, 51, 123, 123, 3, 123, 51, 132
R	1, 125, 125, 1, 3, 119, 115, 121
S	1, 1, 127, 127, 1, 253, 253, 1
T	0, 0, 231, 231, 231, 231, 231, 231
U	126, 126, 126, 126, 126, 126, 0, 0
V	126, 126, 126, 126, 126, 189, 219, 231
W	102, 102, 102, 102, 102, 165, 219, 255
X	126, 189, 219, 231, 231, 219, 189, 126
Y	126, 189, 219, 231, 231, 231, 231, 231
Z	128, 254, 253, 251, 247, 239, 223, 128
0	195, 189, 189, 189, 189, 189, 189, 195
1	231, 199, 167, 103, 231, 231, 231, 0
2	0, 253, 251, 247, 239, 223, 191, 0
3	129, 253, 253, 129, 129, 253, 253, 129
4	189, 189, 189, 129, 253, 253, 253, 253
5	129, 191, 191, 129, 253, 253, 129, 255

6	129, 191, 191, 129, 189, 189, 129, 255
7	131, 251, 251, 251, 251, 251, 251, 255
8	129, 189, 189, 129, 189, 189, 129, 255
9	129, 189, 189, 129, 253, 253, 129, 255

Tabla 21: Valores de los arreglos para la matriz led PRÁCTICA 15

7. Se procede a llamar a la librería del microcontrolador, definir los fusibles, el bit del conversor análogo a digital, velocidad del cristal donde de define la velocidad, declara el spi con sus respectivos pines como se muestra en la figura.

```
#include<18f4550.h>              //Libreria del pic
#DEVICE ADC=8
#FUSES HS                       //Fusible para cristal
#use delay(crystal=20000000)
#USE SPI (MASTER, CLK=PIN_D1, DI=PIN_A0, DO=PIN_D0, ENABLE=PIN_D2, BAUD=200000, MODE=0, BITS=32, LSB_FIRST)
```

Figura 279. Declaración de librerías y registros -PRÁCTICA 15

8. Declarar las variables para conversión análoga a digital, variables para separar los datos en unidades, decena y centena, los arreglos correspondientes a los datos de los caracteres del 0 al 9 y el dato para multiplexación de los valores del spi como se muestra en la figura.

```
int8 valor;
long digito1,digito2,digito3;//Variables para separación de valores unidad, decena, centena
int8 m[8]={128,64,32,16,8,4,2,1};//Dato 1 de multiplexación  del display
int8 aa=0;//Bit para multiplexacion de los arreglos
// asigna variables para numero en arreglos
int8 m0[8]={195, 189, 189, 189, 189, 189, 189, 195};//dato de caracter 0
int8 m1[8]={231, 199, 167, 103, 231, 231, 231, 0};//dato de caracter 1
int8 m2[8]={0, 253, 251, 247, 239, 223, 191, 0};//dato de caracter 2
int8 m3[8]={129, 253, 253, 129, 129, 253, 253, 129};//dato de caracter 3
int8 m4[8]={189, 189, 189, 129, 253, 253, 253, 253};//dato de caracter 4
int8 m5[8]={129, 191, 191, 129, 253, 253, 129, 255};//dato de caracter 5
int8 m6[8]={129, 191, 191, 129, 189, 189, 129, 255};//dato de caracter 6
int8 m7[8]={131, 251, 251, 251, 251, 251, 251, 255};//dato de caracter 7
int8 m8[8]={129, 189, 189, 129, 189, 189, 129, 255};//dato de caracter 8
int8 m9[8]={129, 189, 189, 129, 253, 253, 129, 255};//dato de caracter 9
```

Figura 280. Variables PRÁCTICA 15

9. Declarar un sub-método el cual reciba un valor entero y envié de vuelta el valor correspondiente a cada carácter a mostrarse en las matrices led como se muestra en la figura.

```
int8 dato_d(int a)
{
     if (a==0){return m0[aa];}//envia caracter 0
else if (a==1){return m1[aa];}//envia caracter 1
else if (a==2){return m2[aa];}//envia caracter 2
else if (a==3){return m3[aa];}//envia caracter 3
else if (a==4){return m4[aa];}//envia caracter 4
else if (a==5){return m5[aa];}//envia caracter 5
else if (a==6){return m6[aa];}//envia caracter 6
else if (a==7){return m7[aa];}//envia caracter 7
else if (a==8){return m8[aa];}//envia caracter 8
else if (a==9){return m9[aa];}//envia caracter 9
}
```

Figura 281. Método de conversión para caracteres 15

10. En el método principal se procede a configurar los registros para el adc como el reloj y la dirección del adc a utilizar, en el lazo de repetición se posiciona el canal a utilizar el canal 0 y se realiza la adquisición y guardado en la variable valor, después se separa dicho valor en unidad, decena y centena, posteriormente se une binariamente para formar un dato de 32 bits y se envia por el comando **spi_xfer** al registro de desplazamiento, se incrementa el bit del registro denominado **aa** y se designa la regla que cuando supere el valor de 8 vuelva a 0, de esta manera el bit ira realizando un registro binario de los valores a mostrarse como se aprecia en la figura

```
void main()
{
setup_adc(ADC_CLOCK_INTERNAL);
setup_adc_ports(AN0);
int32 x;// dato a enviar por el spi
//bit para la multiplexacion
while(true)
{
set_adc_channel(0);//Configura el canal a utilizar
delay_us(10);//espera 10 microsegundo
valor=read_adc();//se obtiene valor analogo a digital
digito1 = valor / 100;//se separa la centena como digito unico
digito2 = (valor % 100) / 10;//se separa la decena como digito unico
digito3 = (valor % 100) % 10 / 1;//se separa la unidad como digito unico
x=make32(m[aa],dato_d(digito1),dato_d(digito2),dato_d(digito3));
//Se une los datos
spi_xfer(x);//se envia x el buffer
delay_us(10);//espera
aa++;//aumento para cambio de multiplexacion
if (aa>=8){//cuando el bit de multiplexacion pasa 8 vuelve a 0
aa=0;
}
    }
}
```

Figura 282. Método de principal PRÁCTICA 15

11. Se compila el programa mediante el botón de compilación ubicado en la barra de tareas en la parte superior como se muestra en la figura 180 Y 181.

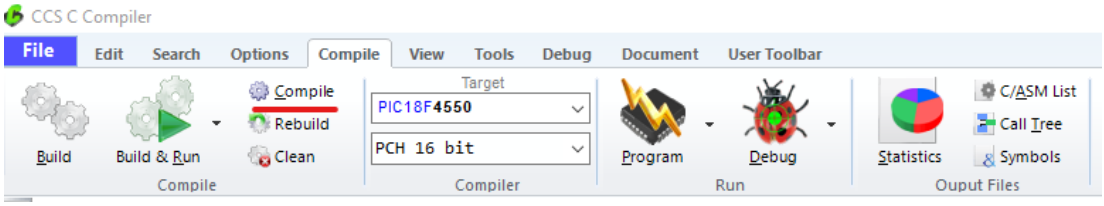


Figura 283. Botón para compilar el programa – Práctica 15

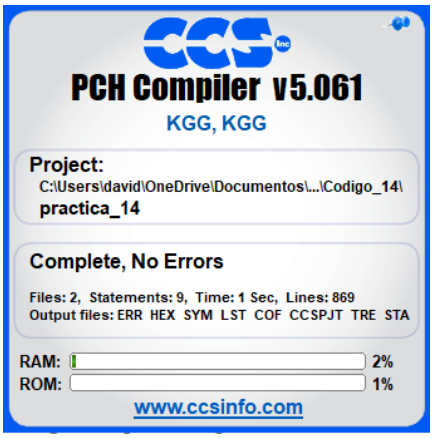


Figura 284. Compilación del programa – Práctica 14

12. Al compilar se crean una serie de archivos o registros pertenecientes al programa, el archivo con formato **.hex** es el que se carga en el microcontrolador, se procede a realizar la carga del archivo hexadecimal conectando el programador en la tarjeta como se muestra en la figura

Figura 285. Conexión del Pickit 3 con la tarjeta de entrenamiento – Práctica 14

13. Ejecutamos el pickit 3 la conexión ocurre de manera automática , caso contrario dar clic en **Tools>Check Communication,** al lograr la conexión

seleccionamos **File>>Import hex,** buscamos el archivo a programar y dar clic en **Write** como se muestra en la figura.

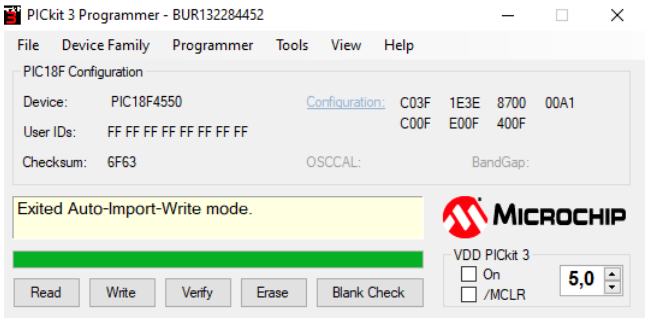
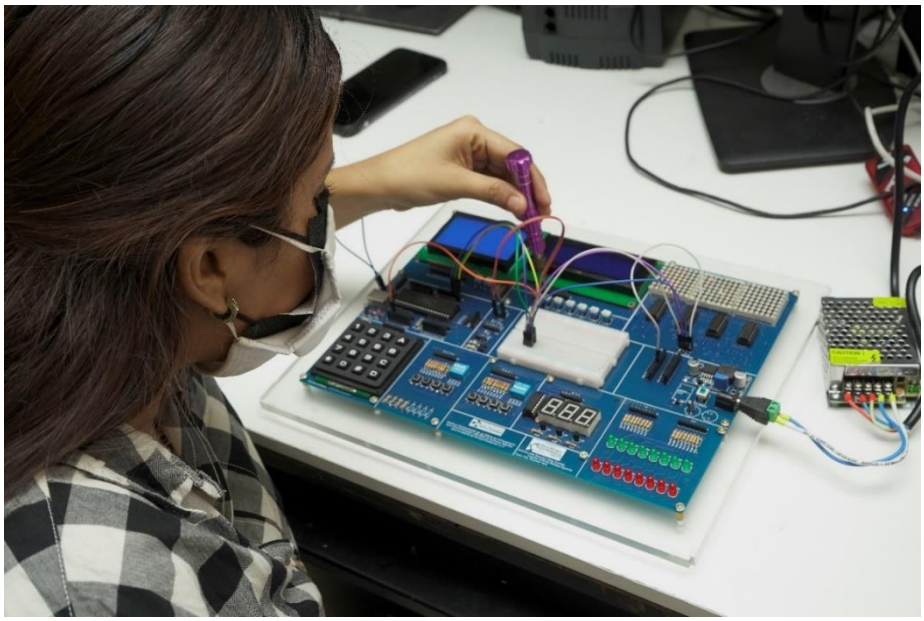


Figura 286. programación de PIC mediante el Pickit 3 – Práctica 15

D. Registro de Resultados

En la décima quinta PRÁCTICA se obtiene como resultado la visualización de una conversión análoga a digital en unas matrices led mediante la multiplexación y envió de datos por 4 registros de desplazamiento 4094 comunicándose con un controlador Pic 18f4550.

E. Resultados físicos de la PRÁCTICA



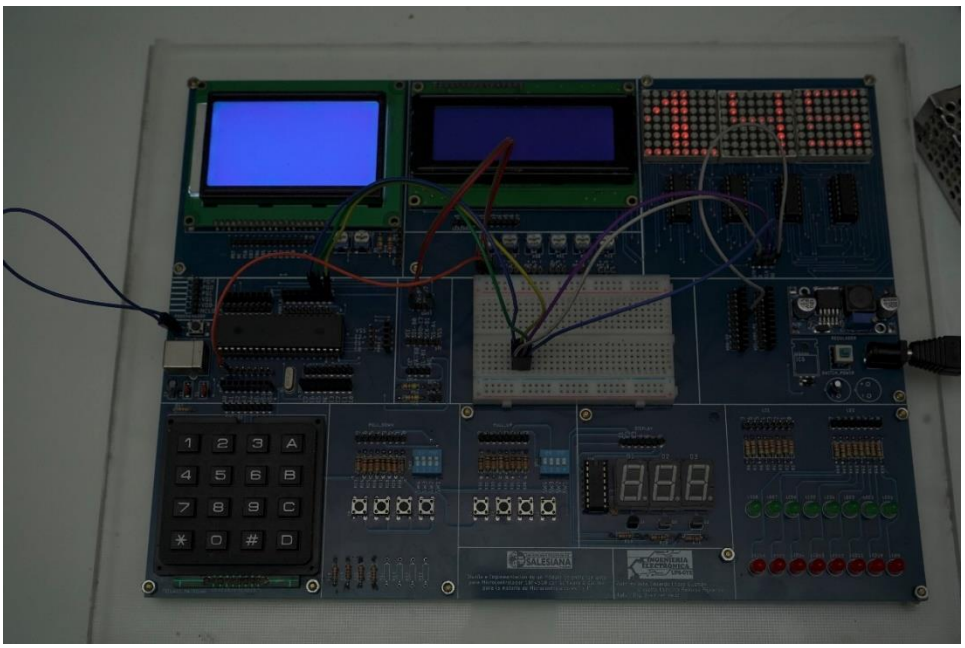


Figura 287. Resultados - Práctica 15

ANEXO DE LIBRERIAS

LIBRERÍA ST7920.h

```
/************************************************************************
*   LCD graphics driver for Digole 12864w with ST7920 driver using      *
*   CCS software. May work with other ST7920 driven LCD. It has         *
*   the following Pin assignments                                       *
*                                                                       *
*   Pin 1 ------------> Gnd                                             *
*   Pin 2 ------------> +5volts                                         *
*   Pin 3 ------------> Contrast                                        *
*   Pin 4 ------------> Register Select                                 *
*   Pin 5 ------------> Read/Write                                      *
*   Pin 6 ------------> Enable                                          *
*   Pin 7-14 ---------> Data bits                                       *
*   Pin 15 -----------> PSB (parallel=high & serial=low)                *
*   Pin 16 -----------> NoConnection                                    *
*   Pin 17 -----------> Reset                                           *
*   Pin 18 -----------> Vout                                            *
*   Pin 19 -----------> +5volts                                         *
*   Pin 20 -----------> Gnd                                             *

************************************************************************/
// change the pin assignment depending on your circuit

#define  rs  PIN_D5              //COMMNAD/DATA SELECT
#define  rw  PIN_D6              //READ/WRITE SELECT
#define  e   PIN_D7              //ENABLE SIGNAL
#define  rst PIN_D4              //RESET SIGNAL

#define ON   1
#define OFF   0
#define XVAL 16  // 16 X 16 or 256 for there is 8 word values for the upper
and lower
#define YVAL 32
```

```
// PSB is tied to Vcc for this driver because this driver uses Parallel
// operation.
// data is sent using port B so change output_b() to other ports you
//want to use. Dont for get to change the Busy pin @ lcd_check_busy

#define GLCD_WIDTH   128



////////////////////////////////////////////////////////////////////////////
//The following are the functions included in this driver file
// glcd_readbyte();
// glcd_instruction( instruction );
// glcd_data( data ); - data can be an array of characters!
// glcd_check_busy();
// glcd_update(); -must be called always after writing a pixel or using
functions
//                from GRAPHICS.C .. Only applicaticable in Graphing mode
// glcd_fillscreen( ON or OFF);
// glcd_init_graph(); initialize for graphing mode
// glcd_init_basic(); initilize for accessing the stored Characters
//                    you can use glcd_data() for writing text
// glcd_pixel(x coordinate, y coordinate, ON or OFF);
//            -WORKS WITH GRAPHIC.C  from CCS Drivers
// glcd_plot_image(width,height,X coor, Y coor, inverse);
//            -plots the image[] array. Declare it first before this driver.
//             or modify this driver
//
////////////////////////////////////////////////////////////////////////////



typedef union
{
  int16 word;
```

```
  int8 nbyte[2];
} Dots;

typedef struct
{
  int1 refresh;
  Dots pix[YVAL][XVAL];   // Max dimensions for display (x,y) = (128,32)
  } GD_RAM;             //  (0,0) corresponds to upper lefthand corner.

GD_RAM gdram;



unsigned int8 glcd_readByte (unsigned int1 address)
{
  unsigned int8 data;   // Stores the data read from the LCD
  if(address==1){
     output_high(rs);
  }
  if(address==0){
     output_low(rs);
  }
  output_high(rw);//GLCD_RW = RW_READ;      // Set for reading
  output_high(e);//GLCD_E = 1;      // Pulse the enable pin
  delay_us(1);
  data=input_b();      // Get the data from the display's output register
  output_low(e);//GLCD_E = 0;
  return (data);
}



void glcd_check_busy(){
   int1 busy=1;
   output_low(rs);      // LOW RS and High RW will put the lcd to
   output_high(rw);      // read busy flag and address counter
   while(busy){         // will cycle until busy flag is 0
```

```
      output_high(e);
      if(!input(PIN_B7)){
         busy=0;
      }
      output_low(e);
   }
}

void glcd_instruction(unsigned char x){
   glcd_check_busy();      //must be satisfied before sending instruction
   output_low(rs);      // LOW RS and LOW RW will put the lcd to
   output_low(rw);      // Write instruction mode
   output_b(x);         // 8bit data to bus
   output_high(e);      // enable
   delay_us(1);
   output_low(e);      // disable
}
void glcd_data(unsigned char x){
   glcd_check_busy();
   output_high(rs);      // HIGH RS and LOW RW will put the lcd to
   output_low(rw);      // Write data register mode
   output_b(x);
   output_high(e);
   delay_us(1);
   output_low(e);
}


void glcd_fillScreen (unsigned int1 color)
{
  int8 v, h;
  int16 d;


  d = (color == ON ? 0xFFFFL : 0x0000L);
```

```
  for (v=0; v < YVAL; v++)
  {
    for (h=0; h < XVAL; h++)
    {
      gdram.pix[v][h].word = d;
    }
  }
  gdram.refresh = TRUE;
}


void glcd_update ()
{
  int8 v, h;


  if (gdram.refresh)
  {
    for (v=0; v <YVAL; v++)
    {
      glcd_instruction( 0x80 | v);   // Set Vertical Address.
      glcd_instruction( 0x80 | 0);   // Set Horizontal Address.

      for (h=0; h <XVAL; h++)
      {
        glcd_data( gdram.pix[v][h].nbyte[1]);   // Write High Byte.
        glcd_data(  gdram.pix[v][h].nbyte[0]);   // Write Low Byte.
      }
    }
    gdram.refresh = FALSE;
  }
}

void glcd_init_graph(){
```

```
  delay_ms(40);
  output_low(rst);        //reset LCD
  delay_us(1);
  output_high(rst);       //LCD normal operation
  glcd_instruction(0x30);  //set 8 bit operation and basic instruction set
  delay_us(144);
  glcd_instruction(0x0C);  //display on cursor off and char blink off
  delay_us(100);
  glcd_instruction(0x01);  //display clear
  delay_ms(10);
  glcd_instruction(0x06);  //entry mode set
  delay_us(72);
 glcd_instruction(0x34);   // Select extended instruction set.
 delay_ms (10);
 glcd_instruction(0x36);   // Graphic display ON.
 delay_ms (10);

 glcd_fillScreen (OFF);
 glcd_update ();

}

void glcd_init_basic(){
  delay_ms(40);
  output_low(rst);        //reset LCD
  delay_us(1);
  output_high(rst);       //LCD normal operation
  glcd_instruction(0x30);  //set 8 bit operation and basic instruction set
  delay_us(144);
  glcd_instruction(0x0C);  //display on cursor off and char blink off
  delay_us(100);
  glcd_instruction(0x01);  //display clear
  delay_ms(10);
  glcd_instruction(0x06);  //entry mode set
  delay_us(72);
```

```
}

void glcd_pixel(int8 x, int8 y, int1 color)
{
  int8 v, h, b;
  if(y>31){x += 128; y-= 32;};
  v = y;
  h = x/16;
  b = 15 - (x%16);

  if (color == ON) bit_set (gdram.pix[v][h].word, b);
  else bit_clear (gdram.pix[v][h].word, b);

  gdram.refresh = TRUE;
}
//
void glcd_plot_image(ROM char *img,int width,int height,int x,int y,int
inverse)//CARREGA IMAGEM
{
   unsigned int i=0, j=0, k=0;
   unsigned int16 count=0;

   //glcd_fillScreen(OFF);                    //Clears the screen (opt.)
   for(j=0;j<height;j++)//linhas
     {
        for(;i<width;)//colunas
        {
          for(k=0;k<8;k++)//bite
           {
              if(inverse)
              {//          0  ,0  ,
                glcd_pixel(i+x,j+y,~bit_test(img[count],(k)));
              }
              else
              {
```

```
                glcd_pixel(i+x,j+y,bit_test(img[count],(k)));
              }
              i++;
            }
            count++;
          }
        i=0;
        }
}
```

LIBRERÍA KBD.C

```
////////////////////////////////////////////////////////////////////////
////                        Flex_KBD.C                          ////
////              Generic keypad scan driver                    ////
////                                                            ////
////  kbd_init()   Must be called before any other function.          ////
////                                                            ////
////  c = kbd_getc(c)  Will return a key value if pressed or /0 if not ////
////                This function should be called frequently so as ////
////                not to miss a key press.                    ////
////                                                            ////
////////////////////////////////////////////////////////////////////////
```

//The PCH compiler defines this pre-processor identifier. It may be used to determine if the PCH compiler is doing the compilation.

//Keypad connection:

```
#define col0 PIN_B0
#define col1 PIN_B1
#define col2 PIN_B2
#define row0 PIN_B4
#define row1 PIN_B5
#define row2 PIN_B6
#define row3 PIN_B7
```

// Keypad layout:

```
char const KEYS[4][3] = {{'1','2','3'},
                         {'4','5','6'},
                         {'7','8','9'},
                         {'*','0','#'}};
```

#define KBD_DEBOUNCE_FACTOR 33 // 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() {
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);
                break;
       case 1   : output_high(col0);
            output_low(col1);
            output_high(col2);
                break;
       case 2   : output_high(col0);
            output_high(col1);
            output_low(col2);
```

```
                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==3)
               col=0;
           }
        }
       kbd_call_count=0;
    }
   return(kchar);
}
```

LIBRERÍA LCD.C

```
////////////////////////////////////////////////////////////////////////////
////                        LCD.C                                       ////
////              Driver for common LCD modules                         ////
////                                                                    ////
////  lcd_init()   Must be called before any other function.            ////
////                                                                    ////
////  lcd_putc(c)  Will display c on the next position of the LCD.      ////
////               \a  Set cursor position to upper left                ////
////               \f  Clear display, set cursor to upper left          ////
////               \n  Go to start of second line                      ////
////               \b  Move back one position                           ////
////            If LCD_EXTENDED_NEWLINE is defined, the \n character    ////
////            will erase all remanining characters on the current     ////
////            line, and move the cursor to the beginning of the next  ////
////            line.                                                   ////
////            If LCD_EXTENDED_NEWLINE is defined, the \r character    ////
////            will move the cursor to the start of the current        ////
////            line.                                                   ////
////                                                                    ////
////  lcd_gotoxy(x,y) Set write position on LCD (upper left is 1,1)     ////
////                                                                    ////
////  lcd_getc(x,y)   Returns character at position x,y on LCD          ////
////                                                                    ////
////  lcd_cursor_on(int1 on)   Turn the cursor on (on=TRUE) or off      ////
////            (on=FALSE).                                             ////
////                                                                    ////
////  lcd_set_cgram_char(w, *p)   Write a custom character to the CGRAM.
////
////                                                                    ////
////                                                                    ////
////  CONFIGURATION                                                     ////
////  The LCD can be configured in one of two ways: a.) port access or  ////
////  b.) pin access.  Port access requires the entire 7 bit interface  ////
////  connected to one GPIO port, and the data bits (D4:D7 of the LCD)  ////
```

```
////  connected to sequential pins on the GPIO.  Pin access              ////
////  has no requirements, all 7 bits of the control interface can       ////
////  can be connected to any GPIO using several ports.                  ////
////                                                                     ////
////  To use port access, #define LCD_DATA_PORT to the SFR location of
////
////  of the GPIO port that holds the interface, -AND- edit LCD_PIN_MAP  ////
////  of this file to configure the pin order.  If you are using a       ////
////  baseline PIC (PCB), then LCD_OUTPUT_MAP and LCD_INPUT_MAP
also must  ////
////  be defined.                                                        ////
////                                                                     ////
////  Example of port access:                                            ////
////    #define LCD_DATA_PORT getenv("SFR:PORTD")                        ////
////                                                                     ////
////  To use pin access, the following pins must be defined:             ////
////    LCD_ENABLE_PIN                                                   ////
////    LCD_RS_PIN                                                       ////
////    LCD_RW_PIN                                                       ////
////    LCD_DATA4                                                        ////
////    LCD_DATA5                                                        ////
////    LCD_DATA6                                                        ////
////    LCD_DATA7                                                        ////
////                                                                     ////
////  Example of pin access:                                             ////
////    #define LCD_ENABLE_PIN  PIN_E0                                   ////
////    #define LCD_RS_PIN      PIN_E1                                   ////
////    #define LCD_RW_PIN      PIN_E2                                   ////
////    #define LCD_DATA4       PIN_D4                                   ////
////    #define LCD_DATA5       PIN_D5                                   ////
////    #define LCD_DATA6       PIN_D6                                   ////
////    #define LCD_DATA7       PIN_D7                                   ////
////                                                                     ////
/////////////////////////////////////////////////////////////////////////////
////        (C) Copyright 1996,2010 Custom Computer Services             ////
```

```
//// This source code may only be used by licensed users of the CCS C  ////
//// compiler.  This source code may only be distributed to other      ////
//// licensed users of the CCS C compiler.  No other use, reproduction ////
//// or distribution is permitted without written permission.          ////
//// Derivative programs created using this software in object code    ////
//// form are not restricted in any way.                               ////
/////////////////////////////////////////////////////////////////////////

#ifndef __LCD_C__
#define __LCD_C__

// define the pinout.
// only required if port access is being used.
typedef struct
{                       // This structure is overlayed
   int1 enable;         // on to an I/O port to gain
   int1 rs;             // access to the LCD pins.
   int1 rw;             // The bits are allocated from
   int1 unused;         // low order up.  ENABLE will
   unsigned int     data : 4;        // be LSB pin of that port.
  #if defined(__PCD__)      // The port used will be LCD_DATA_PORT.
   unsigned int    reserved: 8;
  #endif
} LCD_PIN_MAP;

// this is to improve compatability with previous LCD drivers that accepted
// a define labeled 'use_portb_lcd' that configured the LCD onto port B.
#if ((defined(use_portb_lcd)) && (use_portb_lcd==TRUE))
 #define LCD_DATA_PORT getenv("SFR:PORTB")
#endif

#if defined(__PCB__)
   // these definitions only need to be modified for baseline PICs.
   // all other PICs use LCD_PIN_MAP or individual LCD_xxx pin definitions.
/*                          EN, RS,   RW,   UNUSED,  DATA  */
```

```
 const LCD_PIN_MAP LCD_OUTPUT_MAP =  {0,  0,    0,    0,       0};
 const LCD_PIN_MAP LCD_INPUT_MAP =   {0,  0,    0,    0,       0xF};
#endif

////////////////////// END CONFIGURATION //////////////////////////////////

#ifndef LCD_ENABLE_PIN
  #define lcd_output_enable(x) lcdlat.enable=x
  #define lcd_enable_tris()   lcdtris.enable=0
#else
  #define lcd_output_enable(x) output_bit(LCD_ENABLE_PIN, x)
  #define lcd_enable_tris()  output_drive(LCD_ENABLE_PIN)
#endif

#ifndef LCD_RS_PIN
  #define lcd_output_rs(x) lcdlat.rs=x
  #define lcd_rs_tris()   lcdtris.rs=0
#else
  #define lcd_output_rs(x) output_bit(LCD_RS_PIN, x)
  #define lcd_rs_tris()  output_drive(LCD_RS_PIN)
#endif

#ifndef LCD_RW_PIN
  #define lcd_output_rw(x) lcdlat.rw=x
  #define lcd_rw_tris()   lcdtris.rw=0
#else
  #define lcd_output_rw(x) output_bit(LCD_RW_PIN, x)
  #define lcd_rw_tris()  output_drive(LCD_RW_PIN)
#endif

// original version of this library incorrectly labeled LCD_DATA0 as
LCD_DATA4,
// LCD_DATA1 as LCD_DATA5, and so on.  this block of code makes the
driver
// compatible with any code written for the original library
```

```
#if (defined(LCD_DATA0) && defined(LCD_DATA1) &&
defined(LCD_DATA2) && defined(LCD_DATA3) && !defined(LCD_DATA4)
&& !defined(LCD_DATA5) && !defined(LCD_DATA6) &&
!defined(LCD_DATA7))
  #define  LCD_DATA4    LCD_DATA0
  #define  LCD_DATA5    LCD_DATA1
  #define  LCD_DATA6    LCD_DATA2
  #define  LCD_DATA7    LCD_DATA3
#endif

#ifndef LCD_DATA4
#ifndef LCD_DATA_PORT
  #if defined(__PCB__)
    #define LCD_DATA_PORT      0x06     //portb
    #define set_tris_lcd(x)   set_tris_b(x)
  #else
    #if defined(PIN_D0)
     #define LCD_DATA_PORT      getenv("SFR:PORTD")     //portd
    #else
     #define LCD_DATA_PORT      getenv("SFR:PORTB")     //portb
    #endif
  #endif
#endif

#if defined(__PCB__)
  LCD_PIN_MAP lcd, lcdlat;
  #byte lcd = LCD_DATA_PORT
  #byte lcdlat = LCD_DATA_PORT
#elif defined(__PCM__)
  LCD_PIN_MAP lcd, lcdlat, lcdtris;
  #byte lcd = LCD_DATA_PORT
  #byte lcdlat = LCD_DATA_PORT
  #byte lcdtris = LCD_DATA_PORT+0x80
#elif defined(__PCH__)
  LCD_PIN_MAP lcd, lcdlat, lcdtris;
```

```
   #byte lcd = LCD_DATA_PORT
   #byte lcdlat = LCD_DATA_PORT+9
   #byte lcdtris = LCD_DATA_PORT+0x12
#elif defined(__PCD__)
   LCD_PIN_MAP lcd, lcdlat, lcdtris;
   #word lcd = LCD_DATA_PORT
   #word lcdlat = LCD_DATA_PORT+2
   #word lcdtris = LCD_DATA_PORT-0x02
#endif
#endif   //LCD_DATA4 not defined

#ifndef LCD_TYPE
   #define LCD_TYPE 2          // 0=5x7, 1=5x10, 2=2 lines
#endif

#ifndef LCD_LINE_TWO
   #define LCD_LINE_TWO 0x40    // LCD RAM address for the second line
#endif

#ifndef LCD_LINE_LENGTH
   #define LCD_LINE_LENGTH 20
#endif

unsigned int8 lcd_read_nibble(void);

unsigned int8 lcd_read_byte(void)
{
   unsigned int8 low,high;

 #if defined(__PCB__)
   set_tris_lcd(LCD_INPUT_MAP);
 #else
  #if (defined(LCD_DATA4) && defined(LCD_DATA5) &&
defined(LCD_DATA6) && defined(LCD_DATA7))
   output_float(LCD_DATA4);
```

```
   output_float(LCD_DATA5);
   output_float(LCD_DATA6);
   output_float(LCD_DATA7);
  #else
   lcdtris.data = 0xF;
  #endif
 #endif

   lcd_output_rw(1);
   delay_cycles(1);
   lcd_output_enable(1);
   delay_cycles(1);
   high = lcd_read_nibble();

   lcd_output_enable(0);
   delay_cycles(1);
   lcd_output_enable(1);
   delay_us(1);
   low = lcd_read_nibble();

   lcd_output_enable(0);

 #if defined(__PCB__)
   set_tris_lcd(LCD_OUTPUT_MAP);
 #else
  #if (defined(LCD_DATA4) && defined(LCD_DATA5) &&
defined(LCD_DATA6) && defined(LCD_DATA7))
   output_drive(LCD_DATA4);
   output_drive(LCD_DATA5);
   output_drive(LCD_DATA6);
   output_drive(LCD_DATA7);
  #else
   lcdtris.data = 0x0;
  #endif
 #endif
```

```
   return( (high<<4) | low);
}

unsigned int8 lcd_read_nibble(void)
{
  #if (defined(LCD_DATA4) && defined(LCD_DATA5) &&
defined(LCD_DATA6) && defined(LCD_DATA7))
   unsigned int8 n = 0x00;

   /* Read the data port */
   n |= input(LCD_DATA4);
   n |= input(LCD_DATA5) << 1;
   n |= input(LCD_DATA6) << 2;
   n |= input(LCD_DATA7) << 3;

   return(n);
  #else
   return(lcd.data);
  #endif
}

void lcd_send_nibble(unsigned int8 n)
{
  #if (defined(LCD_DATA4) && defined(LCD_DATA5) &&
defined(LCD_DATA6) && defined(LCD_DATA7))
   /* Write to the data port */
   output_bit(LCD_DATA4, bit_test(n, 0));
   output_bit(LCD_DATA5, bit_test(n, 1));
   output_bit(LCD_DATA6, bit_test(n, 2));
   output_bit(LCD_DATA7, bit_test(n, 3));
  #else
   lcdlat.data = n;
  #endif
```

```
   delay_cycles(1);
   lcd_output_enable(1);
   delay_us(2);
   lcd_output_enable(0);
}

void lcd_send_byte(unsigned int8 address, unsigned int8 n)
{
  #if defined(__PCB__)
   set_tris_lcd(LCD_OUTPUT_MAP);
  #else
   lcd_enable_tris();
   lcd_rs_tris();
   lcd_rw_tris();
  #endif

   lcd_output_rs(0);
   while ( bit_test(lcd_read_byte(),7) ) ;
   lcd_output_rs(address);
   delay_cycles(1);
   lcd_output_rw(0);
   delay_cycles(1);
   lcd_output_enable(0);
   lcd_send_nibble(n >> 4);
   lcd_send_nibble(n & 0xf);
}

#if defined(LCD_EXTENDED_NEWLINE)
unsigned int8 g_LcdX, g_LcdY;
#endif

void lcd_init(void)
{
   unsigned int8 i;
```

```
  unsigned int8 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_output_enable(0);
  lcd_output_rs(0);
  lcd_output_rw(0);

 #if defined(__PCB__)
  set_tris_lcd(LCD_OUTPUT_MAP);
 #else
  #if (defined(LCD_DATA4) && defined(LCD_DATA5) &&
defined(LCD_DATA6) && defined(LCD_DATA7))
  output_drive(LCD_DATA4);
  output_drive(LCD_DATA5);
  output_drive(LCD_DATA6);
  output_drive(LCD_DATA7);
  #else
  lcdtris.data = 0x0;
  #endif
  lcd_enable_tris();
  lcd_rs_tris();
  lcd_rw_tris();
 #endif

  delay_ms(15);
  for(i=1;i<=3;++i)
  {
     lcd_send_nibble(3);
     delay_ms(5);
  }

  lcd_send_nibble(2);
```

```
   delay_ms(5);
   for(i=0;i<=3;++i)
      lcd_send_byte(0,LCD_INIT_STRING[i]);

  #if defined(LCD_EXTENDED_NEWLINE)
   g_LcdX = 0;
   g_LcdY = 0;
  #endif
}

void lcd_gotoxy(unsigned int8 x, unsigned int8 y)
{
   unsigned int8 address;

   if(y!=1)
      address=LCD_LINE_TWO;
   else
      address=0;

   address+=x-1;
   lcd_send_byte(0,0x80|address);

  #if defined(LCD_EXTENDED_NEWLINE)
   g_LcdX = x - 1;
   g_LcdY = y - 1;
  #endif
}

void lcd_putc(char c)
{
   switch (c)
   {
      case '\a'   :  lcd_gotoxy(1,1);     break;

      case '\f'   :  lcd_send_byte(0,1);
```

```
                delay_ms(2);
               #if defined(LCD_EXTENDED_NEWLINE)
                g_LcdX = 0;
                g_LcdY = 0;
               #endif
                break;

    #if defined(LCD_EXTENDED_NEWLINE)
     case '\r'   :  lcd_gotoxy(1, g_LcdY+1);   break;
     case '\n'   :
       while (g_LcdX++ < LCD_LINE_LENGTH)
       {
         lcd_send_byte(1, ' ');
       }
       lcd_gotoxy(1, g_LcdY+2);
       break;
    #else
     case '\n'   : lcd_gotoxy(1,2);        break;
    #endif

     case '\b'   : lcd_send_byte(0,0x10);  break;

    #if defined(LCD_EXTENDED_NEWLINE)
     default     :
       if (g_LcdX < LCD_LINE_LENGTH)
       {
         lcd_send_byte(1, c);
         g_LcdX++;
       }
       break;
    #else
     default     : lcd_send_byte(1,c);     break;
    #endif
   }
}
```

```
char lcd_getc(unsigned int8 x, unsigned int8 y)
{
  char value;

  lcd_gotoxy(x,y);
  while ( bit_test(lcd_read_byte(),7) ); // wait until busy flag is low
  lcd_output_rs(1);
  value = lcd_read_byte();
  lcd_output_rs(0);

  return(value);
}

// write a custom character to the ram
// which is 0-7 and specifies which character array we are modifying.
// ptr points to an array of 8 bytes, where each byte is the next row of
//    pixels.  only bits 0-4 are used.  the last row is the cursor row, and
//    usually you will want to leave this byte 0x00.
void lcd_set_cgram_char(unsigned int8 which, unsigned int8 *ptr)
{
  unsigned int i;

  which <<= 3;
  which &= 0x38;

  lcd_send_byte(0, 0x40 | which);  //set cgram address

  for(i=0; i<8; i++)
  {
    lcd_send_byte(1, *ptr++);
  }

  #if defined(LCD_EXTENDED_NEWLINE)
   lcd_gotoxy(g_LcdX+1, g_LcdY+1);  //set ddram address
```

```
  #endif
}

void lcd_cursor_on(int1 on)
{
  if (on)
  {
    lcd_send_byte(0,0x0F);          //turn LCD cursor ON
  }
  else
  {
    lcd_send_byte(0,0x0C);          //turn LCD cursor OFF
  }
}

#endif
```

LIBRERÍA stdlib.h

```
////////////////////////////////////////////////////////////////////////////
////        (C) Copyright 1996,2007 Custom Computer Services            ////
//// This source code may only be used by licensed users of the CCS C  ////
//// compiler.  This source code may only be distributed to other      ////
//// licensed users of the CCS C compiler.  No other use, reproduction ////
//// or distribution is permitted without written permission.          ////
//// Derivative programs created using this software in object code    ////
//// form are not restricted in any way.                               ////
////////////////////////////////////////////////////////////////////////////


#ifndef _STDLIB
#define _STDLIB

//-------------------------------------------------------------------------
// Definitions and types
//-------------------------------------------------------------------------

#ifndef RAND_MAX
#define RAND_MAX  32767    // The value of which is the maximum value
                           // ... returned by the rand function
#endif


#IF (sizeof(int16*)>1)
#DEFINE LONG_POINTERS 1
#ELSE
#DEFINE LONG_POINTERS 0
#ENDIF

typedef struct {
  signed int quot;
  signed int rem;
} div_t;
```

```
typedef struct {
  signed long quot;
  signed long rem;
} ldiv_t;

#include <stddef.h>

//---------------------------------------------------------------------------
// String conversion functions
//---------------------------------------------------------------------------

/* Standard template: signed int  atoi(char * s)
 * converts the initial portion of the string s to a signed int
 * returns the converted value if any, 0 otherwise
 */
signed int atoi(char *s);

/* Syntax: signed int32  atoi32(char * s)
   converts the initial portion of the string s to a signed int32
   returns the converted value if any, 0 otherwise*/
#if (sizeof(long)==4)
 #define atoi32(s) atol(s)
#else
 signed int32 atoi32(char *s);
#endif

#if defined(__PCD__)
// The following functions only work on the 24 bit compiler
// for the 30F, 33F, 24F and 24H parts
/* Syntax: signed int48  atoi48(char * s)
   converts the initial portion of the string s to a signed int48
   returns the converted value if any, 0 otherwise*/

signed int48 atoi48(char *s);
```

```
/* Syntax: signed int64  atoi64(char * s)
   converts the initial portion of the string s to a signed int64
   returns the converted value if any, 0 otherwise*/
signed int64 atoi64(char *s);
#endif

/* Syntax: char *  itoa(signed int32 num, int8 base, char * s)
   converts the signed int32 to a string and
   returns the converted value if any, 0 otherwise*/
char * itoa(signed int32 num, unsigned int base, char * s);

/* Standard template: signed int16  atol(char * s)
 * converts the initial portion of the string s to a signed int16
 * returns the converted value if any, 0 otherwise
 */
signed long atol(char *s);

/* Standard template: int16 strtoul(char * s,char *endptr,signed int base)
 * converts the initial portion of the string s, represented as an
 * integral value of radix base  to a signed long.
 * Returns the converted value if any, 0 otherwise
 * the final string is returned in the endptr, if endptr is not null
 */
signed long strtol(char *s,char *endptr, signed int base);

/* Standard template: int16 strtoul(char * s,char *endptr,signed int base)
 * converts the initial portion of the string s, represented as an
 * integral value of radix base to a unsigned long.
 * returns the converted value if any, 0 otherwise
 * the final string is returned in the endptr, if endptr is not null
 */
unsigned long strtoul(char *s,char *endptr, signed int base);

/* Standart template: float strtof(char * s,char *endptr)
```

```
                float48 strtof48(char *s,char *endptr);
                float64 strtod(char *s,char *endptr);
 * converts the initial portion of the string s to a float32, float48 or float64,
 * returns the converted value if any, 0 otherwise
 * the final string is returned in the endptr, if endptr is not null
*/
float strtof(char *s,char **endptr);
#if defined(__PCD__)
float48 strtof48(char *s,char **endptr);
float64 strtod(char *s,char **endptr);
#else
//provided for compatibility
#define strtof48(s, e) strtof(s, e)
#define strtod(s, e) strtof(s, e)
#endif

/* Standard template: float32 atof(char * s)
 * converts the initial portion of the string s to a float.
 * returns the converted value if any, 0 otherwise
 */
#define atof(s)   strtof(s, 0)

#if defined(__PCD__)
// The following functions only work on the 24 bit compiler
// for the 30F, 33F, 24F and 24H parts

/* Standard template: float48 atof48(char * s)
 * converts the initial portion of the string s to a float.
 * returns the converted value if any, 0 otherwise
 */
#define atof48(s) strtof48(s, 0)

/* Standard template: float64 atof64(char * s)
 * converts the initial portion of the string s to a float.
 * returns the converted value if any, 0 otherwise
```

```
 */
#define atof64(s) strtod(s, 0)
#endif

/* Standard template: float32 atoe(char * s)
 * converts the initial portion of the string s to a float.
 * returns the converted value if any, 0 otherwise
 * also handles E format numbers
 */
#if !defined(__PCD__)
float atoe(char * s);
#endif

#if defined(__PCD__)
float32 atoe(char * s);
#endif

//---------------------------------------------------------------------
// Pseudo-random sequence generation functions
//---------------------------------------------------------------------

/* The rand function computes a sequence of pseudo-random integers in
 * the range 0 to RAND_MAX
 *
 * Parameters:
 *       (none)
 *
 * Returns:
 *       The pseudo-random integer
 */
unsigned int16 rand(void);

/* The srand function uses the argument as a seed for a new sequence of
 * pseudo-random numbers to be returned by subsequent calls to rand.
 *
```

```
 * Parameters:
 *      [in] seed: The seed value to start from. You might need to pass
 *
 * Returns:
 *      (none)
 *
 * Remarks
 *          The srand function sets the starting point for generating
 *      a series of pseudorandom integers. To reinitialize the
 *      generator, use 1 as the seed argument. Any other value for
 *      seed sets the generator to a random starting point. rand
 *      retrieves the pseudorandom numbers that are generated.
 *      Calling rand before any call to srand generates the same
 *      sequence as calling srand with seed passed as 1.
 *          Usually, you need to pass a time here from outer source
 *      so that the numbers will be different every time you run.
 */
void srand(unsigned int32 seed);

//-----------------------------------------------------------------------
// Memory management functions
//-----------------------------------------------------------------------

// Comming soon

//-----------------------------------------------------------------------
// Communication with the environment
//-----------------------------------------------------------------------

/* The function returns 0 always
 */
signed int8 system(char *string);

//-----------------------------------------------------------------------
// Searching and sorting utilities
```

```
//---------------------------------------------------------------------------

/* Performs a binary search of a sorted array..
 *
 * Parameters:
 *      [in] key: Object to search for
 *      [in] base: Pointer to base of search data
 *      [in] num: Number of elements
 *      [in] width: Width of elements
 *      [in] compare: Function that compares two elements
 *
 * Returns:
 *      bsearch returns a pointer to an occurrence of key in the array pointed
 *      to by base. If key is not found, the function returns NULL. If the
 *      array is not in order or contains duplicate records with identical keys,
 *      the result is unpredictable.
 */
//void *bsearch(const void *key, const void *base, size_t num, size_t width,
//              int (*compare)(const void *, const void *));

/* Performs the shell-metzner sort (not the quick sort algorithm). The contents
 * of the array are sorted into ascending order according to a comparison
 * function pointed to by compar.
 *
 * Parameters:
 *      [in] base: Pointer to base of search data
 *      [in] num: Number of elements
 *      [in] width: Width of elements
 *      [in] compare: Function that compares two elements
 *
 * Returns:
 *      (none)
 */
//void *qsort(const void *base, size_t num, size_t width,
//              int (*compare)(const void *, const void *));
```

```
//---------------------------------------------------------------------------
// Integer arithmetic functions
//---------------------------------------------------------------------------

#define labs abs

div_t div(signed int numer, signed int denom);
ldiv_t ldiv(signed long numer, signed long denom);

//---------------------------------------------------------------------------
// Multibyte character functions
//---------------------------------------------------------------------------

// Not supported

//---------------------------------------------------------------------------
// Multibyte string functions
//---------------------------------------------------------------------------

// Not supported


//---------------------------------------------------------------------------
// Internal implementation
//---------------------------------------------------------------------------

#include <stddef.h>
#include <string.h>

div_t div(signed int numer, signed int denom)
{
  div_t val;
  val.quot = numer / denom;
  val.rem = numer - (denom * val.quot);
```

```
  return (val);
}

ldiv_t ldiv(signed long numer, signed long denom)
{
  ldiv_t val;

  val.quot = numer / denom;
  val.rem = numer - (denom * val.quot);
  return (val);
}

#if defined(__PCD__)
float32 atoe(char * s)
{
  float32 pow10 = 1.0;
  float32 result = 0.0;
  unsigned int8 sign = 0;
  unsigned int8 expsign = 0;
  char c;
  unsigned int8 ptr = 0;
  unsigned int8 i;
  float32 exp = 1.0;
  unsigned int8 expcnt = 0;

  c = s[ptr++];

  if ((c>='0' && c<='9') || c=='+' || c=='-' || c=='.' || c=='E' || c=='e') {
    if(c == '-') {
      sign = 1;
      c = s[ptr++];
    }
    if(c == '+')
      c = s[ptr++];
```

```
while((c >= '0' && c <= '9')) {
   result = 10*result + c - '0';
   c = s[ptr++];
}

if (c == '.') {
   c = s[ptr++];
   while((c >= '0' && c <= '9')) {
       pow10 = pow10*10;
       result += (c - '0')/pow10;
       c = s[ptr++];
   }
}

// Handling the exponent
if (c=='e' || c=='E') {
   c = s[ptr++];

   if(c == '-') {
      expsign = 1;
      c = s[ptr++];
   }
   if(c == '+')
      c = s[ptr++];

   while((c >= '0' && c <= '9')) {
      expcnt = 10*expcnt + c - '0';
      c = s[ptr++];
   }

   for(i=0;i<expcnt;i++)
      exp*=10;

   if(expsign==1)
      result/=exp;
```

```
        else
          result*=exp;
      }
    }

    if (sign == 1)
      result = -1*result;
    return(result);
}
#endif

#if !defined(__PCD__)
float atoe(char * s)
{
    float pow10 = 1.0;
    float result = 0.0;
    unsigned int8 sign = 0;
    unsigned int8 expsign = 0;
    char c;
    unsigned int8 ptr = 0;
    unsigned int8 i;
    float exp = 1.0;
    unsigned int8 expcnt = 0;

    c = s[ptr++];

    if ((c>='0' && c<='9') || c=='+' || c=='-' || c=='.' || c=='E' || c=='e') {
      if(c == '-') {
        sign = 1;
        c = s[ptr++];
      }
      if(c == '+')
        c = s[ptr++];

      while((c >= '0' && c <= '9')) {
```

```
    result = 10*result + c - '0';
    c = s[ptr++];
  }

  if (c == '.') {
    c = s[ptr++];
    while((c >= '0' && c <= '9')) {
        pow10 = pow10*10;
        result += (c - '0')/pow10;
        c = s[ptr++];
    }
  }

  // Handling the exponent
  if (c=='e' || c=='E') {
    c = s[ptr++];

    if(c == '-') {
      expsign = 1;
      c = s[ptr++];
    }
    if(c == '+')
      c = s[ptr++];

    while((c >= '0' && c <= '9')) {
      expcnt = 10*expcnt + c - '0';
      c = s[ptr++];
    }

    for(i=0;i<expcnt;i++)
      exp*=10;

    if(expsign==1)
      result/=exp;
    else
```

```
            result*=exp;
        }
    }

    if (sign == 1)
        result = -1*result;
    return(result);
}
#endif

signed int atoi(char *s)
{
    signed int result;
    unsigned int sign, base, index;
    char c;

    index = 0;
    sign = 0;
    base = 10;
    result = 0;

    if (!s)
        return 0;
    // Omit all preceeding alpha characters
    c = s[index++];

    // increase index if either positive or negative sign is detected
    if (c == '-')
    {
        sign = 1;         // Set the sign to negative
        c = s[index++];
    }
    else if (c == '+')
    {
        c = s[index++];
```

```
}

if (c >= '0' && c <= '9')
{

  // Check for hexa number
  if (c == '0' && (s[index] == 'x' || s[index] == 'X'))
  {
    base = 16;
    index++;
    c = s[index++];
  }

  // The number is a decimal number
  if (base == 10)
  {
    while (c >= '0' && c <= '9')
    {
      result = 10*result + (c - '0');
      c = s[index++];
    }
  }
  else if (base == 16)    // The number is a hexa number
  {
    c = toupper(c);
    while ( (c >= '0' && c <= '9') || (c >= 'A' && c<='F'))
    {
      if (c >= '0' && c <= '9')
        result = (result << 4) + (c - '0');
      else
        result = (result << 4) + (c - 'A' + 10);

      c = s[index++];
      c = toupper(c);
    }
```

```
    }
  }

  if (sign == 1 && base == 10)
     result = -result;

  return(result);
}

signed long atol(char *s)
{
  signed long result;
  unsigned int sign, base, index;
  char c;

  index = 0;
  sign = 0;
  base = 10;
  result = 0;

  if (!s)
    return 0;
  c = s[index++];

  // increase index if either positive or negative sign is detected
  if (c == '-')
  {
    sign = 1;         // Set the sign to negative
    c = s[index++];
  }
  else if (c == '+')
  {
    c = s[index++];
  }
```

```
if (c >= '0' && c <= '9')
{
  if (c == '0' && (s[index] == 'x' || s[index] == 'X'))
  {
    base = 16;
    index++;
    c = s[index++];
  }

  // The number is a decimal number
  if (base == 10)
  {
    while (c >= '0' && c <= '9')
    {
      result = 10*result + (c - '0');
      c = s[index++];
    }
  }
  else if (base == 16)    // The number is a hexa number
  {
    c = toupper(c);
    while ( (c >= '0' && c <= '9') || (c >= 'A' && c <='F'))
    {
      if (c >= '0' && c <= '9')
        result = (result << 4) + (c - '0');
      else
        result = (result << 4) + (c - 'A' + 10);

      c = s[index++];c = toupper(c);
    }
  }
}

if (base == 10 && sign == 1)
  result = -result;
```

```
  return(result);
}

/* A fast routine to multiply by 10
 */
signed int32 mult_with10(int32 num)
{
  return ( (num << 1) + (num << 3) );
}

#if sizeof(long)==2
signed int32 atoi32(char *s)
{
  signed int32 result;
  int8 sign, base, index;
  char c;

  index = 0;
  sign = 0;
  base = 10;
  result = 0;

  if (!s)
    return 0;
  c = s[index++];

  // increase index if either positive or negative sign is detected
  if (c == '-')
  {
    sign = 1;        // Set the sign to negative
    c = s[index++];
  }
  else if (c == '+')
  {
```

```
    c = s[index++];
  }

  if (c >= '0' && c <= '9')
  {
    if (c == '0' && (s[index] == 'x' || s[index] == 'X'))
    {
      base = 16;
      index++;
      c = s[index++];
    }

    // The number is a decimal number
    if (base == 10)
    {
      while (c >= '0' && c <= '9') {
        result = (result << 1) + (result << 3);  // result *= 10;
        result += (c - '0');
        c = s[index++];
      }
    }
    else if (base == 16)    // The number is a hexa number
    {
      c = toupper(c);
      while ((c >= '0' && c <= '9') || (c >= 'A' && c <='F'))
      {
        if (c >= '0' && c <= '9')
          result = (result << 4) + (c - '0');
        else
          result = (result << 4) + (c - 'A' + 10);

        c = s[index++];c = toupper(c);
      }
    }
  }
```

```
  if (base == 10 && sign == 1)
    result = -result;

  return(result);
}
#endif

#if defined(__PCD__)

signed int48 atoi48(char *s)
{
  signed int48 result;
  int8 sign, base, index;
  char c;

  index = 0;
  sign = 0;
  base = 10;
  result = 0;

  if (!s)
    return 0;
  c = s[index++];

  // increase index if either positive or negative sign is detected
  if (c == '-')
  {
    sign = 1;        // Set the sign to negative
    c = s[index++];
  }
  else if (c == '+')
  {
    c = s[index++];
  }
```

```
if (c >= '0' && c <= '9')
{
  if (c == '0' && (s[index] == 'x' || s[index] == 'X'))
  {
    base = 16;
    index++;
    c = s[index++];
  }

  // The number is a decimal number
  if (base == 10)
  {
    while (c >= '0' && c <= '9') {
      result = (result << 1) + (result << 3);  // result *= 10;
      result += (c - '0');
      c = s[index++];
    }
  }
  else if (base == 16)    // The number is a hexa number
  {
    c = toupper(c);
    while ((c >= '0' && c <= '9') || (c >= 'A' && c <='F'))
    {
      if (c >= '0' && c <= '9')
        result = (result << 4) + (c - '0');
      else
        result = (result << 4) + (c - 'A' + 10);

      c = s[index++];c = toupper(c);
    }
  }
}

if (base == 10 && sign == 1)
```

```
    result = -result;

  return(result);
}

signed int64 atoi64(char *s)
{
  signed int64 result;
  int8 sign, base, index;
  char c;

  index = 0;
  sign = 0;
  base = 10;
  result = 0;

  if (!s)
    return 0;
  c = s[index++];

  // increase index if either positive or negative sign is detected
  if (c == '-')
  {
    sign = 1;        // Set the sign to negative
    c = s[index++];
  }
  else if (c == '+')
  {
    c = s[index++];
  }

  if (c >= '0' && c <= '9')
  {
    if (c == '0' && (s[index] == 'x' || s[index] == 'X'))
    {
```

```
        base = 16;
        index++;
        c = s[index++];
      }

      // The number is a decimal number
      if (base == 10)
      {
        while (c >= '0' && c <= '9') {
          result = (result << 1) + (result << 3);  // result *= 10;
          result += (c - '0');
          c = s[index++];
        }
      }
      else if (base == 16)    // The number is a hexa number
      {
        c = toupper(c);
        while ((c >= '0' && c <= '9') || (c >= 'A' && c <='F'))
        {
          if (c >= '0' && c <= '9')
            result = (result << 4) + (c - '0');
          else
            result = (result << 4) + (c - 'A' + 10);

          c = s[index++];c = toupper(c);
        }
      }
    }

    if (base == 10 && sign == 1)
      result = -result;

    return(result);
}
#endif
```

```
char * itoa(signed int32 num, unsigned int base, char * s)
{
    unsigned int32 temp=1;
    unsigned int8 i,sign=0,cnt=0;
    char c;

    if(num<0) {
       sign=1;       // Check for negative number
       num*=-1;
    }

    while(temp>0) {
       temp=(num/base);
       s[cnt]=(num%base)+'0';    // Conversion

       if(s[cnt]>0x39)
          s[cnt]+=0x7;

       cnt++;
       num=temp;
    }

    if(sign==1) {
       s[cnt]=0x2D;      // Negative sign
       cnt++;
    }

    for(i = 0;i<(int8)(cnt/2);i++) {

       c=s[i];
       s[i]=s[cnt-i-1];       // Reverse the number
       s[cnt-i-1]=c;
    }
    s[cnt]='\0';     // End the string
```

```
    return s;
}

#if defined(__PCD__)

char * itoa(signed int48 num, unsigned int base, char * s)
{
    unsigned int48 temp=1;
    unsigned int8 i,sign=0,cnt=0;
    char c;

    if(num<0) {
       sign=1;       // Check for negative number
       num*=-1;
    }

    while(temp>0) {
       temp=(num/base);
       s[cnt]=(num%base)+'0';    // Conversion

       if(s[cnt]>0x39)
          s[cnt]+=0x7;

       cnt++;
       num=temp;
    }

    if(sign==1) {
       s[cnt]=0x2D;      // Negative sign
       cnt++;
    }

    for(i = 0;i<(int8)(cnt/2);i++) {

       c=s[i];
```

```
        s[i]=s[cnt-i-1];        // Reverse the number
        s[cnt-i-1]=c;
    }
    s[cnt]='\0';     // End the string
    return s;
}

char * itoa(signed int64 num, unsigned int base, char * s)
{
    unsigned int64 temp=1;
    unsigned int8 i,sign=0,cnt=0;
    char c;

    if(num<0) {
        sign=1;        // Check for negative number
        num*=-1;
    }

    while(temp>0) {
        temp=(num/base);
        s[cnt]=(num%base)+'0';    // Conversion

        if(s[cnt]>0x39)
          s[cnt]+=0x7;

        cnt++;
        num=temp;
    }

    if(sign==1) {
        s[cnt]=0x2D;      // Negative sign
        cnt++;
    }

    for(i = 0;i<(int8)(cnt/2);i++) {
```

```
        c=s[i];
        s[i]=s[cnt-i-1];        // Reverse the number
        s[cnt-i-1]=c;
    }
    s[cnt]='\0';     // End the string
    return s;
}

#endif

float strtof(char *s, char **endptr)
{
  float pow10 = 1.0;
  float result = 0.0;
  int1 skip = 1, sign = 0, point = 0;
  char c;
  unsigned int8 ptr = 0;

  if (!s)
    return 0;

  for(c=s[ptr++]; c!=0; c=s[ptr++])
  {
    if (skip && !isspace(c))
    {
      skip = 0;
      if (c == '+')
      {
        sign = 0;
        continue;
      }
      else if (c == '-')
      {
        sign = 1;
```

```
        continue;
      }
    }
    if (!skip && (c == '.') && !point)
      point = 1;
    else if (!skip && isdigit(c))
    {
      c -= '0';
      if (point)
      {
        pow10 = pow10 * 10.0;
        result += (float)c / pow10;
      }
      else
      {
        result = 10.0 * result + (float)c;
      }
    }
    else if (!skip)
      break;
}

if (sign)
    result = -1*result;

if(endptr)
{
    if (ptr) {
      ptr--;
     #IF LONG_POINTERS
      *((int16 *)endptr)= s+ptr;
     #ELSE
      *((char *)endptr)=s+ptr;
     #ENDIF
    }
```

```
      else
      {
      #IF LONG_POINTERS
      *((int16 *)endptr)= s;
      #ELSE
      *((char *)endptr)=s;
      #ENDIF
      }
   }

   return(result);
}

#if defined(__PCD__)
float48 strtof48(char *s, char **endptr)
{
   float48 pow10 = 1.0;
   float48 result = 0.0;
   int1 skip = 1, sign = 0, point = 0;
   char c;
   unsigned int8 ptr = 0;

   if (!s)
      return 0;

   for(c=s[ptr++]; c!=0; c=s[ptr++])
   {
      if (skip && !isspace(c))
      {
         skip = 0;
         if (c == '+')
         {
            sign = 0;
            continue;
         }
```

```
      else if (c == '-')
      {
        sign = 1;
        continue;
      }
    }
    if (!skip && (c == '.') && !point)
      point = 1;
    else if (!skip && isdigit(c))
    {
      c -= '0';
      if (point)
      {
        pow10 = pow10 * 10.0;
        result += (float48)c / pow10;
      }
      else
      {
        result = 10.0 * result + (float48)c;
      }
    }
    else if (!skip)
      break;
}

if (sign)
  result = -1*result;

if(endptr)
{
  if (ptr) {
    ptr--;
   #IF LONG_POINTERS
    *((int16 *)endptr)= s+ptr;
   #ELSE
```

```
        *((char *)endptr)=s+ptr;
       #ENDIF
      }
      else
      {
      #IF LONG_POINTERS
      *((int16 *)endptr)= s;
      #ELSE
      *((char *)endptr)=s;
      #ENDIF
      }
    }

    return(result);
}

float64 strtod(char *s, char **endptr)
{
    float64 pow10 = 1.0;
    float64 result = 0.0;
    int1 skip = 1, sign = 0, point = 0;
    char c;
    unsigned int8 ptr = 0;

    if (!s)
      return 0;

    for(c=s[ptr++]; c!=0; c=s[ptr++])
    {
      if (skip && !isspace(c))
      {
        skip = 0;
        if (c == '+')
        {
          sign = 0;
```

```
        continue;
      }
      else if (c == '-')
      {
        sign = 1;
        continue;
      }
    }
    if (!skip && (c == '.') && !point)
      point = 1;
    else if (!skip && isdigit(c))
    {
      c -= '0';
      if (point)
      {
        pow10 = pow10 * 10.0;
        result += (float64)c / pow10;
      }
      else
      {
        result = 10.0 * result + (float64)c;
      }
    }
    else if (!skip)
      break;
}

if (sign)
  result = -1*result;

if(endptr)
{
  if (ptr) {
    ptr--;
   #IF LONG_POINTERS
```

```
        *((int16 *)endptr)= s+ptr;
       #ELSE
        *((char *)endptr)=s+ptr;
       #ENDIF
      }
      else
      {
      #IF LONG_POINTERS
      *((int16 *)endptr)= s;
      #ELSE
      *((char *)endptr)=s;
      #ENDIF
      }
   }

   return(result);
}
#endif

unsigned long strtoul(char *s, char *endptr, signed int base)
{
   char *sc,*s1,*sd;
   unsigned long x=0;
   char sign;
   char digits[]="0123456789abcdefghijklmnopqstuvwxyz";
   for(sc=s;isspace(*sc);++sc);
   sign=*sc=='-'||*sc=='+'?*sc++:'+';
   if(sign=='-' || base <0 || base ==1|| base >36) // invalid base
   goto StrtoulGO;

   else if (base)
   {
      if(base==16 && *sc =='0'&&(sc[1]=='x' || sc[1]=='X'))
         sc+=2;
      if(base==8 && *sc =='0')
```

```
      sc+=1;
   if(base==2 && *sc =='0'&&sc[1]=='b')
      sc+=2;

}
else if(*sc!='0') // base is 0, find base
   base=10;
else if (sc[1]=='x' || sc[1]=='X')
   base =16,sc+=2;
else if(sc[1]=='b')
   base=2,sc+=2;
else
   base=8;
for (s1=sc;*sc=='0';++sc);// skip leading zeroes
sd=memchr(digits,tolower(*sc),base);
for(; sd!=0; )
{
   x=x*base+(int16)(sd-digits);
   ++sc;
   sd=memchr(digits,tolower(*sc),base);
}
if(s1==sc)
{
StrtoulGO:
   if (endptr)
   {
      #IF LONG_POINTERS
      *((int16 *)endptr)= s;
      #ELSE
      *((char *)endptr)=s;
      #ENDIF
      }
return 0;
}
if (endptr)
```

```
  {
      #IF LONG_POINTERS
      *((int16 *)endptr)= sc;
      #ELSE
      *((char *)endptr)=sc;
      #ENDIF
  }
  return x;
}


signed long strtol(char *s,char *endptr, signed int base)
{
  char *sc,*s1,*sd;
  signed long x=0;
  char sign;
  char digits[]="0123456789abcdefghijklmnopqstuvwxyz";
  for(sc=s;isspace(*sc);++sc);
  sign=*sc=='-'||*sc=='+'?*sc++:'+';
  if (base <0 || base ==1|| base >36) // invalid base
  goto StrtolGO;
  else if (base)
  {
    if(base==16 && *sc =='0'&&(sc[1]=='x' || sc[1]=='X'))
      sc+=2;
    if(base==8 && *sc =='0')
      sc+=1;
    if(base==2 && *sc =='0'&&sc[1]=='b')
      sc+=2;

  }
  else if(*sc!='0') // base is 0, find base
    base=10;
  else if (sc[1]=='x' || sc[1]=='X')
    base =16,sc+=2;
```

```
else if(sc[1]=='b')
   base=2,sc+=2;
else
   base=8;
for (s1=sc;*sc=='0';++sc);// skip leading zeroes

sd=memchr(digits,tolower(*sc),base);
for(;sd!=0;)
{
   x=x*base+(int16)(sd-digits);
   ++sc;
   sd=memchr(digits,tolower(*sc),base);
}
if(s1==sc)
{
StrtolGO:
   if (endptr)
   {
      #IF LONG_POINTERS
      *((int16 *)endptr)= s;
      #ELSE
      *((char *)endptr)=s;
      #ENDIF
   }
return 0;
}
if(sign=='-')
   x  =-x;
if (endptr)
{
     #IF LONG_POINTERS
      *((int16 *)endptr)= sc;
     #ELSE
     *((char *)endptr)=sc;
     #ENDIF
```

```
  }
  return x;
}

signed int8 system(char *string)
{
  return 0;
}

size_t mblen(char *s,size_t n)
{
  return strlen(s);
}

int8 mbtowc(wchar_t *pwc,char *s,size_t n)
{
  *pwc=*s;
  return 1;
}

int8 wctomb(char *s,wchar_t wchar)
{
  *s=wchar;
  return 1;
}

size_t mbstowcs(wchar_t *pwcs,char *s,size_t n)
{
  strncpy(pwcs,s,n);
  return strlen(pwcs);
}

size_t wcstombs(char *s,wchar_t *pwcs,size_t n)
{
  strncpy(s,pwcs,n);
```

```
  return strlen(s);
}

//---------------------------------------------------------------------
// The random number implementation
//---------------------------------------------------------------------

unsigned int32 _Randseed;

unsigned int16 rand(void)
{
  _Randseed = _Randseed * 1103515245 + 12345;
  return ((unsigned int16)(_Randseed >> 16) % RAND_MAX);
}

void srand(unsigned int32 seed)
{
  _Randseed = seed;
}

//---------------------------------------------------------------------
// Searching and sorting utilities implementation
//---------------------------------------------------------------------

#if !defined(__PCD__)
typedef signed int8 (*_Cmpfun)(char * p1,char * p2);
#else
typedef signed int16 (*_Cmpfun)(char * p1,char * p2);
#endif



void qsort(char * qdata, unsigned int qitems, unsigned int qsize, _Cmpfun
cmp) {
  unsigned int m,j,i,l;
```

```
  int1 done;
  unsigned int8 t[16];

  m = qitems/2;
  while( m > 0 ) {
   for(j=0; j<(qitems-m); ++j) {
     i = j;
     do
     {
       done=1;
       l = i+m;
       if( (*cmp)(qdata+i*qsize, qdata+l*qsize) > 0 ) {
         memcpy(t, qdata+i*qsize, qsize);
         memcpy(qdata+i*qsize, qdata+l*qsize, qsize);
         memcpy(qdata+l*qsize, t, qsize);
         if(m <= i)
           i -= m;
           done = 0;
       }
     } while(!done);
   }
   m = m/2;
  }
}


char *bsearch(char *key, char *base, size_t num, size_t width,_Cmpfun cmp)
{
  char *p, *q;
  size_t n;
  size_t pivot;
  signed int val;

  p = base;
  n = num;
```

```
  while (n > 0)
  {
    pivot = n >> 1;
    q = p + width * pivot;

    val = (*cmp)(key, q);

    if (val < 0)
      n = pivot;
    else if (val == 0)
      return ((char *)q);
    else {
      p = q + width;
      n -= pivot + 1;
    }
  }

  return NULL;      // There's no match
}



#endif
```

LIBRERÍA math.h

```
////////////////////////////////////////////////////////////////////////////
////        (C) Copyright 1996,2011 Custom Computer Services            ////
//// This source code may only be used by licensed users of the CCS C   ////
//// compiler.  This source code may only be distributed to other       ////
//// licensed users of the CCS C compiler.  No other use, reproduction  ////
//// or distribution is permitted without written permission.           ////
//// Derivative programs created using this software in object code     ////
//// form are not restricted in any way.                                ////
////////////////////////////////////////////////////////////////////////////
////                                                                    ////
//// History:                                                           ////
```

```
////  * 9/20/2001 :  Improvments are made to sin/cos code.            ////
////               The code now is small, much faster,              ////
////               and more accurate.                               ////
////  * 2/21/2007 :  Compiler handles & operator differently and does   ////
////               not return generic (int8 *) so type cast is done   ////
////  * 6/19/2010 :  Divisions by constants converted to multiplication ////
////               by its inverse to improve computation speed        ////
////  * 1/21/2011 :  Constants used for double precision math updated   ////
////               Updated routines include 64 bit versions of cos,   ////
////               asin,atan,exp,log and dependant functions          ////
////  * 1/31/2011 :  Optimized current 32-bit float routines for cos    ////
////               and log function                                   ////
////  * 6/13/2011 :  Fixed PCD overload CEIL_FLOOR functions for values ////
////               greater then 10000000 and less then -10000000      ////
////                                                                  ////
//////////////////////////////////////////////////////////////////////////////

#ifndef MATH_H
#define MATH_H

#ifdef PI
#undef  PI
#endif
#define PI     3.1415926535897932



#define SQRT2  1.4142135623730950

//float const ps[4] = {5.9304945, 21.125224, 8.9403076, 0.29730279};
//float const qs[4] = {1.0000000, 15.035723, 17.764134, 2.4934718};

////////////////////////////// Round Functions //////////////////////////////

float32 CEIL_FLOOR(float32 x, unsigned int8 n)
{
```

```
 float32 y, res;
 unsigned int16 l;
 int1 s;

 s = 0;
 y = x;

 if (x < 0)
 {
   s = 1;
   y = -y;
 }

 if (y <= 32768.0)
   res = (float32)(unsigned int16)y;

else if (y < 10000000.0)
 {
   l = (unsigned int16)(y*0.000030517578125);
   y = 32768.0*(y*0.000030517578125 - (float32)l);
   res = 32768.0*(float32)l;
   res += (float32)(unsigned int16)y;
 }

else
 res = y;

y = y - (float32)(unsigned int16)y;

if (s)
 res = -res;

if (y != 0)
{
 if (s == 1 && n == 0)
```

```
   res -= 1.0;

  if (s == 0 && n == 1)
   res += 1.0;
 }
 if (x == 0)
    res = 0;

 return (res);
}

// Overloaded Functions to take care for new Data types in PCD
// Overloaded function CEIL_FLOOR() for data type - Float48
#if defined(__PCD__)
float48 CEIL_FLOOR(float48 x, unsigned int8 n)
{
   float48 y, res;
   unsigned int32 l;
   int1 s;

   s = 0;
   y = x;

   if (x < 0)
   {
      s = 1;
      y = -y;
   }

   if (y <= 32768.0)
      res = (float48)(unsigned int16)y;

   else if (y < 549755813888.0)
   {
      l = (unsigned int32)(y*0.000030517578125);
```

```
    y = 32768.0*(y*0.000030517578125 - (float48)l);
    res = 32768.0*(float48)l;
    res += (float48)(unsigned int16)y;
  }
  else
  {
    res = y;
    y = 0.0;
  }

  if(y != 0)
    y = y - (float48)(unsigned int16)y;

  if (s)
    res = -res;

  if (y != 0)
  {
    if (s == 1 && n == 0)
      res -= 1.0;

    if (s == 0 && n == 1)
      res += 1.0;
  }
  if (x == 0)
    res = 0;

  return (res);
}


// Overloaded function CEIL_FLOOR() for data type - Float64
float64 CEIL_FLOOR(float64 x, unsigned int8 n)
{
  float64 y, res;
```

```
int64 l;
int1 s;

s = 0;
y = x;

if (x < 0)
{
  s = 1;
  y = -y;
}

if (y <= 32768.0)
  res = (float64)(unsigned int16)y;
else if (y < 4503599627370496.0)
{
  l = (int64)(y*0.000030517578125);
  y = 32768.0*(y*0.000030517578125 - (float64)l);
  res = 32768.0*(float64)l;
  res += (float64)(unsigned int16)y;
}
else
{
  res = y;
  y = 0.0;
}

if(y != 0)
  y = y - (float64)(unsigned int16)y;

if (s)
  res = -res;

if (y != 0)
{
```

```
    if (s == 1 && n == 0)
      res -= 1.0;

    if (s == 0 && n == 1)
      res += 1.0;
  }

  if (x == 0)
    res = 0;

  return (res);
}
#endif

////////////////////////////////////////////////////////////////////////////
//   float floor(float x)
////////////////////////////////////////////////////////////////////////////
// Description : rounds down the number x.
// Date : N/A
//
float32 floor(float32 x)
{
  return CEIL_FLOOR(x,0);
}
// Following 2 functions are overloaded functions of floor() for PCD
// Overloaded function floor() for data type - Float48
#if defined(__PCD__)
float48 floor(float48 x)
{
  return CEIL_FLOOR(x, 0);
}

// Overloaded function floor() for data type - Float64
float64 floor(float64 x)
{
```

```
  return CEIL_FLOOR(x, 0);
}
#endif


////////////////////////////////////////////////////////////////////////////
//   float ceil(float x)
////////////////////////////////////////////////////////////////////////////
// Description : rounds up the number x.
// Date : N/A
//
float32 ceil(float32 x)
{
  return CEIL_FLOOR(x, 1);
}
// Following 2 functions are overloaded functions of ceil() for PCD
// Overloaded function ceil() for data type - Float48
#if defined(__PCD__)
float48 ceil(float48 x)
{
  return CEIL_FLOOR(x, 1);
}

// Overloaded function ceil() for data type - Float64
float64 ceil(float64 x)
{
  return CEIL_FLOOR(x, 1);
}
#endif

 ////////////////////////////////////////////////////////////////////////////
//   float fabs(float x)
////////////////////////////////////////////////////////////////////////////
// Description : Computes the absolute value of floating point number x
// Returns : returns the absolute value of x
```

```
// Date : N/A
//
#define fabs abs

//////////////////////////////////////////////////////////////////////////////
//   float fmod(float x)
//////////////////////////////////////////////////////////////////////////////
// Description : Computes the floating point remainder of x/y
// Returns : returns the value of x= i*y, for some integer i such that, if y
// is non zero, the result has the same isgn of x na dmagnitude less than the
// magnitude of y. If y is zero then a domain error occurs.
// Date : N/A
//

float fmod(float32 x,float32 y)
{
  float32 i;
  if (y!=0.0)
  {
    i=(x/y < 0.0)? ceil(x/y): floor(x/y);
    return(x-(i*y));
  }
  else
  {
  #ifdef _ERRNO
  {
    errno=EDOM;
  }
  #endif
  }
}
//Overloaded function for fmod() for PCD
// Overloaded function fmod() for data type - Float48
#if defined(__PCD__)
float48 fmod(float48 x,float48 y)
```

```
{
  float48 i;
  if (y!=0.0)
  {
    i=(x/y < 0.0)? ceil(x/y): floor(x/y);
    return(x-(i*y));
  }
  else
  {
  #ifdef _ERRNO
  {
    errno=EDOM;
  }
  #endif
  }
}
// Overloaded function fmod() for data type - Float64
float64 fmod(float64 x,float64 y)
{
  float64 i;
  if (y!=0.0)
  {
    i=(x/y < 0.0)? ceil(x/y): floor(x/y);
    return(x-(i*y));
  }
  else
  {
  #ifdef _ERRNO
  {
    errno=EDOM;
  }
  #endif
  }
}
#endif
```

```
//////////////////// Exponential and logarithmic functions ////////////////////
////////////////////////////////////////////////////////////////////////////
//   float exp(float x)
////////////////////////////////////////////////////////////////////////////
// Description : returns the value (e^x)
// Date : N/A
//
#define LN2 0.6931471805599453
#define LN2_INV 1.4426950408889634073

float const pe[6] = {0.000207455774, 0.00127100575, 0.00965065093,
                     0.0554965651,  0.240227138,  0.693147172};



float32 exp(float32 x)
{
  float32 y, res, r;
  #if defined(__PCD__)
  int8 data1;
  #endif
  signed int8 n;
  int1 s;
  #ifdef _ERRNO
  if(x > 88.722838)
  {
    errno=ERANGE;
    return(0);
  }
  #endif
  n = (signed int16)(x*LN2_INV);
  s = 0;
  y = x;

  if (x < 0)
  {
```

```
    s = 1;
    n = -n;
    y = -y;
  }

  res = 0.0;
#if !defined(__PCD__)
  *((unsigned int8 *)(&res)) = n + 0x7F;
#endif

#if defined(__PCD__)  // Takes care of IEEE format for PCD
  data1 = n+0x7F;
  if(bit_test(data1,0))
  bit_set(*(((unsigned int8 *)(&res)+2)),7);
  rotate_right(&data1,1);
  bit_clear(data1,7);
  *(((unsigned int8 *)(&res)+3)) = data1;
#endif

  y = y*LN2_INV - (float32)n;

  r = pe[0]*y + pe[1];
  r = r*y + pe[2];
  r = r*y + pe[3];
  r = r*y + pe[4];
  r = r*y + pe[5];

  res = res*(1.0 + y*r);

  if (s)
    res = 1.0/res;
  return(res);
}
```

```
//Overloaded function for exp() for PCD
// Overloaded function exp() for data type - Float48
#if defined(__PCD__)
float48 exp(float48 x)
{
  float48 y, res, r;
  int8 data1;
  signed int8 n;
  int1 s;
  #ifdef _ERRNO
  if(x > 88.722838)
  {
    errno=ERANGE;
    return(0);
  }
  #endif
  n = (signed int16)(x*LN2_INV);
  s = 0;
  y = x;

  if (x < 0)
  {
    s = 1;
    n = -n;
    y = -y;
  }

  res = 0.0;

  data1 = n+0x7F;
  if(bit_test(data1,0))
  bit_set(*(((unsigned int8 *)(&res)+4)),7);
  rotate_right(&data1,1);
  bit_clear(data1,7);
  *(((unsigned int8 *)(&res)+5)) = data1;
```

```
  y = y*LN2_INV - (float48)n;

  r = pe[0]*y + pe[1];
  r = r*y + pe[2];
  r = r*y + pe[3];
  r = r*y + pe[4];
  r = r*y + pe[5];

  res = res*(1.0 + y*r);

  if (s)
    res = 1.0/res;
  return(res);
}

// Overloaded function exp() for data type - Float64
float64 const pe_64[12] ={9.30741400474913e-011,-4.28655416283316e-
011,
                    8.71486547014137e-009,9.84458531538385e-008,
                    1.32588296983536e-006,1.52489283823016e-005,
                    0.000154037598423921,0.00133335487036216,
                    0.00961812936407326,0.0555041086222122,
                    0.240226506962827,0.693147180559823};

float64 exp(float64 x)
{
  float64 y, res, r;
  unsigned int16 data1, data2;
  unsigned int16 *p;
  signed int16 n;
  int1 s;
  #ifdef _ERRNO
  if(x > 709.7827128)
  {
```

```
    errno=ERANGE;
    return(0);
  }
  #endif
  y = x*LN2_INV;
  n = (signed int16)y;
  s = 0;
  y = x;

  if (x < 0)
  {
    s = 1;
    n = -n;
    y = -y;
  }

  res = 0.0;

#if !defined(__PCD__)
  *((unsigned int16 *)(&res)) = n + 0x7F;
#endif
  p= (((unsigned int16 *)(&res))+3);
  data1 = *p;
  data2 = *p;
  data1 = n + 0x3FF;
  data1 = data1 <<4;
  if(bit_test(data2,15))
  bit_set(data1,15);
  data2 = data2 & 0x000F;
  data1 ^= data2;

  *(((unsigned int16 *)(&res)+3)) = data1;


  y = y*LN2_INV - (float64)n;
```

```
   r = pe_64[0]*y + pe_64[1];
   r = r*y + pe_64[2];
   r = r*y + pe_64[3];
   r = r*y + pe_64[4];
   r = r*y + pe_64[5];
   r = r*y + pe_64[6];
   r = r*y + pe_64[7];
   r = r*y + pe_64[8];
   r = r*y + pe_64[9];
   r = r*y + pe_64[10];
   r = r*y + pe_64[11];

   res = res*(1.0 + y*r);

   if (s)
      res = 1.0/res;
   return(res);
}
#ENDIF


/*************************************************************/

float32 const pl[4] = {-1.080310025160955, 1.999999947089217};
float32 const ql[4] = {0.091284365719509, -0.873491916557671};

//////////////////////////////////////////////////////////////////////////////
//   float log(float x)
//////////////////////////////////////////////////////////////////////////////
// Description : returns the the natural log of x
// Date : N/A
//
float32 log(float32 x)
{
```

```
  float32 y, res, r, y2;
  #if defined(__PCD__)
  unsigned int8  data1,data2;
  #endif
  signed int8 n;
  #ifdef _ERRNO
  if(x <0)
  {
    errno=EDOM;
  }
  if(x ==0)
  {
    errno=ERANGE;
    return(0);
  }
  #endif
  y = x;

  if (y != 1.0)
  {
#if !defined(__PCD__)
   *((unsigned int8 *)(&y)) = 0x7E;
#endif

#if defined(__PCD__) // Takes care of IEEE format
  data2 = *(((unsigned int8 *)(&y))+3);
  *(((unsigned int8 *)(&y))+3) = 0x3F;
  data1 = *(((unsigned int8 *)(&y))+2);
  bit_clear(data1,7);
  *(((unsigned int8 *)(&y))+2) = data1;
  if(bit_test(data2,7))
    bit_set(*(((unsigned int8 *)(&y))+3),7);
#endif

    y = (y - 1.0)/(y + 1.0);
```

```
      y2=y*y;

      res = pl[0]*y2 + pl[1];

      r = ql[0]*y2 + ql[1];
      r = r*y2 + 1.0;

      res = y*res/r;
#if !defined(__PCD__)
      n = *((unsigned int8 *)(&x)) - 0x7E;
#endif
#if defined(__PCD__)
    data1 = *(((unsigned int8 *)(&x)+3));
    rotate_left(&data1,1);
    data2 = *(((unsigned int8 *)(&x)+2));
    if(bit_test (data2,7))
      bit_set(data1,0);
    n = data1 - 0x7E;
#endif

      if (n<0)
         r = -(float32)-n;
      else
         r = (float32)n;

      res += r*LN2;
   }

   else
      res = 0.0;

   return(res);
}
```

```
//Overloaded function for log() for PCD
// Overloaded function log() for data type - Float48
#if defined(__PCD__)
// Overloaded function log() for data type - float64
float64 const pl_64[4] = {-0.145694052150681, 1.365485300000370,
                    -3.120535668274329, 2.000000000000000};

float64 const ql_64[4] = {0.006047500465908, -0.208298281937234,
                    1.113943039156721, -1.893601167470470};

float48 log(float48 x)
{
  float48 y, res, r, y2;
  unsigned int8  data1,data2;
  signed int8 n;
#ifdef _ERRNO
  if(x <0)
  {
    errno=EDOM;
  }
  if(x ==0)
  {
    errno=ERANGE;
    return(0);
  }
#endif
  y = x;

  if (y != 1.0)
  {

#if !defined(__PCD__)
    *((unsigned int8 *)(&y)) = 0x7E;
#endif
    data2 = *(((unsigned int8 *)(&y))+5);
```

```
*(((unsigned int8 *)(&y))+5) = 0x3F;
data1 = *(((unsigned int8 *)(&y))+4);
bit_clear(data1,7);
*(((unsigned int8 *)(&y))+4) = data1;

if(bit_test(data2,7))
bit_set(*(((unsigned int8 *)(&y))+4),7);
y = (y - 1.0)/(y + 1.0);

y2=y*y;

res = pl_64[0]*y2 + pl_64[1];
res = res*y2 + pl_64[2];
res = res*y2 + pl_64[3];

r = ql_64[0]*y2 + ql_64[1];
r = r*y2 + ql_64[2];
r = r*y2 + ql_64[3];
r = r*y2 + 1.0;
res = y*res/r;

data1 = *(((unsigned int8 *)(&x)+5));
rotate_left(&data1,1);
data2 = *(((unsigned int8 *)(&x)+4));
if(bit_test (data2,7))
bit_set(data1,0);

n = data1 - 0x7E;

if (n<0)
r = -(float48)-n;
else
r = (float48)n;

res += r*LN2;
```

```
  }

  else
  res = 0.0;

  return(res);
}

float64 log(float64 x)
{
  float64 y, res, r, y2;
  unsigned int16  data1,data2;
  unsigned int16 *p;
  signed int16 n;
  #ifdef _ERRNO
  if(x <0)
  {
    errno=EDOM;
  }
  if(x ==0)
  {
    errno=ERANGE;
    return(0);
  }
  #endif
  y = x;

  if (y != 1.0)
  {
  #if !defined(__PCD__)
     *((unsigned int8 *)(&y)) = 0x7E;
  #endif
    p= (((unsigned int16 *)(&y))+3);
    data1 = *p;
    data2 = *p;
```

```
data1 = 0x3FE;
data1 = data1 <<4;
if(bit_test (data2,15))
   bit_set(data1,15);
data2 = data2 & 0x000F;
data1 ^=data2;

*p = data1;

y = (y - 1.0)/(y + 1.0);

y2=y*y;

res = pl_64[0]*y2 + pl_64[1];
res = res*y2 + pl_64[2];
res = res*y2 + pl_64[3];

r = ql_64[0]*y2 + ql_64[1];
r = r*y2 + ql_64[2];
r = r*y2 + ql_64[3];
r = r*y2 + 1.0;

res = y*res/r;

p= (((unsigned int16 *)(&x))+3);
data1 = *p;
bit_clear(data1,15);
data1 = data1 >>4;
n = data1 - 0x3FE;


if (n<0)
   r = -(float64)-n;
else
   r = (float64)n;
```

```
    res += r*LN2;
  }

  else
    res = 0.0;

  return(res);
}
#endif


#define LN10 2.3025850929940456
#define LN10_INV 0.4342944819032518276

////////////////////////////////////////////////////////////////////////////
//   float log10(float x)
////////////////////////////////////////////////////////////////////////////
// Description : returns the the log base 10 of x
// Date : N/A
//
float32 log10(float32 x)
{
  float32 r;

  r = log(x);
  r = r*LN10_INV;
  return(r);
}

//Overloaded functions for log10() for PCD
// Overloaded function log10() for data type - Float48
#if defined(__PCD__)
float48 log10(float48 x)
{
```

```
  float48 r;

  r = log(x);
  r = r*LN10_INV;
  return(r);
}

// Overloaded function log10() for data type - Float64
float64 log10(float64 x)
{
  float64 r;

  r = log(x);
  r = r*LN10_INV;
  return(r);
}
#endif
////////////////////////////////////////////////////////////////////////////
//   float modf(float x)
////////////////////////////////////////////////////////////////////////////
// Description :breaks the argument value int integral and fractional parts,
// ach of which have the same sign as the argument.  It stores the integral
part
// as a float in the object pointed to by the iptr
// Returns : returns the signed fractional part of value.
// Date : N/A
//

float32 modf(float32 value,float32 *iptr)
{
  *iptr=(value < 0.0)? ceil(value): floor(value);
  return(value - *iptr);
}
//Overloaded functions for modf() for PCD
// Overloaded function modf() for data type - Float48
```

```
#if defined(__PCD__)
float48 modf(float48 value,float48 *iptr)
{
  *iptr=(value < 0.0)? ceil(value): floor(value);
  return(value - *iptr);
}
// Overloaded function modf() for data type - Float64
float64 modf(float64 value,float64 *iptr)
{
  *iptr=(value < 0.0)? ceil(value): floor(value);
  return(value - *iptr);
}
#endif

////////////////////////////////////////////////////////////////////////////
//   float pwr(float x,float y)
////////////////////////////////////////////////////////////////////////////
// Description : returns the value (x^y)
// Date : N/A
// Note : 0 is returned when the function will generate an imaginary number
//
float32 pwr(float32 x,float32 y)
{
  if(0 > x && fmod(y, 1) == 0) {
    if(fmod(y, 2) == 0) {
      return (exp(log(-x) * y));
    } else {
      return (-exp(log(-x) * y));
    }
  } else if(0 > x && fmod(y, 1) != 0) {
    return 0;
  } else {
    if(x != 0 || 0 >= y) {
      return (exp(log(x) * y));
    }
```

```
  }
}
//Overloaded functions for pwr() for PCD
// Overloaded function pwr() for data type - Float48
#if defined(__PCD__)
float48 pwr(float48 x,float48 y)
{
  if(0 > x && fmod(y, 1) == 0) {
    if(fmod(y, 2) == 0) {
      return (exp(log(-x) * y));
    } else {
      return (-exp(log(-x) * y));
    }
  } else if(0 > x && fmod(y, 1) != 0) {
    return 0;
  } else {
    if(x != 0 || 0 >= y) {
      return (exp(log(x) * y));
    }
  }
}
// Overloaded function pwr() for data type - Float64
float64 pwr(float64 x,float64 y)
{
  if(0 > x && fmod(y, 1) == 0) {
    if(fmod(y, 2) == 0) {
      return (exp(log(-x) * y));
    } else {
      return (-exp(log(-x) * y));
    }
  } else if(0 > x && fmod(y, 1) != 0) {
    return 0;
  } else {
    if(x != 0 || 0 >= y) {
      return (exp(log(x) * y));
```

```
    }
  }
}
#endif

//////////////////// Power functions ////////////////////

////////////////////////////////////////////////////////////////////////////
//   float pow(float x,float y)
////////////////////////////////////////////////////////////////////////////
// Description : returns the value (x^y)
// Date : N/A
// Note : 0 is returned when the function will generate an imaginary number
//
float32 pow(float32 x,float32 y)
{
  if(0 > x && fmod(y, 1) == 0) {
    if(fmod(y, 2) == 0) {
      return (exp(log(-x) * y));
    } else {
      return (-exp(log(-x) * y));
    }
  } else if(0 > x && fmod(y, 1) != 0) {
    return 0;
  } else {
    if(x != 0 || 0 >= y) {
      return (exp(log(x) * y));
    }
  }
}
//Overloaded functions for pow() for PCD
// Overloaded function for pow() data type - Float48
#if defined(__PCD__)
float48 pow(float48 x,float48 y)
{
```

```
  if(0 > x && fmod(y, 1) == 0) {
    if(fmod(y, 2) == 0) {
      return (exp(log(-x) * y));
    } else {
      return (-exp(log(-x) * y));
    }
  } else if(0 > x && fmod(y, 1) != 0) {
    return 0;
  } else {
    if(x != 0 || 0 >= y) {
      return (exp(log(x) * y));
    }
  }
}

// Overloaded function pow() for data type - Float64
float64 pow(float64 x,float64 y)
{
  if(0 > x && fmod(y, 1) == 0) {
    if(fmod(y, 2) == 0) {
      return (exp(log(-x) * y));
    } else {
      return (-exp(log(-x) * y));
    }
  } else if(0 > x && fmod(y, 1) != 0) {
    return 0;
  } else {
    if(x != 0 || 0 >= y) {
      return (exp(log(x) * y));
    }
  }
}
#endif

////////////////////////////////////////////////////////////////////////////////
```

```
//   float sqrt(float x)
////////////////////////////////////////////////////////////////////////////
// Description : returns the square root of x
// Date : N/A
//
float32 sqrt(float32 x)
{
  float32 y, res;
  #if defined(__PCD__)
  unsigned int16 data1,data2;
  #endif
  unsigned int8 *p;

  #ifdef _ERRNO
  if(x < 0)
  {
    errno=EDOM;
  }
  #endif

  if( x<=0.0)
    return(0.0);

  y=x;

  #if !defined(__PCD__)
   p=&y;
  (*p)=(unsigned int8)((((unsigned int16)(*p)) + 127) >> 1);
  #endif

  #if defined(__PCD__)
   p = (((unsigned int8 *)(&y))+3);
   data1 = *(((unsigned int8 *)(&y))+3);
   data2 = *(((unsigned int8 *)(&y))+2);
   rotate_left(&data1,1);
```

```
  if(bit_test(data2,7))
     bit_set(data1,0);
  data1 = ((data1+127) >>1);
  bit_clear(data2,7);
  if(bit_test(data1,0))
     bit_set(data2,7);
  data1 = data1 >>1;
  *(((unsigned int8 *)(&y))+3) = data1;
  *(((unsigned int8 *)(&y))+2) = data2;

#endif

 do {
    res=y;
    y+=(x/y);

    #if !defined(__PCD__)
   (*p)--;
 #endif

 #if defined(__PCD__)
  data1 = *(((unsigned int8 *)(&y))+3);
  data2 = *(((unsigned int8 *)(&y))+2);
  rotate_left(&data1,1);
  if(bit_test(data2,7))
     bit_set(data1,0);
  data1--;
  bit_clear(data2,7);
  if(bit_test(data1,0))
     bit_set(data2,7);
  data1 = data1 >>1;
  *(((unsigned int8 *)(&y))+3) = data1;
  *(((unsigned int8 *)(&y))+2) = data2;

#endif
```

```
  } while(res != y);

  return(res);
}
//Overloaded functions for sqrt() for PCD
// Overloaded function sqrt() for data type - Float48
#if defined(__PCD__)
float48 sqrt(float48 x)
{
  float48 y, res;
  unsigned int16 data1,data2;
  unsigned int8 *p;

  #ifdef _ERRNO
  if(x < 0)
  {
    errno=EDOM;
  }
  #endif

  if( x<=0.0)
    return(0.0);

  y=x;

  #if !defined(__PCD__)
   p=&y;
  (*p)=(unsigned int8)((((unsigned int16)(*p)) + 127) >> 1);
  #endif

  #if defined(__PCD__)
   p = (((unsigned int8 *)(&y))+5);
   data1 = *(((unsigned int8 *)(&y))+5);
   data2 = *(((unsigned int8 *)(&y))+4);
   rotate_left(&data1,1);
```

```
 if(bit_test(data2,7))
    bit_set(data1,0);
 data1 = ((data1+127) >>1);
 bit_clear(data2,7);
 if(bit_test(data1,0))
    bit_set(data2,7);
 data1 = data1 >>1;
 *(((unsigned int8 *)(&y))+5) = data1;
 *(((unsigned int8 *)(&y))+4) = data2;

#endif

 do {
   res=y;
   y+=(x/y);

   #if !defined(__PCD__)
  (*p)--;
 #endif

 data1 = *(((unsigned int8 *)(&y))+5);
 data2 = *(((unsigned int8 *)(&y))+4);
 rotate_left(&data1,1);
 if(bit_test(data2,7))
    bit_set(data1,0);
 data1--;
 bit_clear(data2,7);
 if(bit_test(data1,0))
    bit_set(data2,7);
 data1 = data1 >>1;
 *(((unsigned int8 *)(&y))+5) = data1;
 *(((unsigned int8 *)(&y))+4) = data2;

 } while(res != y);
```

```
  return(res);
}

// Overloaded function sqrt() for data type - Float64
float64 sqrt(float64 x)
{
  float64 y, res;
  unsigned int16 *p;
  unsigned int16 temp1,temp2;

  #ifdef _ERRNO
  if(x < 0)
  {
    errno=EDOM;
  }
  #endif

  if( x<=0.0)
    return(0.0);

  y=x;
  p= (((unsigned int16 *)(&y))+3);
  temp1 = *p;
  temp2 = *p;
  bit_clear(temp1,15);
  temp1 = (temp1>>4)+1023;
  temp1 = temp1 >> 1;
  temp1 = (temp1<<4) & 0xFFF0;
  if(bit_test(temp2,15))
    bit_set(temp1,15);
  temp2 = temp2 & 0x000F;
  temp1 ^= temp2;

  (*p) = temp1;
```

```
  do {
     res=y;
     y+=(x/y);
    temp1 = *p;
    temp2 = *p;
    bit_clear(temp1,15);
    temp1 = (temp1>>4);
    temp1--;
    temp1 = (temp1<<4) & 0xFFF0;
    if(bit_test(temp2,15))
       bit_set(temp1,15);
    temp2 = temp2 & 0x000F;
    temp1 ^= temp2;
    (*p) = temp1;

  } while(res != y);

  return(res);
}
#endif

////////////////////////////// Trig Functions //////////////////////////////
#ifdef PI_DIV_BY_TWO_INV
#undef PI_DIV_BY_TWO_INV
#endif
#define PI_DIV_BY_TWO_INV 0.636619772367581 34
#ifdef PI_DIV_BY_TWO
#undef PI_DIV_BY_TWO
#endif
#define PI_DIV_BY_TWO   1.5707963267948966
#ifdef TWOBYPI
#undef TWOBYPI
#endif
#define TWOBYPI         0.6366197723675813
```

```
////////////////////////////////////////////////////////////////////////////
//   float cos(float x)
////////////////////////////////////////////////////////////////////////////
// Description : returns the cosine value of the angle x, which is in radian
// Date : 9/20/2001
//
float32 cos(float32 x)
{
  float32 y, t, t2 = 1.0;
  unsigned int8 quad, i;
  float32 frac;
  float32 p[5] = {                  //by the series definition for cosine
    -0.49999999456337096,           // sum ( ( (-1)^n * x^2n )/(2n)! )
     0.04166663896921267,
    -0.00138883894522527,
     0.00002476138231734,
    -0.00000026070414770
    //-0.00000000001147,
    // 0.00000000000005
  };

  if (x < 0) x = -x;                // absolute value of input

  quad = (unsigned int8)(x * PI_DIV_BY_TWO_INV);    // quadrant
  frac = (x * PI_DIV_BY_TWO_INV) - quad;  // fractional part of input
  quad = quad % 4;                  // quadrant (0 to 3)

  if (quad == 0 || quad == 2)
    t = frac * PI_DIV_BY_TWO;
  else if (quad == 1)
    t = (1-frac) * PI_DIV_BY_TWO;
  else // should be 3
    t = (frac-1) * PI_DIV_BY_TWO;

  y = 1.0;
```

```
  t = t * t;
  for (i = 0; i <= 4; i++)
  {
    t2 = t2 * t;
    y = y + p[i] * t2;
  }

  if (quad == 2 || quad == 1)
    y = -y;  // correct sign

  return (y);
}


//Overloaded functions for cos() for PCD
// Overloaded function cos() for data type - Float48
#if defined(__PCD__)
float48 cos(float48 x)
{
  float48 y, t, t2 = 1.0;
  unsigned int8 quad, i;
  float48 frac;
  float48 p[6] = {                  //by the series definition for cosine
    -0.5,                         // sum ( ( (-1)^n * x^2n )/(2n)! )
     0.04166666666667,
    -0.00138888888889,
     0.00002480158730,
    -0.00000027557319,
     0.00000000208767
  };

  if (x < 0) x = -x;                // absolute value of input

  quad = (unsigned int8)(x * PI_DIV_BY_TWO_INV);    // quadrant
  frac = (x * PI_DIV_BY_TWO_INV) - quad;  // fractional part of input
```

```
  quad = quad % 4;                  // quadrant (0 to 3)

  if (quad == 0 || quad == 2)
    t = frac * PI_DIV_BY_TWO;
  else if (quad == 1)
    t = (1-frac) * PI_DIV_BY_TWO;
  else // should be 3
    t = (frac-1) * PI_DIV_BY_TWO;

  y = 0.999999999781;
  t = t * t;
  for (i = 0; i <= 5; i++)
  {
    t2 = t2 * t;
    y = y + p[i] * t2;
  }

  if (quad == 2 || quad == 1)
    y = -y;  // correct sign

  return (y);
}

// Overloaded function cos() for data type - Float64
float64 cos(float64 x)
{
  float64 y, t, t2 = 1.0;
  unsigned int8 quad, i;
  float64 frac;
  float64 p_64[9] = {             //by the series definition for cosine
    -0.49999999999998740,         // sum ( ( (-1)^n * x^2n )/(2n)! )
     0.04166666666656518,
    -0.00138888888851691,
     0.00002480158658490,
    -0.00000027557239796,
```

```
    0.00000000208715031,
   -0.00000000001126577,
    0.00000000000000427,
    0.00000000000000372};

  if (x < 0) x = -x;                // absolute value of input

  quad = (unsigned int8)(x * PI_DIV_BY_TWO_INV);    // quadrant
  frac = (x * PI_DIV_BY_TWO_INV) - quad;  // fractional part of input
  quad = quad % 4;                  // quadrant (0 to 3)

  if (quad == 0 || quad == 2)
    t = frac * PI_DIV_BY_TWO;
  else if (quad == 1)
    t = (1-frac) * PI_DIV_BY_TWO;
  else // should be 3
    t = (frac-1) * PI_DIV_BY_TWO;

  y = 1.0;
  t = t * t;
  for (i = 0; i <= 8; i++)
  {
    t2 = t2 * t;
    y = y + p_64[i] * t2;
  }

  if (quad == 2 || quad == 1)
    y = -y;  // correct sign

  return (y);
}

#endif

////////////////////////////////////////////////////////////////////////////////
```

```
//   float sin(float x)
////////////////////////////////////////////////////////////////////////////
// Description : returns the sine value of the angle x, which is in radian
// Date : 9/20/2001
//
float32 sin(float32 x)
{
  return cos(x - PI_DIV_BY_TWO);
}

//Overloaded functions for sin() for PCD
// Overloaded function sin() for data type - Float48
#if defined(__PCD__)
float48 sin(float48 x)
{
  return cos(x - PI_DIV_BY_TWO);
}

// Overloaded function sin() for data type - Float48
float64 sin(float64 x)
{
  return cos(x - PI_DIV_BY_TWO);
}
#endif

////////////////////////////////////////////////////////////////////////////
//   float tan(float x)
////////////////////////////////////////////////////////////////////////////
// Description : returns the tangent value of the angle x, which is in radian
// Date : 9/20/2001
//
float32 tan(float32 x)
{
  float32 c, s;
```

```
  c = cos(x);
  if (c == 0.0)
    return (1.0e+36);

  s = sin(x);
  return(s/c);
}
//Overloaded functions for tan() for PCD
// Overloaded function tan() for data type - Float48
#if defined(__PCD__)
float48 tan(float48 x)
{
  float48 c, s;

  c = cos(x);
  if (c == 0.0)
    return (1.0e+36);

  s = sin(x);
  return(s/c);
}

// Overloaded function tan() for data type - Float48
float64 tan(float64 x)
{
  float64 c, s;

  c = cos(x);
  if (c == 0.0)
    return (1.0e+36);

  s = sin(x);
  return(s/c);
}
#endif
```

```
float32 const pas[3] = {0.49559947, -4.6145309, 5.6036290};
float32 const qas[3] = {1.0000000,  -5.5484666, 5.6036290};

float32 ASIN_COS(float32 x, unsigned int8 n)
{
  float32 y, res, r, y2;
  int1 s;
  #ifdef _ERRNO
  if(x <-1 || x > 1)
  {
    errno=EDOM;
  }
  #endif
  s = 0;
  y = x;

  if (x < 0)
  {
    s = 1;
    y = -y;
  }

  if (y > 0.5)
  {
    y = sqrt((1.0 - y)*0.5);
    n += 2;
  }

  y2=y*y;

  res = pas[0]*y2 + pas[1];
  res = res*y2 + pas[2];

  r = qas[0]*y2 + qas[1];
```

```
  r = r*y2 + qas[2];

  res = y*res/r;

  if (n & 2)     // |x| > 0.5
    res = PI_DIV_BY_TWO - 2.0*res;
  if (s)
    res = -res;
  if (n & 1)          // take arccos
    res = PI_DIV_BY_TWO - res;

  return(res);
}


//Overloaded functions for ASIN_COS() for PCD
// Overloaded function ASIN_COS() for data type - Float48
#if defined(__PCD__)
float48 ASIN_COS(float48 x, unsigned int8 n)
{
  float48 y, res, r, y2;
  int1 s;
  #ifdef _ERRNO
  if(x <-1 || x > 1)
  {
    errno=EDOM;
  }
  #endif
  s = 0;
  y = x;

  if (x < 0)
  {
    s = 1;
    y = -y;
```

```
  }

  if (y > 0.5)
  {
    y = sqrt((1.0 - y)*0.5);
    n += 2;
  }

  y2=y*y;

  res = pas[0]*y2 + pas[1];
  res = res*y2 + pas[2];

  r = qas[0]*y2 + qas[1];
  r = r*y2 + qas[2];

  res = y*res/r;

  if (n & 2)     // |x| > 0.5
    res = PI_DIV_BY_TWO - 2.0*res;
  if (s)
    res = -res;
  if (n & 1)          // take arccos
    res = PI_DIV_BY_TWO - res;

  return(res);
}

// Overloaded function ASIN_COS() for data type - Float64
double pas_64[5]={1.000000000000000,-
2.069938587724477,1.421444580548907,
                -0.360690137621842,0.024929451660228};

double qas_64[5]={-2.236605254391134,1.719212122946451,-
0.524122954476133,
```

```
                0.052808872045749,-0.000662536036570};

float64 ASIN_COS(float64 x, unsigned int8 n)
{
  float64 y, res, r, y2;
  int1 s;
  #ifdef _ERRNO
  if(x <-1 || x > 1)
  {
    errno=EDOM;
  }
  #endif
  s = 0;
  y = x;

  if (x < 0)
  {
    s = 1;
    y = -y;
  }

  if (y > 0.5)
  {
    y = sqrt((1.0 - y)*0.5);
    n += 2;
  }

  y2=y*y;

  res = pas_64[4]*y2 + pas_64[3];
  res = res*y2 + pas_64[2];
  res = res*y2 + pas_64[1];
  res = res*y2 + pas_64[0];

  r = qas_64[4]*y2 + qas_64[3];
```

```
  r = r*y2 + qas_64[2];
  r = r*y2 + qas_64[1];
  r = r*y2 + qas_64[0];
  r = r*y2 + 1.0;

  res = y*res/r;

  if (n & 2)     // |x| > 0.5
    res = PI_DIV_BY_TWO - 2.0*res;
  if (s)
    res = -res;
  if (n & 1)          // take arccos
    res = PI_DIV_BY_TWO - res;

  return(res);
}
#endif

////////////////////////////////////////////////////////////////////////////////
//   float asin(float x)
////////////////////////////////////////////////////////////////////////////////
// Description : returns the arcsine value of the value x.
// Date : N/A
//
float32 asin(float32 x)
{
  float32 r;

  r = ASIN_COS(x, 0);
  return(r);
}
//Overloaded functions for asin() for PCD
// Overloaded function asin() for data type - Float48
#if defined(__PCD__)
float48 asin(float48 x)
```

```
{
  float48 r;

  r = ASIN_COS(x, 0);
  return(r);
}

// Overloaded function asin() for data type - Float64
float64 asin(float64 x)
{
  float64 r;

  r = ASIN_COS(x, 0);
  return(r);
}
#endif

////////////////////////////////////////////////////////////////////////////
//   float acos(float x)
////////////////////////////////////////////////////////////////////////////
// Description : returns the arccosine value of the value x.
// Date : N/A
//
float32 acos(float32 x)
{
  float32 r;

  r = ASIN_COS(x, 1);
  return(r);
}
//Overloaded functions for acos() for PCD
// Overloaded function acos() for data type - Float48
#if defined(__PCD__)
float48 acos(float48 x)
{
```

```
  float48 r;

  r = ASIN_COS(x, 1);
  return(r);
}

// Overloaded function acos() for data type - Float64
float64 acos(float64 x)
{
  float64 r;

  r = ASIN_COS(x, 1);
  return(r);
}
#endif

float32 const pat[4] = {0.17630401, 5.6710795, 22.376096, 19.818457};
float32 const qat[4] = {1.0000000,  11.368190, 28.982246, 19.818457};

////////////////////////////////////////////////////////////////////////////
//   float atan(float x)
////////////////////////////////////////////////////////////////////////////
// Description : returns the arctangent value of the value x.
// Date : N/A
//
float32 atan(float32 x)
{
  float32 y, res, r;
  int1 s, flag;

  s = 0;
  flag = 0;
  y = x;

  if (x < 0)
```

```
  {
    s = 1;
    y = -y;
  }

  if (y > 1.0)
  {
    y = 1.0/y;
    flag = 1;
  }

  res = pat[0]*y*y + pat[1];
  res = res*y*y + pat[2];
  res = res*y*y + pat[3];

  r = qat[0]*y*y + qat[1];
  r = r*y*y + qat[2];
  r = r*y*y + qat[3];

  res = y*res/r;


  if (flag)                       // for |x| > 1
    res = PI_DIV_BY_TWO - res;
  if (s)
    res = -res;

  return(res);
}
//Overloaded functions for atan() for PCD
// Overloaded function atan() for data type - Float48
#if defined(__PCD__)
float48 atan(float48 x)
{
  float48 y, res, r;
```

```
int1 s, flag;

s = 0;
flag = 0;
y = x;

if (x < 0)
{
  s = 1;
  y = -y;
}

if (y > 1.0)
{
  y = 1.0/y;
  flag = 1;
}

res = pat[0]*y*y + pat[1];
res = res*y*y + pat[2];
res = res*y*y + pat[3];

r = qat[0]*y*y + qat[1];
r = r*y*y + qat[2];
r = r*y*y + qat[3];

res = y*res/r;


if (flag)                       // for |x| > 1
  res = PI_DIV_BY_TWO - res;
if (s)
  res = -res;

return(res);
```

```
}

float64
pat_64[6]={0.999999999999999,2.249923645595566,1.771541617806449,
                0.579678874003185,0.072162702162882,0.002281100141660};

float64
qat_64[6]={2.583256978928510,2.432627277466967,1.016760379885393,
                0.182482977188688,0.011155377971690,0.000100013019160};
// Overloaded function atan() for data type - Float64
float64 atan(float64 x)
{
  float64 y,y2, res, r;
  int1 s, flag;

  s = 0;
  flag = 0;
  y = x;

  if (x < 0)
  {
    s = 1;
    y = -y;
  }

  if (y > 1.0)
  {
    y = 1.0/y;
    flag = 1;
  }
  y2 = y*y;
  res = pat_64[5]*y2 + pat_64[4];
  res = res*y2 + pat_64[3];
  res = res*y2 + pat_64[2];
  res = res*y2 + pat_64[1];
```

```
  res = res*y2 + pat_64[0];

  r = qat_64[5]*y2 + qat_64[4];
  r = r*y2 + qat_64[3];
  r = r*y2 + qat_64[2];
  r = r*y2 + qat_64[1];
  r = r*y2 + qat_64[0];
  r = r*y2 + 1.0;

  res = y*res/r;



  if (flag)                         // for |x| > 1
    res = PI_DIV_BY_TWO - res;
  if (s)
    res = -res;

  return(res);
}
#endif

////////////////////////////////////////////////////////////////////////////
//   float atan2(float y, float x)
////////////////////////////////////////////////////////////////////////////
// Description :computes the principal value of arc tangent of y/x, using the
// signs of both the arguments to determine the quadrant of the return value
// Returns : returns the arc tangent of y/x.
// Date : N/A
//

float32 atan2(float32 y,float32 x)
{
  float32 z;
  int1 sign;
  unsigned int8 quad;
```

```
sign=0;
quad=0; //quadrant
quad=((y<=0.0)?((x<=0.0)?3:4):((x<0.0)?2:1));
if(y<0.0)
{
  sign=1;
  y=-y;
}
if(x<0.0)
{
  x=-x;
}
if (x==0.0)
{
  if(y==0.0)
  {
  #ifdef _ERRNO
  {
    errno=EDOM;
  }
  #endif
  }
  else
  {
    if(sign)
    {
    return (-(PI_DIV_BY_TWO));
    }
    else
    {
    return (PI_DIV_BY_TWO);
    }
  }
}
else
```

```
  {
    z=y/x;
    switch(quad)
    {
      case 1:
      {
        return atan(z);
        break;
      }
      case 2:
      {
//          return (atan(z)+PI_DIV_BY_TWO);  //2L3122
        return (PI-atan(z));
        break;
      }
      case 3:
      {
        return (atan(z)-PI);
        break;
      }
      case 4:
      {
        return (-atan(z));
        break;
      }
    }
  }
}

//Overloaded functions for atan2() for PCD
// Overloaded function atan2() for data type - Float48
#if defined(__PCD__)
float48 atan2(float48 y,float48 x)
{
  float48 z;
```

```
int1 sign;
unsigned int8 quad;
sign=0;
quad=0; //quadrant
quad=((y<=0.0)?((x<=0.0)?3:4):((x<0.0)?2:1));
if(y<0.0)
{
  sign=1;
  y=-y;
}
if(x<0.0)
{
  x=-x;
}
if (x==0.0)
{
  if(y==0.0)
  {
  #ifdef _ERRNO
  {
    errno=EDOM;
  }
  #endif
  }
  else
  {
    if(sign)
    {
    return (-(PI_DIV_BY_TWO));
    }
    else
    {
    return (PI_DIV_BY_TWO);
    }
  }
```

```
  }
  else
  {
    z=y/x;
    switch(quad)
    {
      case 1:
      {
        return atan(z);
        break;
      }
      case 2:
      {
//          return (atan(z)+PI_DIV_BY_TWO);  //2L3122
        return (PI-atan(z));
        break;
      }
      case 3:
      {
        return (atan(z)-PI);
        break;
      }
      case 4:
      {
        return (-atan(z));
        break;
      }
    }
  }
}

// Overloaded function atan2() for data type - Float64
float64 atan2(float64 y,float64 x)
{
  float64 z;
```

```
int1 sign;
unsigned int8 quad;
sign=0;
quad=0; //quadrant
quad=((y<=0.0)?((x<=0.0)?3:4):((x<0.0)?2:1));
if(y<0.0)
{
  sign=1;
  y=-y;
}
if(x<0.0)
{
  x=-x;
}
if (x==0.0)
{
  if(y==0.0)
  {
  #ifdef _ERRNO
  {
    errno=EDOM;
  }
  #endif
  }
  else
  {
    if(sign)
    {
    return (-(PI_DIV_BY_TWO));
    }
    else
    {
    return (PI_DIV_BY_TWO);
    }
  }
```

```
  }
  else
  {
    z=y/x;
    switch(quad)
    {
      case 1:
      {
        return atan(z);
        break;
      }
      case 2:
      {
//          return (atan(z)+PI_DIV_BY_TWO);  //2L3122
        return (PI-atan(z));
        break;
      }
      case 3:
      {
        return (atan(z)-PI);
        break;
      }
      case 4:
      {
        return (-atan(z));
        break;
      }
    }
  }
}
#endif

////////////////////// Hyperbolic functions //////////////////////

////////////////////////////////////////////////////////////////////////////
```

```
//   float cosh(float x)
////////////////////////////////////////////////////////////////////////////
// Description : Computes the hyperbolic cosine value of x
// Returns : returns the hyperbolic cosine value of x
// Date : N/A
//

float32 cosh(float32 x)
{
  return ((exp(x)+exp(-x))*0.5);
}
//Overloaded functions for cosh() for PCD
// Overloaded function cosh() for data type - Float48
#if defined(__PCD__)
float48 cosh(float48 x)
{
  return ((exp(x)+exp(-x))*0.5);
}

// Overloaded function cosh() for data type - Float64
float64 cosh(float64 x)
{
  return ((exp(x)+exp(-x))*0.5);
}
#endif

////////////////////////////////////////////////////////////////////////////
//   float sinh(float x)
////////////////////////////////////////////////////////////////////////////
// Description : Computes the hyperbolic sine value of x
// Returns : returns the hyperbolic sine value of x
// Date : N/A
//

float32 sinh(float32 x)
```

```
{

  return ((exp(x) - exp(-x))*0.5);
}
//Overloaded functions for sinh() for PCD
// Overloaded function sinh() for data type - Float48
#if defined(__PCD__)
float48 sinh(float48 x)
{

  return ((exp(x) - exp(-x))*0.5);
}

// Overloaded function sinh() for data type - Float48
float64 sinh(float64 x)
{

  return ((exp(x) - exp(-x))*0.5);
}
#endif

////////////////////////////////////////////////////////////////////////////
//   float tanh(float x)
////////////////////////////////////////////////////////////////////////////
// Description : Computes the hyperbolic tangent value of x
// Returns : returns the hyperbolic tangent value of x
// Date : N/A
//

float32 tanh(float32 x)
{
  return(sinh(x)/cosh(x));
}
//Overloaded functions for tanh() for PCD
// Overloaded function tanh() for data type - Float48
```

```
#if defined(__PCD__)
float48 tanh(float48 x)
{
  return(sinh(x)/cosh(x));
}

// Overloaded function tanh() for data type - Float64
float64 tanh(float64 x)
{
  return(sinh(x)/cosh(x));
}
#endif

////////////////////////////////////////////////////////////////////////////
//   float frexp(float x, signed int *exp)
////////////////////////////////////////////////////////////////////////////
// Description : breaks a floating point number into a normalized fraction and
an integral
// power of 2. It stores the integer in the signed int object pointed to by exp.
// Returns : returns the value x, such that x is a double with magnitude in the
interval
// [1/2,1) or zero, and value equals x times 2 raised to the power *exp.If value
is zero,
// both parts of the result are zero.
// Date : N/A
//

#define LOG2 .30102999566398119521
#define LOG2_INV 3.32192809488736234787
float32 frexp(float32 x, signed int8 *exp)
{
  float32 res;
  int1 sign = 0;
  if(x == 0.0)
  {
```

```
   *exp=0;
   return (0.0);
}
if(x < 0.0)
{
  x=-x;
  sign=1;
}
if (x > 1.0)
{
   *exp=(ceil(log10(x)*LOG2_INV));
   res=x/(pow(2, *exp));
   if (res == 1)
   {
      *exp=*exp+1;
       res=.5;
   }
}
else
{
   if(x < 0.5)
   {
      *exp=-1;
      res=x*2;
   }
   else
   {
      *exp=0;
       res=x;
   }
}
if(sign)
{
   res=-res;
}
```

```
  return res;
}

//Overloaded functions for frexp() for PCD
// Overloaded function frexp() for data type - Float48
#if defined(__PCD__)
float48 frexp(float48 x, signed int8 *exp)
{
  float48 res;
  int1 sign = 0;
  if(x == 0.0)
  {
    *exp=0;
    return (0.0);
  }
  if(x < 0.0)
  {
   x=-x;
   sign=1;
  }
  if (x > 1.0)
  {
    *exp=(ceil(log10(x)*LOG2_INV));
    res=x/(pow(2, *exp));
    if (res == 1)
    {
      *exp=*exp+1;
       res=.5;
    }
  }
  else
  {
    if(x < 0.5)
    {
      *exp=-1;
```

```
        res=x*2;
      }
      else
      {
        *exp=0;
         res=x;
      }
   }
   if(sign)
   {
     res=-res;
   }
   return res;
}

// Overloaded function frexp() for data type - Float64
float64 frexp(float64 x, signed int8 *exp)
{
   float64 res;
   int1 sign = 0;
   if(x == 0.0)
   {
     *exp=0;
     return (0.0);
   }
   if(x < 0.0)
   {
    x=-x;
    sign=1;
   }
   if (x > 1.0)
   {
     *exp=(ceil(log10(x)*LOG2_INV));
     res=x/(pow(2, *exp));
     if (res == 1)
```

```
    {
      *exp=*exp+1;
       res=.5;
    }
  }
  else
  {
    if(x < 0.5)
    {
      *exp=-1;
      res=x*2;
    }
    else
    {
      *exp=0;
       res=x;
    }
  }
  if(sign)
  {
    res=-res;
  }
  return res;
}
#endif

////////////////////////////////////////////////////////////////////////////
//   float ldexp(float x, signed int *exp)
////////////////////////////////////////////////////////////////////////////
// Description : multiplies a floating point number by an integral power of 2.
// Returns : returns the value of x times 2 raised to the power exp.
// Date : N/A
//

float32 ldexp(float32 value, signed int8 exp)
```

```
{
   return (value * pow(2,exp));
}
//Overloaded functions for ldexp() for PCD
// Overloaded function ldexp() for data type - Float48

#if defined(__PCD__)
float48 ldexp(float48 value, signed int8 exp)
{
   return (value * pow(2,exp));
}
// Overloaded function ldexp() for data type - Float64
float64 ldexp(float64 value, signed int8 exp)
{
   return (value * pow(2,exp));
}
#endif

#endif
```