



INSTITUTO TECNOLÓGICO SUPERIOR DE TANTOYUCA

TITULACIÓN INTEGRAL TESIS PROFESIONAL

**“PUESTA EN MARCHA Y ELABORACIÓN DE UN MANUAL DE
PRÁCTICAS CON EL MÓDULO PIC3000”**

Que para obtener el título de:

INGENIERO EN ELECTRÓNICA

Presenta

MARISOL MARTINEZ MARTINEZ

No. de control 143S0011

ASESOR

M.I.I PABLO IVÁN ROMERO DE LA ROSA

Tantoyuca, Ver., Julio de 2019

AGRADECIMIENTOS

Agradezco a mi padre celestial por las bendiciones que derramo en mi vida en los tiempos de dificultad y debilidad. A mis padres Ubaldo y María por los principios, valores y consejos que me inculcaron durante el transcurso de mi carrera profesional.

A mis tíos por extenderme su apoyo en momentos difíciles, y por el amor que me brindaron en cada día que me sentía desconsolada.

Mi reconocimiento a los docentes de la División de Ingeniería Electrónica por haber compartido sus conocimientos a lo largo de mi formación, de manera especial al M.I.I .Pablo Iván Romero de la Rosa asesor del proyecto quien ha guiado con su paciencia y rectitud como docente, a los revisores M.C Raúl Hernández Rivera y M.C Orlando Meza Zaleta por las observaciones realizadas en el proyecto.

Marisol Martínez Martínez

RESUMEN

El presente proyecto consiste en elaborar un manual de prácticas con el Módulo DIGIAC3000 (16.1) con un microcontrolador PIC16F84A, es un controlador de interface periférico. El cual se desarrollan los programas en distintos lenguajes de programación a través de una serie de pasos hasta grabar el programa al microcontrolador. Para el desarrollo, edición y compilación de los programas realizados en este manual, se utiliza el entorno de desarrollo MPLAB IDE en lenguaje ensamblador, y PIC C COMPILER en lenguaje C, ambos softwares se ejecutan en el sistema operativo Windows 10, al realizar la compilación del código fuente el software genera un archivo .hex que se utiliza para grabar físicamente en el microcontrolador, utilizando el programador Master Prog, antes de subir el archivo se simula con el software ISIS-PROTEUS para comprobar el funcionamiento del programa, se implementan las siguientes prácticas, configuración de entradas/salidas, lectura de diferentes sensores y el proyecto final automatización de una cisterna.

ABSTRACT

The present project consists in developing a practical manual with the DIGIAC3000 Module (16.1) with a PIC16F84A microcontroller, it is a peripheral interface controller. Which programs are developed in different programming languages through a series of steps to record the program to the microcontroller. For the development, edition and compilation of the programs made in this manual, the MPLAB IDE development environment is used in assembly language, and PIC C COMPILER in C language, both software are executed in the Windows 10 operating system, when compiling from the source code the software generates a .hex file that is used to physically record in the microcontroller, using the Master Prog programmer, before uploading the file it is simulated with the ISIS-PROTEUS software to check the operation of the program, the following are implemented practices, input-output configuration, reading of different sensors and the final project automation of a cistern.

Tabla de contenido

AGRADECIMIENTOS	ii
RESUMEN.....	iii
INTRODUCCIÓN	16
CAPÍTULO 1. MARCO REFERENCIAL.....	17
1.1 DESCRIPCIÓN DE LA EMPRESA U ORGANIZACIÓN Y DEL PUESTO O AREA DEL TRABAJO.....	18
1.2 JUSTIFICACIÓN.....	19
1.3 OBJETIVOS.....	20
1.3.1 OBJETIVO GENERAL.....	20
1.3.2 OBJETIVOS ESPECÍFICOS.....	20
CAPÍTULO 2. MARCO TEÓRICO	21
2.1 ¿QUÉ ES UN MICROCONTROLADOR?	22
2.2 DIFERENCIA ENTRE MICROCONTROLADOR Y MICROPROCESADOR	22
2.3 HISTORIA DE LOS MICROCONTROLADORES.	23
2.4 FAMILIAS DE MICROCONTROLADORES.	24
2.4.1 PIC12CXXX FAMILIA DE GAMA BAJA.....	24
2.4.2 PIC16C5X FAMILIA DE GAMA BÁSICA.....	25
2.4.3 PIC16CXXX FAMILIA DE GAMA MEDIA.....	26
2.4.4 PIC 17CXX FAMILIA DE GAMA ALTA.....	27
2.5 GENERALIDADES DEL PIC16F84A	29
2.5.1 DESCRIPCIÓN GENERAL DEL PIC16F84A.....	29
2.5.2 CARACTERÍSTICAS DE LA ESTRUCTURA RISC DEL CPU.....	33
2.5.3 CARACTERÍSTICAS DE LOS PERIFÉRICOS.....	33
2.6 ARQUITECTURA DEL PIC16F84A.....	33
2.6.1 ALU (UNIDAD LÓGICA ARITMÉTICA)	34
2.6.3 MEMORIA DE PROGRAMA.....	35
2.6.4 FRECUENCIA DE FUNCIONAMIENTO.....	35
2.7 ORGANIZACIÓN DE MEMORIA	38
2.7.1 ORGANIZACIÓN DE LA MEMORIA DE PROGRAMA	38
2.7.2 ORGANIZACIÓN DE LA MEMORIA DE DATOS	40

2.7.3 DIRECCIONAMIENTO DE LA MEMORIA DE DATOS RAM	50
2.7.4 LA PILA.....	53
2.8 LOS PUERTOS DE ENTRADA/SALIDA	54
2.8.1 CONSIDERACIONES DE LOS PUERTOS.....	55
2.8.2 PUERTO A (REGISTROS PORTA Y TRISA)	56
2.8.3 PUERTO B (REGISTROS PORTB Y TRISB)	59
2.9 MÓDULO TEMPORIZADOR/CONTADOR TMR0	62
2.9.1 REGISTRO OPTION	67
2.10 MEMORIA EEPROM.....	68
2.10.1 MAPA DE MEMORIA EEPROM.....	69
2.10.2 MANEJO DE LA EEPROM	69
2.10.3 OPERACIÓN DE LECTURA EN LA EEPROM.....	72
2.10.4 OPERACIÓN DE ESCRITURA EN LA EEPROM.....	72
2.11 INTERRUPTIONES.....	73
2.11.1 CAUSAS DE UNA INTERRUPCIÓN	75
2.11.2 REGISTRO DE CONTROL DE INTERRUPTIONES INTCON	76
2.11.3 INTERRUPCIÓN EXTERNA INT	79
2.11.4 INTERRUPCIÓN POR DESBORDAMIENTO DEL TMR0	80
2.11.5 INTERRUPCIÓN POR CAMBIO DE ESTADO EN RB7:RB4.....	82
2.11.6 INTERRUPCIÓN POR FINALIZACIÓN DE LA ESCRITURA EN LA EEPROM...	83
2.12 EL PERRO GUARDIÁN (WDT)	83
2.13 REINICIALIZACIÓN O RESET	85
2.13.1 CAUSAS QUE PROVOCAN EL RESET	86
2.13.2 ESQUEMA ELÉCTRICO DEL RESET	88
2.13.4 RESET POR FALLO EN LA ALIMENTACIÓN.....	89
2.14 MODO DE REPOSO O BAJO CONSUMO DE ENERGÍA (SLEEP)	89
2.14.1 SALIDA DEL ESTADO DE REPOSO	90
2.14.2 SALIDA DEL ESTADO DE REPOSO MEDIANTE UNA INTERRUPCIÓN	91
2.15 TIPOS DE OSCILADORES.....	93
2.15.1 OSCILADOR RC.....	94
2.15.2 OSCILADORES DE CRISTAL / RESONADORES CERÁMICOS	95
2.15.3 OSCILADOR LP	96

2.15.4 OSCILADOR XT	96
2.15.5 OSCILADOR HS.....	97
2.15.6 OTROS CIRCUITOS OSCILADORES	97
2.15.6.1 OSCILADORES DE RESONANCIA EN PARALELO.....	97
2.15.6.2 OSCILADOR DE RESONANCIA SERIE.....	98
2.16 PALABRA DE CONFIGURACIÓN	98
2.17 PALABRA DE IDENTIFICACIÓN (ID)	100
2.18 CIRCUITERÍA FIJA	101
2.19 JUEGO DE INSTRUCCIONES	102
2.19.1 CLASIFICACIÓN DE FORMATOS	102
2.19.2 INSTRUCCIONES DEL PIC16F84A.....	104
2.19.3 INSTRUCCIONES QUE MANEJAN REGISTROS.....	105
2.19.4 INSTRUCCIONES QUE MANEJAN BITS.....	105
2.19.5 INSTRUCCIONES DE BRINCO (SKIP)	106
2.19.6 INSTRUCCIONES QUE MANEJAN OPERANDOS INMEDIATOS.....	106
2.19.7 INSTRUCCIONES DE CONTROL Y ESPECIALES	107
2.20 SENSORES Y TRANSDUCTORES	109
2.20.1 SENSOR DE LUZ.....	109
2.20.2 SENSOR CNY70.....	110
2.20.3 SENSOR ULTRASÓNICO	111
2.20.4 INTERRUPTORES MECÁNICOS.....	111
2.20.5 INTERRUPTORES DE ESTADO SÓLIDO	112
2.20.6 INTEGRADO LM358	113
2.20.7 DRIVER L293D.....	114
2.20.8 MOTOR CD.....	115
2.20.9 LCD 16X2 LMO16L	116
2.20.10 MOTORES PAP	117
CAPÍTULO 3 MARCO METODOLÓGICO	119
3.1 TIPO DE ESTUDIO	120
3.2 FUENTES DE INFORMACIÓN	120
3.3 METODOLOGÍA	120
CAPÍTULO 4. MARCO OPERATIVO	121

4.1 INSTALACIÓN DEL PROGRAMA MPLAB IDE v8.92.....	122
4.2 INTALACIÓN DE PROTEUS 7.8.....	127
4.3 INSTALACIÓN DE LA MÁQUINA VIRTUAL	133
CAPÍTULO 5. RESULTADOS	138
5.1 PRÁCTICA 1 CONFIGURACIÓN DE ENTRADAS Y SALIDAS	139
5.2 PRÁCTICA 2 DISPLAY DE 7 SEGMENTOS	142
5.3 PRÁCTICA 3 MENSAJE	146
5.4 PRÁCTICA 4 MOTOR DC	151
5.5 PRÁCTICA 5 MOTOR PASO A PASO	156
5.6 PRÁCTICA 6 SENSOR CNY70	162
5.7 PRÁCTICA 7 SENSOR LDR.....	167
5.8 PRÁCTICA 8 SENSOR ULTRASÓNICO	171
5.9 PRÁCTICA 9 SERVO MOTOR	179
5.10 PRÁCTICA 10 TECLADO PHONE.....	185
5.11 PRÁCTICA 11 TMR0 COMO CONTADOR	190
5.12 PRÁCTICA 12 AUTOMATIZACIÓN DE UNA CISTERNA.....	193
CAPÍTULO 6. CONCLUSIONES Y RECOMENDACIONES	200
6.1 CONCLUSIONES	201
6.2 RECOMENDACIONES	202
REFERENCIAS.....	203
ANEXOS.....	205
MÓDULO DIGIAC 3000	205
MASTER PROG.....	206

ÍNDICE DE FIGURAS

- Figura 1. Ubicación del Instituto Superior de Tantoyuca.
- Figura 2. Diagrama de conexiones de los pic12cxxx de la gama baja
- Figura 3. Diagrama de conexiones del PIC16C54/56
- Figura 4. Diagrama de conexiones de los PIC16C8X, modelos representativos de la gama media
- Figura 5. Diagrama de conexiones Del PIC17C75X
- Figura 6. Diagrama de pines del PIC16F8X
- Figura 7. Arquitectura interna Del PIC16F84A
- Figura 8. Impulsos del reloj externo (OSC1) Q1, Q2, Q3 y Q4, configuran un ciclo de instrucción
- Figura 9. La segmentación permite traslapar en el mismo ciclo la fase de ejecución de una instrucción y la de búsqueda de la siguiente, excepto en las instrucciones de salto
- Figura 10. Organización de la memoria de programa
- Figura 11. Organización de la memoria RAM del PIC16F84A
- Figura 12. Dirección y mapeo de bits del registro ESTADO (STATUS)
- Figura 13. Dirección y mapeo de bits del registro OPTION y TMR0
- Figura 14. Mapeo de bits del registro INTCON
- Figura 15. Carga del contador de programa
- Figura 16. Mapeo general de bancos de los dispositivos PIC
- Figura 17. Direccionamiento directo en el PIC16F84
- Figura 18. Direccionamiento indirecto en los dispositivos PIC
- Figura 19. La instrucción CALL y la interrupción, provocan la carga automática del contenido del PC en la pila
- Figura 20. Las instrucciones de RETORNO de subrutina o de interrupción descargan automáticamente el contenido Del nivel 1 de la pila sobre el PC
- Figura 21. Mapeo de direcciones de los puertos
- Figura 22. Los puertos no utilizados se deben conectar a la fuente
- Figura 23. Capacidad máxima de corriente que soportan los puertos

Figura 24. Ubicación de PORTA y TRISA en los bancos del PIC

Figura 25. Diagrama de conexiones de los pines RA3-RA0 a las señales Del procesador

Figura 26. Ubicación de PORTB y TRISB en los bancos Del PIC

Figura 27. Conexionado de los pines RB7 – RB4 y las líneas correspondientes al bus interno de datos y las señales de control.

Figura 28. Conexionado de los pines RB3 – RB0 y las líneas correspondientes al bus interno de datos y las señales de control

Figura 29. Diagrama simplificador Del temporizador TMR0

Figura 30. Esquema simplificado del circuito de control de tiempos usado en los PIC16X8X

Figura 31. Esquema general del funcionamiento del TMR0.

Figura 32. Mapeo de bits del registro OPTION

Figura 33. Mapeo de memoria EEPROM del PIC16F84

Figura 34. Ubicación de los registros EEDATA, EEADR, EECON1 y EECON2

Figura 35. Bits correspondientes al registro EECON1

Figura 36. Localización del vector de interrupción

Figura 37. Organigrama del desarrollo de una interrupción, teniendo en cuenta el papel del bit GIE

Figura 38. Bits que conforman el registro INTCON

Figura 39. Lógica de control para la generación de una interrupción en los PIC16X8X

Figura 40. Diagrama a bloques de la interrupción por desbordamiento Del TMR0

Figura 41. Estructura y conexionado del teclado matriciales a los pines del PUERTO B del PIC

Figura 42. Diagrama a bloques de la operación Del perro guardián (WDT)

Figura 43. Diagrama de conexión para controlar el reset del PIC

Figura 44. El diodo impide que circule corriente del positivo al interior del microcontrolador

Figura 45. Diagrama simplificado a bloques del Reset

Figura 46. Circuito auxiliar de Reset

Figura 47. Cronograma del despertar del PIC al producirse una interrupción

Figura 48. Oscilador de tipo RC

Figura 49. Oscilador de tipo HS, XT o LP

Figura 50. Entrada de reloj externa (configuración de oscilador HS, XT o LP)

Figura 51. Circuito típico para el oscilador XT

Figura 52. Oscilador de Cristal y resonancia en paralelo

Figura 53. Oscilador de Cristal y resonancia en serie

Figura 54. Estructura interna de la palabra de configuración

Figura 55. Palabras de identificación. Solo son válidos los 4 bits de menos peso.

Figura 56. Se resaltan los cinco pines del pic16f84 que soportan las tres funciones principales: alimentación, frecuencia de referencia y reset.

Figura 57. Esquema interno del CNY70

Figura 58. Partes del módulo

Figura 59. Patillaje del LCD LM016

Figura 60. Descomprimir el archivo setup

Figura 61. Bienvenido a la instalación del programa MPLAB.

Figura 62. Términos y condiciones de Microchip.

Figura 63 Instalación de todas las funciones del programa.

Figura 64. Selección del directorio de instalación.

Figura 65. Términos de licencia Microchip.

Figura 66. Componentes que se instalaran en el programa

Figura 67. Proceso de instalación

Figura 68 La instalación se ha terminado

Figura 69. Mensaje de bienvenida a la instalación del programa Proteus 7.8.

Figura 70. Configuración de licencia.

Figura 71. Términos de licencia

Figura 72. La licencia esta instalado.

Figura 73. Selección del directorio de instalación.

Figura 74. Selección de componentes a instalar

Figura 75. Selección del archivo donde se utilizara el programa.

Figura 76. Proceso de instalación

Figura 77. La instalación ha terminado

Figura 78. Selección de la carpeta “LXK Proteus 7.8 SP2 RUS v1.0.0”

Figura 79. Proceso de instalación terminado.

Figura 80. Descarga del programa Virtual Box.

Figura 81. Mensaje de bienvenida Virtual Box

Figura 82. Instalación de la máquina virtual

Figura 83. Selección de funciones con las cuales se instalara el programa.

Figura 84 selección de las características de instalación.

Figura 85. Procede en la instalación

Figura 86. Instalación en progreso

Figura 87. Diagrama de la práctica configuración de entradas y salidas.

Figura 88. Implementación de la práctica configuración de entradas y salidas.

Figura 89. Simulación de la práctica display de 7 segmentos

Figura 90. Implementación de la práctica display de 7 segmentos

Figura 91. Subrutina de visualización mensajes en movimiento

Figura 92. Simulación de la práctica Mensaje

Figura 93. Esquema de la práctica Mensaje.

Figura 94. Implementación de la práctica Mensaje

Figura 95. Organigrama del programa Motor DC

Figura 96. Simulación de la práctica Motor DC

Figura 97. Esquema de la práctica motor DC.

Figura 98. Implementación de la práctica Motor DC.

Figura 99. Organigrama Motor PAP.

Figura 100. Simulación de la práctica Motor Paso a Paso.

Figura 101. Esquema de la práctica motor Paso a Paso.

Figura 102. Implementación de la práctica Motor Paso a Paso.

Figura 103. Diagrama Sensor CNY70

Figura 104. Esquema de la práctica sensor CNY70

Figura 105. Implementación de la práctica CNY70

Figura 106. Simulación de la práctica sensor LDR.

Figura 107. Esquema de la práctica sensor LDR

Figura 108. Implementación de la práctica sensor LDR

Figura 109. Simulación de la práctica sensor ultrasónico

Figura 110. Esquema de la práctica sensor Ultrasónico

Figura 111. Implementación de la práctica sensor ultrasónico

Figura 112. Simulación de la práctica servo motor

Figura 113. Esquema de la práctica servo motor

Figura 114. Implementación de la práctica servo motor.

Figura 115. Organigrama Teclado phone

Figura 116. Simulación de la práctica teclado phone.

Figura 117. Esquema de la práctica teclado phone

Figura 118. Implementación de la práctica teclado phone.

Figura 119. Simulación de la práctica TIMER0 como contador.

Figura 120. Esquema de la práctica TIMER0 como contador.

Figura 121. Implementación de la práctica TMR0 como contador.

Figura 122. Simulación de la práctica control de un depósito de líquidos

Figura 123. Esquema de la práctica control de un depósito de líquidos

Figura 124. Placa de circuitos DIGIAC 3000.

Figura 125. Mater Prog.

ÍNDICE DE TABLAS

- Tabla 1. Características de los modelos PIC12C (F) XXX de la gama baja
- Tabla 2. Principales características de los modelos de la gama básica
- Tabla 3. Características relevantes de los modelos PIC16X8X de la gama media
- Tabla 4. Características más destacadas de los modelos PIC17CXXX de la gama alta
- Tabla 5. Lista de características del PIC16F8X
- Tabla 6. Organización detallada de los registros del PIC16F84A
- Tabla 7. Valor de preescala asignado al divisor de frecuencia
- Tabla 8. Descripción de las terminales del puerto A
- Tabla 9. Descripción de las terminales del puerto B
- Tabla 10. Distribución y asignación de funciones de los bits del registro OPTION
- Tabla 11. Valores que toman los bits de los registros SFR tras los posibles Reset
- Tabla 12. Los bits TO y PD del registro de STATUS determinan la causa que ha originado el Reset
- Tabla 13. Consecuencias del modo de reposo
- Tabla 14. Status y significado de los señalizadores PD y TO
- Tabla 15. Instrucciones que ejecuta el PIC cuando “despierta” por una interrupción según el valor de bit GIE
- Tabla 16. Valores recomendados para los componentes del oscilador RC con respecto a la frecuencia deseada
- Tabla 17. Tabla de selección de cristal y capacitor
- Tabla 18. Combinaciones para elegir el tipo de oscilador
- Tabla 19. Clasificación de formatos
- Tabla 20. Juego de instrucciones del PIC16f84A
- Tabla 21. Principales características de las instrucciones de los PIC16X8X
- Tabla 22. Características más importantes de las dos instrucciones que manejan un bit determinado de un registro.
- Tabla 23. Características más relevantes de las cuatro instrucciones condicionales de brinco
- Tabla 24. Características más importantes de las instrucciones que manejan

operandos inmediatos (k)

Tabla 25. Principales características de las instrucciones del control del flujo del programa y de las especiales

Tabla 26. Función de los pines en un LM016

Tabla 27. Ángulo de posicionamiento del servo motor

Tabla 28. Tabla de verdad del funcionamiento del control de nivel de un depósito

INTRODUCCIÓN

El microcontrolador es uno de los logros más sobresalientes del siglo XX. Hoy en día, existen casi 15,000 millones de PIC's de alguna clase en uso. Actualmente los podemos encontrar en cualquier sitio: microondas, frigoríficos, coches, aviones, mandos a distancia, radios, televisores, etc. El objetivo principal de esta guía consiste en que el lector comprenda el funcionamiento de los microcontroladores y sea capaz de emplearlos en situaciones que requieren una solución práctica. Dentro de este manual es posible encontrar información acerca del origen de esta línea de microcontroladores PIC, así como las diferentes familias que integran la amplia gama de dispositivos PIC Micro. Un aspecto fundamental es la explicación clara y explícita de la operación del microcontrolador PIC16F84A, dispositivo el cual está enfocado este manual, se tratan temas relacionados de las características internas, arquitectura, organización de la memoria, periféricos etc.

El presente proyecto “Puesta en marcha y elaboración de un manual de prácticas con el módulo PIC3000” se divide en 6 capítulos.

En el Capítulo 1 Marco Referencial se exponen la descripción de la institución, el objetivo general y los específicos, así como la justificación del presente proyecto.

En el Capítulo 2 Marco Teórico se ofrece una revisión de literatura en la cual se encuentran características generales del microcontrolador PIC16F84A, de los sensores y actuadores utilizados para el desarrollo de las prácticas propuestas.

En el Capítulo 3 Marco Metodológico se da a conocer la metodología utilizada para alcanzar los objetivos de la presente investigación.

En el Capítulo 4 Marco operativo se explica el proceso de instalación de los softwares utilizados.

En el Capítulo 5 Resultados se propone un Manual de prácticas donde se muestran los códigos, diagramas de conexiones y simulación de dichas prácticas.

Y por último se presenta el Capítulo 6 Conclusiones y Recomendaciones.

CAPÍTULO 1.

MARCO

REFERENCIAL

1.1 DESCRIPCIÓN DE LA EMPRESA U ORGANIZACIÓN Y DEL PUESTO O AREA DEL TRABAJO

En octubre 9 de 1995 se inician las actividades del Instituto Tecnológico con la recepción de solicitudes y otorgamiento de fichas, y es en octubre 23 cuando se dan inicio a las clases con una matrícula de 145 alumnos con las carreras de: Ingeniería en Agronomía e Ingeniería Electrónica.

Las instalaciones provisionales que ocupa el Instituto Tecnológico están ubicadas en el C.B.T. i.s. No. 71.

El Gobierno del Estado adquirió un terreno donde se empezó a construir el edificio que albergara a nuestra Institución y que contara con una primera etapa con una unidad departamental para 7 aulas, laboratorio múltiple, centro de cómputo, biblioteca y un taller de dibujo

La investigación fue realizada en el Instituto Tecnológico Superior de Tantoyuca, Veracruz Desviación Lindero Tametate SN Col. La Morita CP. 92100 Tel/Fax 7898931680 www.itsta.edu.mx

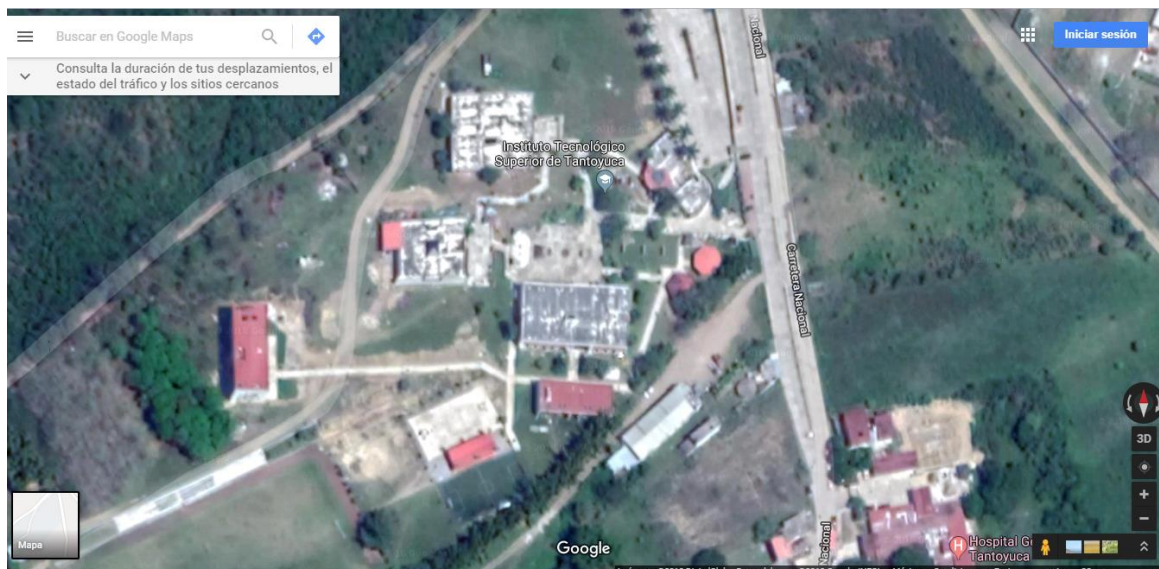


Figura 1. Ubicación del Instituto Superior de Tantoyuca.

1.2 JUSTIFICACIÓN

En la actualidad en este mundo globalizado se requiere de profesionistas que sean capaces de dar solución a los problemas que se presente en las industrias, esto solo es posible cuando el estudiante puede relacionar los conocimientos teórico-prácticos, por tal motivo se desarrollará el presente proyecto, con la finalidad de proporcionar a los estudiantes de las carreras de Electrónica, Mecatrónica y Sistemas Computacionales del Instituto Tecnológico Superior de Tantoyuca un manual que les permita la implementación de circuitos prácticos con el Módulo PIC3000.

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

Realizar un manual de prácticas de Microcontroladores utilizando el Módulo PIC3000 para relacionar los conocimientos teórico-prácticos dentro del Laboratorio de Ingeniería Electrónica del Instituto Tecnológico Superior de Tantoyuca.

1.3.2 OBJETIVOS ESPECÍFICOS

- Indagar en diversas fuentes de información aspectos técnicos de los microcontroladores PIC16F84A.
- Instalar los softwares a utilizar (MPLAB, PROTEUS, MASTER PROG, Oracle VM VirtualBox, DIGIAC 3000 ViP, PIC C COMPILER y Driver RS232).
- Realizar los códigos en lenguaje ensamblador y compilarlo en el software MPLAB.
- Realizar los códigos en lenguaje C y compilarlo en el software PIC C compiler.
- Implementar los circuitos en el software PROTEUS para simular los códigos.
- Compilar los programas en el PIC16F84A mediante el programador MASTER PROG
- Implementar los circuitos en el Módulo PIC3000.

CAPÍTULO 2.

MARCO

TEÓRICO

2.1 ¿QUÉ ES UN MICROCONTROLADOR?

El microcontrolador se emplea en aplicaciones concretas y no es universal como el microprocesador.

Un microcontrolador es un circuito integrado que contiene una Unidad Central de Proceso (CPU) y una serie de recursos internos en un solo encapsulado. El CPU permite que el microcontrolador pueda ejecutar instrucciones almacenadas en una memoria. Los recursos internos son memoria RAM, memoria ROM, memoria EEPROM, puerto serie, puertos de entrada/salida, temporizadores, comparadores, etc. Se puede decir que es una evolución del microprocesador, al añadirle a este último las funciones que antes era necesario situar externamente con otros circuitos. El ejemplo típico está en los puertos de entrada/salida y en la memoria RAM, en los sistemas con microprocesadores es necesario desarrollar una lógica de control y unos circuitos para implementar las funciones anteriores, con un microcontrolador no hace falta porque lo lleva todo incorporado, además en el caso de tener que ampliar el sistema ya ofrece recursos que facilitan esto. En resumen, un microcontrolador es un circuito integrado independiente, que no necesita memoria ni puertos externos pues los lleva en su interior, que facilita la tarea de diseño y reduce el espacio, traduciéndose todo a una aplicación final más económica y fiable.

2.2 DIFERENCIA ENTRE MICROCONTROLADOR Y MICROPROCESADOR

En el diseño de un sistema con un microprocesador, además del procesador y dependiendo del circuito, se requiere de algunos circuitos integrados adicionales, como, por ejemplo: memorias RAM para almacenar los datos temporalmente y memorias ROM para almacenar el programa que se encargará del proceso del equipo, un circuito integrado para los puertos de entrada y salida y finalmente un decodificador de direcciones.

Un microcontrolador es un solo circuito integrado que contiene todos los elementos electrónicos que se utilizaban para hacer funcionar un sistema basado con un microprocesador; es decir contiene en un solo integrado la Unidad de Proceso, la

memoria RAM, memoria ROM, puertos de entrada, salida y otros periféricos.

2.3 HISTORIA DE LOS MICROCONTROLADORES.

En 1965 GI formó una división de microelectrónica, destinada a generar las primeras arquitecturas viables de memoria EPROM y EEPROM. De forma complementaria GI Microelectronics División fue también responsable de desarrollar una amplia variedad de funciones digitales y analógicas en las familias AY3-xxxx y AY5-xxxx.

GI también creó un microprocesador de 16 bit, denominado CP1600, a principios de los 70's. Este fue un microprocesador razonable, pero no particularmente bueno manejando puertos de e/s. Alrededor del año 1975 GI diseñó un controlador de interface periférico (PIC) para algunas aplicaciones muy específicas. Fue diseñado para ser muy rápido, además de ser un controlador de e/s para una máquina de 16 bits pero sin necesitar una gran cantidad de funcionalidades, por lo que su lista de instrucciones fue pequeña.

No es de extrañar que la estructura diseñada en 1975 es, sustancialmente, la arquitectura del actual PIC16C5x. Además, la versión de 1975 fue fabricada con tecnología NMOS y sólo estaba disponible en versiones de ROM de máscara, pero seguía siendo un buen pequeño microcontrolador. El mercado, no obstante, no pensó así y el PIC quedó reducido a ser empleado únicamente por grandes fabricantes.

Durante los 80's, GI renovó su apariencia y se reestructuró, centrando su trabajo en sus principales actividades, semiconductores de potencia esencialmente, lo cual siguen haciendo actualmente con bastante éxito. GI Microelectronics División cambió a GI Microelectronics Inc. (una especie de subsidiaria), la cual fue finalmente vendida en 1985 a Venture Capital Investors, incluyendo la fábrica en Chandler, Arizona. La gente de Ventura realizó una profunda revisión de los productos en la compañía, desechando la mayoría de los componentes AY3, AY5 y otra serie de cosas, dejando sólo el negocio de los PIC y de las memorias EEPROM y EPROM.

Se tomó la decisión de comenzar una nueva compañía, denominada Arizona Microchip Technology, tomando como elemento diferenciador sus controladores integrados.

Como parte de esta estrategia, la familia NMOS PIC165x fue rediseñada para emplear algo que la misma compañía fabricaba bastante bien, memoria EPROM. De esta forma nació el concepto de basarse en tecnología CMOS, OTP y memoria de programación EPROM, naciendo la familia PIC16C5x. Actualmente Microchip ha realizado un gran número de mejoras a la arquitectura original, adaptándola a las actuales tecnologías y al bajo costo de los semiconductores.

2.4 FAMILIAS DE MICROCONTROLADORES.

2.4.1 PIC12CXXX FAMILIA DE GAMA BAJA.

La familia de la gama enana consiste en un grupo de microcontroladores PIC de reciente aparición que ha acaparado la atención del mercado. Su principal característica es su tamaño, ya que todos los integrantes de esta familia poseen 8 pines. Consumen menos de 2 mA cuando trabajan a 5V y 4MHz, además de trabajar con alimentación de corriente continua comprendida entre 2.5V y 5.5V. El formato de sus instrucciones puede ser de 12 o 14 bits y su repertorio es de 33 o 35 instrucciones, respectivamente. En la Figura 2 se muestra el diagrama de conexiones de los microcontroladores PIC12C508 y PIC12C509, pertenecientes a la familia de gama baja. En la Tabla 1 se muestran las características de algunos microcontroladores que forman parte de la gama baja.

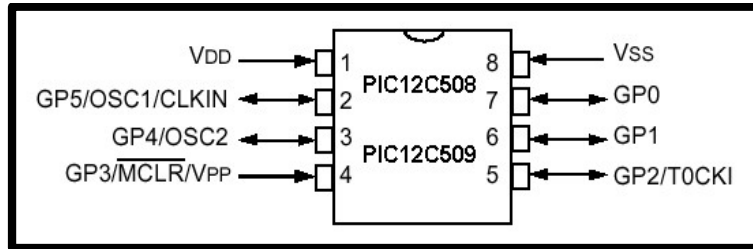


Figura 2. Diagrama de conexiones de los PIC12CXXX de la gama baja

Tabla 1. Características de los modelos PIC12C (F)XXX de la gama baja

MODELO	MEMORIA PROGRAMA	MEMORIA DATOS	FRECUENCIA MÁXIMA	LÍNEAS E/S	ADC 8 BITS	TEMPORIZADORES	PINES
PIC12C508	512x12	25x8	4MHz	6		TMR0 + WDT	8
PIC12C509	1024x12	41x8	4MHz	6		TMR0 + WDT	8
PIC12C670	512x12	80x8	4MHz	6		TMR0 + WDT	8
PIC12C671	1024x12	128x8	4MHz	6	2	TMR0 + WDT	8
PIC12C672	2048x14	128x8	4MHz	6	4	TMR0 + WDT	8
PIC12C680	512x12 FLASH	80x8 16x8 EPROM	4MHz	6	4	TMR0 + WDT	8
PIC12C681	1024x14 FLASH	80x8 16x8 EPROM	4MHz	6		TMR0 + WDT	8

2.4.2 PIC16C5X FAMILIA DE GAMA BÁSICA.

La familia 16C5X se consolidó como la base en el desarrollo de nuevas tecnologías ofreciendo la solución costeable más efectiva. Estos microcontroladores cuentan con un conjunto de instrucciones de 12 bits de ancho, 33 instrucciones, 2 niveles de acumulador, sin interrupciones. En algunos casos la memoria es del tipo ROM, definida en fábrica y se ofrecen en empaquetados de 18, 20 y 22 pines. En opciones de empaquetado SOIC o SSOP. Con bajo voltaje de operación hasta de 2 volt, hacen de esta familia un elemento ideal para ser operado en aplicaciones con baterías. En la Figura 3 se muestra el PIC16C54, microcontrolador perteneciente a la familia de gama básica. En la Tabla 2 se muestran las características de algunos microcontroladores que forman parte de la gama básica.

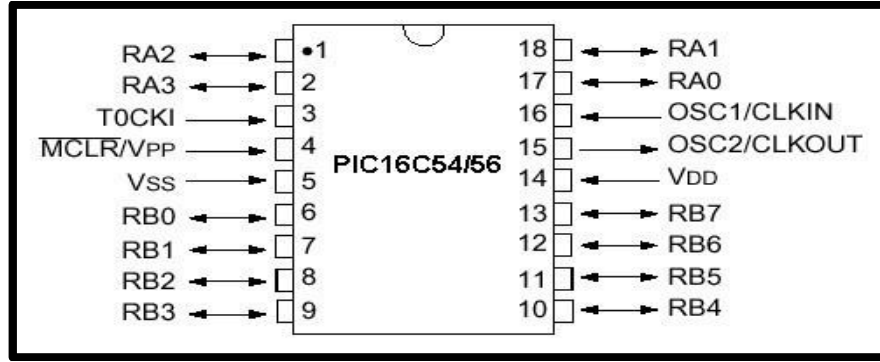


Figura 3. Diagrama de conexiones del PIC16C54/56

Tabla 2. Principales características de los modelos de la gama básica

MODELO	MEMORIA PROGRAMA (x12 Bits)		MEMORIA DATOS (Bytes)	FRECUENCIA MÁXIMA	LÍNEAS E/S	TEMPORIZADORES	PINES
	EEPROM	ROM					
PIC16C52	384		25	4 MHz	4	TMR0 + WDT	18
PIC16C54	512		25	20 MHz	12	TMR0 + WDT	18
PIC16C54A	512		25	20 MHz	12	TMR0 + WDT	18
PIC16CR54A		512	25	20 MHz	12	TMR0 + WDT	18
PIC16C55	512		24	20 MHz	20	TMR0 + WDT	28
PIC16C56	1 K		25	20 MHz	12	TMR0 + WDT	18
PIC16C57	2 K		72	20 MHz	20	TMR0 + WDT	28
PIC16CR57B		2 K	72	20 MHz	20	TMR0 + WDT	28
PIC16C58A	2 K		73	20 MHz	12	TMR0 + WDT	18
PIC16CR58A		2 K	73	20 MHz	12	TMR0 + WDT	18

2.4.3 PIC16CXXX FAMILIA DE GAMA MEDIA.

La familia de rango medio ofrece un amplio rango de opciones, desde empaquetados de 18 hasta 44 pines, así como un alto nivel de integración de periféricos. Esta familia cuenta con un conjunto de instrucciones de 14 bits de ancho, 35 instrucciones, 8 niveles de acumulador. El PIC16C84 posee memoria EEPROM y la capacidad de manejar interrupciones. En la Figura 4 se muestra el PIC16C745, microcontrolador perteneciente a la familia base. En la Tabla 3 se muestran las características de algunos microcontroladores que forman parte de la gama media.

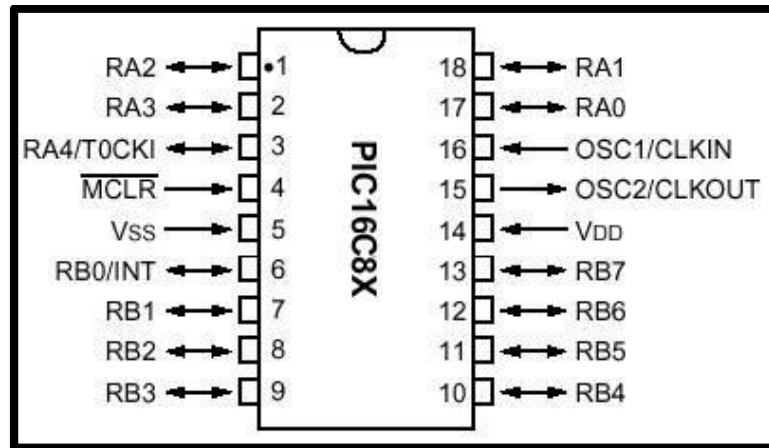


Figura 4. Diagrama de conexiones de los PIC16C8X, gama media

Tabla 3. Características relevantes de los modelos PIC16X8X de la gama media

MODELO	MEMORIA PROGRAMA	MEMORIA DATOS		REGISTROS ESPECÍFICOS	TEMPORIZADORES	INTERRUPCIONES	E/S	PINES
		RAM	EEPROM					
PIC16C84	1Kx14 EEPROM	36	64	11	TMR0 + WDT	4	13	18
PIC16F84	1Kx14 FLASH	68	64	11	TMR0 + WDT	4	13	18
PIC16F83	512x14 FLASH	36	64	11	TMR0 + WDT	4	13	18
PIC16CR84	1Kx14 ROM	68	64	11	TMR0 + WDT	4	13	18
PIC16CR83	512x14 ROM	36	64	11	TMR0 + WDT	4	13	18

2.4.4 PIC 17CXX FAMILIA DE GAMA ALTA.

Esta familia de alto rendimiento ofrece la más alta velocidad de ejecución de todos los microcontroladores de 8 bits de la industria. Utilizando una arquitectura de instrucciones de 16 bits, 55 instrucciones, 16 niveles de acumulador. Mejora el conjunto de instrucciones y las capacidades de interrupción. A menos que se indique, la memoria es del tipo EPROM.

Adicionalmente existen otras familias derivadas, como los PIC16Fxx que emplean memoria del tipo FLASH. Se trata de una memoria no volátil, de bajo consumo, que se puede escribir y borrar en circuito al igual que las EEPROM, pero suelen disponer de mayor capacidad que estas últimas. El borrado sólo es posible con bloques completos y no se pueden realizar sobre posiciones concretas.

En la Figura 5 se muestra el diagrama de conexiones de los PIC17C75X, microcontroladores pertenecientes a la familia de rango alto. En la Tabla 4 se muestran las características de algunos microcontroladores que forman parte de la gama alta.

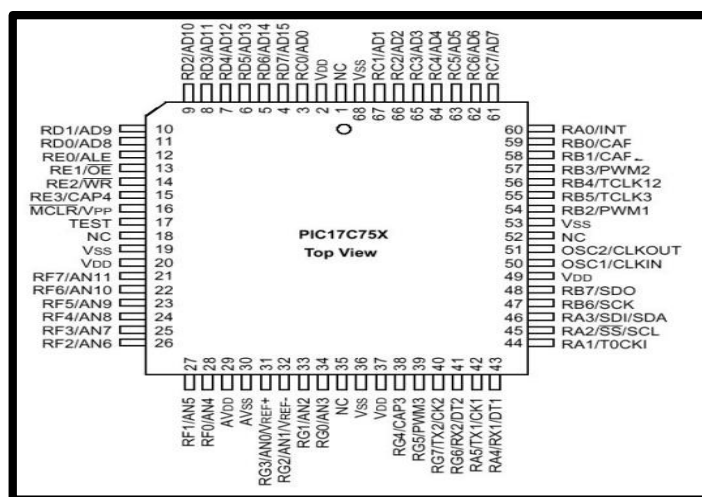


Figura 5. Diagrama de conexiones del PIC17C75X

Tabla 4. Características más destacadas de los modelos PIC17CXXX de la gama alta

MODELO	MEMORIA PROGRAMA	MEMORIA DATOS RAM (Bytes)	REGISTROS ESPECÍFICOS	TEMPORIZADORES	CAP	PWM	CAD 10 bit	INTERRUPCIONES	E/S	MULTIPLICADOR HARDWARE	PINES
PIC17C42A	2Kx16	232	48	4 + WDT	2	2		11	33	8 x 8	40/44
PIC17C43	4Kx16	454	48	4 + WDT	2	2		11	33	8 x 8	40/44
PIC17C44	8Kx16	454	48	4 + WDT	2	2		11	33	8 x 8	40/44
PIC17C752	8Kx16	454	76	4 + WDT	4	3	12	18	50	8 x 8	64/68
PIC17C756	16Kx16	902	76	4 + WDT	4	3	12	18	50	8 x 8	64/68
CAP= Capturador; PWM= Modulador por ancho de pulso; CAD= Convertidor Análogo/Digital											

2.5 GENERALIDADES DEL PIC16F84A

2.5.1 DESCRIPCIÓN GENERAL DEL PIC16F84A

El PIC16F84 al igual que la versión mejorada de mayor velocidad PIC16F84A pertenece a la familia de microcontroladores de 8 bits PIC16CXX, los cuales son de bajo costo, alto rendimiento y CMOS. Este grupo contiene los siguientes dispositivos:

- PIC16F83
- PIC16F84
- PIC16F84A
- PIC16CR83
- PIC16CR84

Todos los microcontrollers emplean una avanzada Arquitectura RISC (Computadores de Juego de Instrucciones Reducido). Los dispositivos PIC16F8X han reforzado características principales, pila profunda de ocho niveles y múltiples fuentes de interrupción internas y externas.

La separación del bus de instrucciones y del bus de datos en la arquitectura Harvard da lugar a un ancho de la palabra de instrucción de 14 bits, con un bus de datos separado de 8 bits de ancho. Los dos conductos de fase de instrucción permiten que todas las instrucciones se ejecuten en un solo ciclo, excepto por los saltos del programa, los cuales requieren dos ciclos.

Un total de 35 instrucciones (Juego de instrucciones reducido) están disponibles. Adicionalmente un set del registro largo es usado para lograr un nivel de alto rendimiento.

Los microcontroladores PIC16F8x típicamente logran una compresión de código de 2:1 y un incremento de 4:1 en la mejora de velocidad (a 20 MHz) sobre otros microcontroladores de 8 bits de su misma clase.

En la Tabla 5 se muestran las características principales de la serie de microcontroladores PIC16F8x.

Los microcontroladores PIC16F8x poseen 68 bytes de RAM, 64 bytes de memoria de datos EEPROM y 13 pines de entrada /Salida. También está disponible un contador / temporizador.

La familia PIC16CXX tiene características especiales para reducir los componentes externos, reduciendo así el costo, reforzando la confiabilidad del sistema y reduciendo el consumo de energía.

Hay cuatro opciones del oscilador, de los cuales el oscilador RC mediante un solo pin, proporciona una solución de bajo costo, el oscilador LP reduce el consumo de energía, el oscilador XT es un cristal estándar y el oscilador HS se refiere a cristales de alta velocidad.

El modo SLEEP (Bajo en consumo de corriente) ofrece un ahorro de energía. El usuario puede despertar al chip del modo SLEEP a través de varias interrupciones internas y externas para reestablecer su funcionamiento.

Un confiable temporizador de perro guardián (WDT) está integrado en el chip, así como su propio oscilador RC proporciona protección cuando el programa ha sido ciclado.

Los dispositivos con memoria de programa Flash permiten que el mismo dispositivo pueda ser usado como prototipo y como elemento de producción.

La gran capacidad de reprogramar el código de circuito permite su actualización sin que el dispositivo sea removido de su aplicación final.

Tabla 5. Lista de características del PIC16F8X

	PIC16F83	PIC16CR83	PIC16F84	PIC16CR84	PIC16F84A
Máxima frecuencia de operación MHz	10	10	10	10	20
Memoria de programa Flash	512	-	1K	-	1K
Memoria de programa EEPROM	-	-	-	-	-
Memoria de programa ROM	-	512	-	1K	-
Memoria de datos (bytes)	36	36	68	68	68
EEPROM de datos (bytes)	64	64	64	64	64
Módulos Temporizadores	TMRO	TMRO	TMRO	TMRO	TMRO
Fuentes de interrupción	4	4	4	4	4
Pines de I/O	13	13	13	13	13
Rangos de Voltaje (Volts)	4.0-6.0	2.0-6.0	4.0-6.0	2.0-6.0	4.0-6.0
Empaques	Dip 18-pin, SOIC	Dip 18-pin, SOIC	Dip 18-pin, SOIC	Dip 18-pin, SOIC	Dip 18-pin, SOIC

Esto es muy útil en situaciones, donde el dispositivo es difícilmente accesible. También es útil en aplicaciones remotas donde el código necesita ser actualizado.

El PIC16F8x encaja perfectamente en rangos de aplicaciones donde se requiere controlar pequeños motores, en la utilización de sensores remotos de baja potencia, cerrojos electrónicos y pequeñas tarjetas con dispositivos de seguridad.

La tecnología Flash/EEPROM optimiza los programas de aplicación (transmisión de códigos, velocidad de motores, receptor de frecuencias, códigos de seguridad, etc.) volviéndolos extremadamente rápidos y convenientes.

Los pequeños empaques hacen perfectos a esta serie de microcontroladores para todas las aplicaciones con limitaciones de espacio.

Bajo costo, bajo consumo de energía, alto rendimiento, fácil de usar y flexibilidad en las entradas/salidas, hacen muy versátil al PIC16F8x en áreas donde no se habían considerado los microcontroladores (funciones de temporizador, comunicación serie, comparadores, moduladores por ancho de pulso, y aplicaciones como procesador).

La medida de programación serial en el sistema (por medio de dos pines), ofrece flexibilidad en la optimización del producto después de haber sido ensamblado y probado.

Esta característica puede ser usada para agregar un número de serie a un producto, una calibración o restablecimiento de datos o programar el dispositivo con la firma de software correspondiente antes de enviarlo.

El diagrama de pines del microcontrolador PIC16F84 se puede apreciar en la siguiente Figura 6.

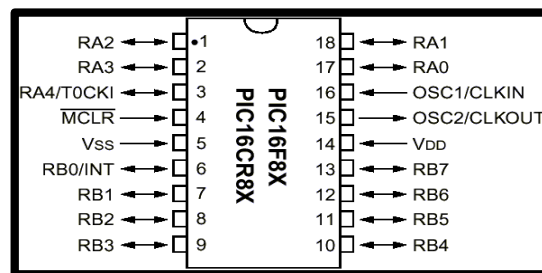


Figura 6. Diagrama de pines del PIC16F8X

2.5.2 CARACTERÍSTICAS DE LA ESTRUCTURA RISC DEL CPU.

Solo 35 sencillas instrucciones para aprender.

Todas las instrucciones son de un ciclo, excepto los saltos de programa que son de dos ciclos.

- Velocidad: 20MHz (0.2us por instrucción).
- Memoria de programa de 1024 palabras. Memoria RAM de 68 bytes.
- Memoria EEPROM de 64 Bytes.
- Palabras de instrucción de 14 bits de ancho.
- Bytes de datos de 8 bits de ancho.
- 15 registros de función especial en el hardware.
- Pila de hardware de 8 niveles de profundidad.
- Modos de direccionamiento directo e indirecto.

Cuatro fuentes de interrupción:

- Pin externo de RB0/INT
- Temporizador TMR0 de sobreflujo
- Interrupción en cambio PORTB <RB7:RB4>
- Escritura completa de la memoria EEPROM

2.5.3 CARACTERÍSTICAS DE LOS PERIFÉRICOS.

- 13 pines de entrada / salida con control individual de dirección
- Fuente / sumidero de alta corriente para controlador directo de LED.
 - Sumidero de 25 mA máximo por pin.
 - Fuente de 25 mA máximo por pin.
 - Temporizador/ contador de 8 bits TMR0 con escala programable de 8 bits.

2.6 ARQUITECTURA DEL PIC16F84A

El alto desempeño de la familia PIC16CXX puede ser atribuido a un número de características comúnmente encontradas en un microprocesador RISC. Los PIC16CXX incluyendo el PIC16F84 usan una arquitectura Harvard. Esta arquitectura tiene la memoria de programa y la de datos en forma separada. Así el dispositivo

tiene un bus de memoria de programa y un bus de memoria de datos. La arquitectura Harvard presenta muchas ventajas sobre la tradicional arquitectura Von Neumann, donde el programa y los datos se sacan de la misma memoria (acceso sobre el mismo bus). Separando la memoria de programa y la memoria de datos permite instrucciones de un tamaño de palabra mayor de 8 bits. La memoria de programa tiene 1K posiciones de 14 bits cada una (1K x 14). La mayor parte de sus instrucciones se ejecutan en un ciclo de reloj excepto las instrucciones de salto. Otra aportación frecuente que aumenta el rendimiento del computador es el fomento del paralelismo implícito, que consiste en la segmentación del procesador (pipe-line), descomponiéndolo en etapas para poder procesar una instrucción diferente en cada una de ellas y trabajar con varias a la vez. El PIC16F84 contiene una ALU de 8 bits de trabajo. Un diagrama a bloques simplificado para el PIC16F84 se muestra en la Figura 7.

2.6.1 ALU (UNIDAD LÓGICA ARITMÉTICA)

La ALU es una unidad aritmética de propósito general. Realiza operaciones aritméticas y booleanas entre registros. La ALU es de 8 bits de ancho y capaz de realizar suma, substracción y operaciones lógicas entre registro de propósito específico, registro de propósito general y el registro de trabajo W. Dependiendo de la instrucción a ejecutar, la ALU puede afectar los valores de acarreo o “carry” (C), el acarreo de dígito o “digit carry” (DC), y Cero (Z) estos bits forman parte del registro ESTADO (STATUS) y son también llamados banderas o señalizadores.

2.6.2 REGISTRO W

El registro de trabajo W es de 8 bits y es usado para las operaciones de la ALU. No es un registro que tenga una dirección específica. Se accede a él mediante las instrucciones o mnemónicos que involucran la utilización de este registro.

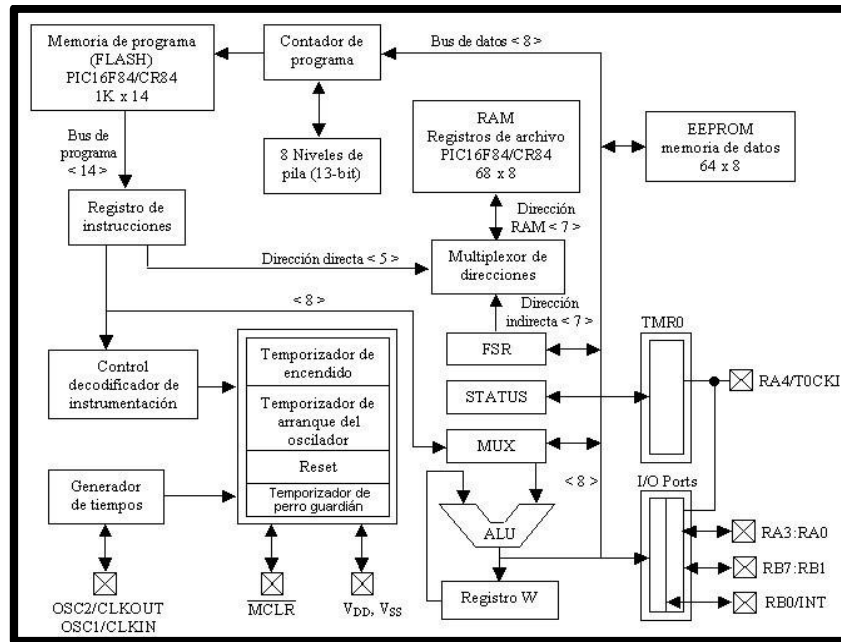


Figura 7. Arquitectura interna del PIC16F84A

2.6.3 MEMORIA DE PROGRAMA

La memoria de programa del PIC16F84 es de tipo Flash de un tamaño de 1K x 14. Por sus mejores prestaciones, la memoria Flash está sustituyendo a la memoria EEPROM para contener instrucciones. De esta forma, Microchip comercializa dos microcontroladores prácticamente iguales que sólo se diferencian en que la memoria de programa de uno de ellos es tipo EEPROM y la del otro tipo Flash. Se trata del PIC16C84 y el PIC16F84, respectivamente.

2.6.4 FRECUENCIA DE FUNCIONAMIENTO

La frecuencia de trabajo del microcontrolador es un parámetro fundamental a la hora de establecer la velocidad en la ejecución de instrucciones y el consumo de energía.

Cuando un PIC16X8X funciona a 10 MHz, que es su máxima frecuencia, le corresponde un ciclo de instrucción de 400ns, puesto que cada instrucción tarda en ejecutarse cuatro periodos de reloj, o sea, $4 \times 100\text{ns} = 400\text{ns}$. Todas las instrucciones del PIC se realizan en un ciclo de instrucción, menos las de salto, que

tardan el doble.

Los impulsos de reloj entran por la terminal OSC1/CLKIN y se dividen por 4 internamente, dando lugar a las señales Q1, Q2, Q3 y Q4, mostradas en la Figura 8. Durante un ciclo de instrucción, que comprende las 4 señales mencionadas, se desarrollan las siguientes operaciones.

Q1: Durante este pulso se incrementa el Contador de Programa.

Q4: Durante este pulso se busca el código de la instrucción en la memoria del programa y se carga en el registro de Instrucciones.

Q2-Q3: Durante la activación de estas dos señales se produce la decodificación y la ejecución de la instrucción.

Para conseguir ejecutar cada instrucción en un ciclo de instrucción (excepto las de salto, que tardan dos) se aplica la técnica de la segmentación o «Pipe-line», que consiste en realizar en paralelo las dos fases que comprende cada instrucción.

En realidad, cada instrucción se ejecuta en dos ciclos: en el primero se lleva a cabo la fase de búsqueda del código de la instrucción en la memoria del programa y en el segundo se decodifica y se ejecuta (fase de ejecución). La estructura segmentada del procesador permite realizar al mismo tiempo la fase de ejecución de una instrucción y la de búsqueda de la siguiente. Cuando la instrucción ejecutada corresponde a un salto no se conoce cuál será la siguiente hasta que se complete, por eso en esta situación se sustituye la fase de búsqueda de la siguiente instrucción por un ciclo «vacío», originando que las instrucciones de salto tarden en realizarse dos ciclos de instrucción, como se puede apreciar en la Figura 9.

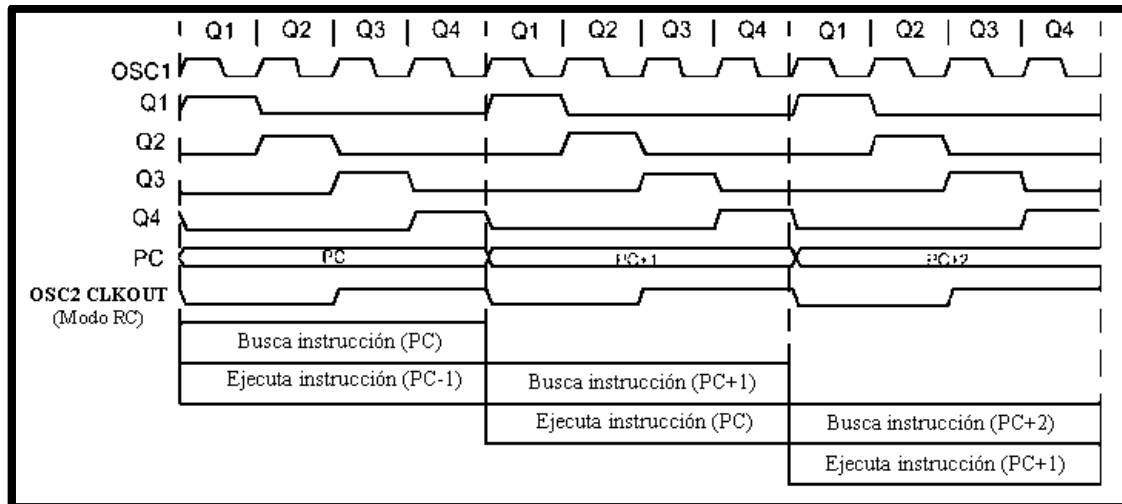


Figura 8. Los impulsos del reloj externo (OSC1) se dividen por 4 formando las señales Q1, Q2, Q3 y Q4 que configuran un ciclo de instrucción

La técnica de la segmentación unida a la arquitectura Harvard del procesador permite al PIC16F84 superar la velocidad de sus competidores directos. Así, por ejemplo, es 1.54 veces más rápido que el microcontrolador de Motorola 68HC05 cuando ambos funcionan a la misma frecuencia de 4 MHz.

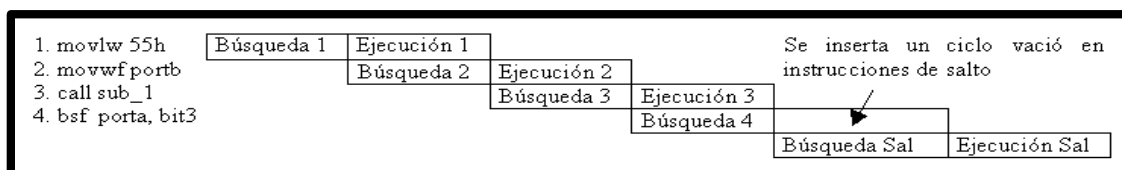


Figura 9. La segmentación permite traslapar en el mismo ciclo la fase de ejecución de una instrucción y la de búsqueda de la siguiente, excepto en las instrucciones de salto

2.7 ORGANIZACIÓN DE MEMORIA

Hay dos clases de memoria en el PIC16F84. Estas son la memoria de programa y la memoria de datos. Cada memoria tiene su propio bus, para que el acceso a cada memoria ocurra durante el mismo ciclo de oscilación.

La memoria de datos puede dividirse en dos tipos de memoria, memoria RAM de propósito general y en registros de función especial (SFR). El área de memoria de datos también contiene los datos de memoria EEPROM. Esta memoria no se traza directamente en la memoria de datos, pero se traza indirectamente. Es decir, un indicador de dirección indirecto especifica la dirección de la memoria de datos EEPROM al leer/escribir. Los 64 bytes de la memoria de datos de la EEPROM tienen un rango de dirección de 0h-3Fh.

2.7.1 ORGANIZACIÓN DE LA MEMORIA DE PROGRAMA

La memoria de programa del PIC16F84 es una memoria de 1 Kbyte de longitud y con palabras de 14 bits. Como es del tipo Flash se puede programar y borrar eléctricamente, lo que facilita el desarrollo de los programas y la experimentación. En ella se graba o almacena, el programa o códigos que el microcontrolador debe ejecutar. Dado que el PIC16F84 tiene un contador de programa de 13 bits, tiene una capacidad de direccionamiento de 8K x 14, pero solamente tiene implementado el primer 1K x 14 (0000h hasta 03FFh), tal como se puede apreciar en la Figura 10. Si se direccionan posiciones de memoria superiores a 3FFh se causará un solapamiento con el espacio del primer 1Kbyte (1K x 14).

Para los PIC16F84 y PIC16CR84, solo el primer 1K x 14 (0000h-03FFh) de memoria de programa esta físicamente implementado.

El vector de reset está en la posición 0000h y el vector de interrupción está en la posición 0004h.

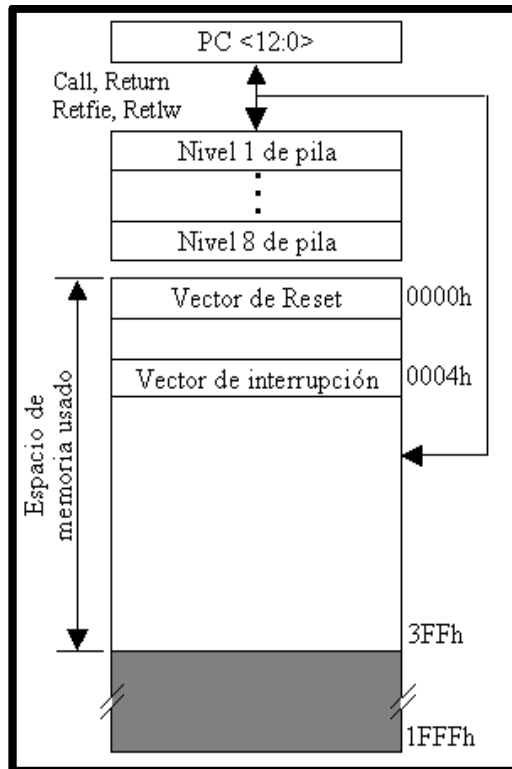


Figura 10. Organización de la memoria de programa

2.7.1.1 VECTOR DE RESET

Cuando ocurre un reset en el microcontrolador, el contador de programa se pone en ceros (0000h). Por esta razón, en la primera dirección del programa se debe escribir todo lo relacionado con la inicialización del mismo.

2.7.1.2 VECTOR DE INTERRUPCIÓN

Cuando el microcontrolador recibe una señal de interrupción, el contador de programa apunta a la dirección 0004h de la memoria de programa, por eso allí se debe escribir toda la programación necesaria para atender dicha interrupción.

2.7.2 ORGANIZACIÓN DE LA MEMORIA DE DATOS

El PIC16F84 puede direccionar 128 posiciones de memoria RAM, pero solo tiene implementados físicamente los primeros 80 (00h-4Fh en hexadecimal). De estos las primeras 12 son registros que cumplen un propósito especial (SFR) en el control del microcontrolador y las 68 siguientes son registros de uso general (GPR) que se pueden usar para guardar los datos temporales de la tarea que se está ejecutando, justo como se muestra en la siguiente Figura 11.

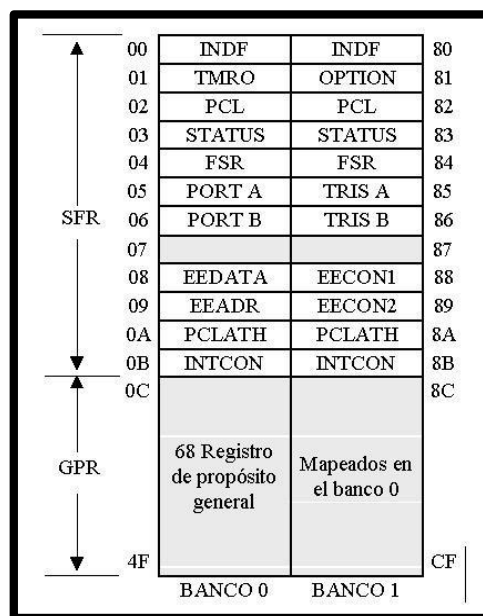


Figura 11. Organización de la memoria RAM del PIC16F84A

SFR: Registros de propósito específico.

GPR: Registros de propósito general.

Los registros están organizados como dos arreglos (bancos) de 128 posiciones de 8 bits cada una (128 x 8); todas las posiciones se pueden acceder directa o indirectamente (esta última a través del registro selector FSR). Para seleccionar que banco de registros se trabaja en un momento determinado se utiliza el bit RP0 del registro ESTADO (STATUS).

Los registros del SFR (Registros de función especial) se localizan en la memoria de

programa de la posición 00h a la posición 0Bh, mientras que, los registros GPR (registros de propósito general) se localizan de la posición 0Ch a la posición 4Fh.

2.7.2.1 REGISTROS DE PROPÓSITO GENERAL (GPR)

También son conocidos como archivos de registro de propósito general (GPR). Todos los dispositivos PIC's tienen alguna área de Registros de Propósito General. Cada GPR tiene un ancho de 8 bits y es accedido directamente o indirectamente a través del FSR.

Los registros de propósito general (GPR) se utilizan para poder guardar alguna información o datos de manera temporal, pero como se trata de una memoria de tipo RAM estática, estos datos se perderán cuando se guarde algún nuevo dato o cuando se corte la alimentación o al momento de un reset.

2.7.2.2 REGISTRO DE PROPÓSITO ESPECIAL (SFR)

Los Registros de la Función Especiales son usados por el CPU y periféricos para controlar el funcionamiento del dispositivo. Estos registros son del tipo RAM estática. Los registros de función especial (SFR) son registros que ya tienen una función determinada para el microcontrolador, por ese motivo el usuario debe tener cuidado al utilizarlos, mediante la utilización de estos registros el usuario determinará el funcionamiento del microcontrolador y los periféricos del mismo.

En la siguiente Tabla 6 se muestra la organización de los registros SFR del PIC16F84A. Se indica la dirección, el banco, el nombre de registro y cada uno de los bits.

Tabla 6. Organización detallada de los registros del PIC16F84A

Dir	Nombre		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		Banco 0								
00h	INDF		Usa el contenido de FSR para el direccionamiento indirecto							
01h	TMR0		Temporizador/contador en tiempo real de 8 bits							
02h	PCL		Contador de programa de 8 bits (PC)							
03h	STATUS		IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
04h	FSR		Direccionamiento indirecto de memoria con INDF							
05h	PORTA					RA4/T0CK	RA3	RA2	RA1	RA0
06h	PORTB		RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT
07h			No implementado. Se lee como "0"							
08h	EEDATA		Registro de datos de EEPROM							
09h	EEDR		Registro de direcciones de EEPROM							
0Ah	PCLATH					5 bits superiores del PC				
0Bh	INTCON		GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
		Banco 1								
80h	INDF		Usa el contenido de FSR para el direccionamiento indirecto							
81h	OPTION		RBPV	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
82h	PCL		Direccionamiento indirecto de memoria							
83h	STATUS		IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
84h	FSR		Direccionamiento indirecto de memoria							
85h	TRISA					Dato de la dirección del registro PORTA				
86h	TRISB		Dato de la dirección del registro PORTB							
87h			No implementado. Se lee como "0"							
88h	EECON1					EEIF	WRERR	WREN	WR	RD
89h	EECON2		Registro de control EEPROM (no es un registro físico)							
0Ah	PCLATH					Buffer de escritura para de 5 bits superiores del PC				
0Bh	INTCON		GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

2.7.2.3 REGISTRO ESTADO (STATUS) 03H

El registro STATUS se ubica en la posición 03h de la memoria de datos del PIC16F84. Contiene el estado aritmético de la ALU, la causa del reset y los bits de preselección de los bancos de memoria de datos. Como con cualquier registro, el registro de STATUS puede estar destinado para cualquier instrucción.

Los bits Z, DC y C del registro ESTADO, son señalizadores que indican una operación realizada por la ALU, por ejemplo, si el resultado de una operación es cero, entonces el señalizador de Z se pondrá en alto (1).

Los bits 5 y 6 (RP0 y RP1) son los bits de selección de banco para el direccionamiento directo de la memoria de datos; solamente RP0 se usa en los PIC16F84. RP1 se puede utilizar como un bit de propósito general de lectura/escritura.

Los bits TO y PD no se pueden modificar por un proceso de escritura, ya que solo muestran la condición por la cual se ocasiona el último reset.

Por ejemplo: CLRF STATUS limpiará los tres bits superiores del registro ESTADO y pondrá en uno el bit Z. Esto dejara el registro de ESTADO como 000u u1uu (donde u=inalterado).

Sólo las instrucciones BCF, BSF, SWAPF MOVLW, MOVF y MOVWF pueden usarse para alterar el registro ESTADO.

Los bits IRP y RP1 (ESTADO <7:6>) no son usados por los PIC16F8X y deben ser programados como cero. El uso de estos bits como bits de propósito general no se recomienda, debido a que esto puede afectar la compatibilidad con futuros productos.

El registro ESTADO (STATUS) se ubica en la dirección 03H del banco 0 y en la dirección 83h del banco 1 de la memoria de datos en la SFR (registros de propósito específico).

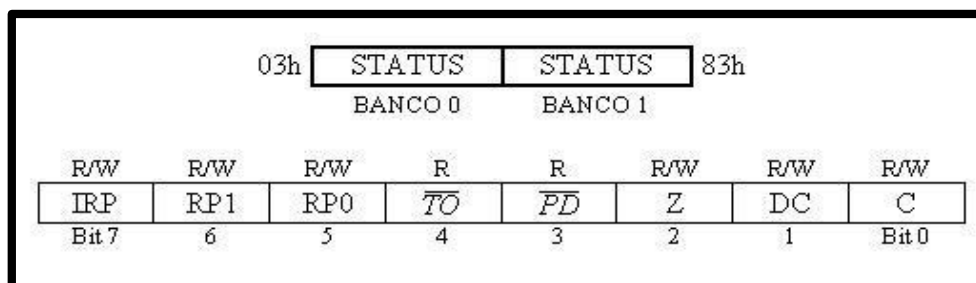


Figura 12. Dirección y mapeo de bits del registro ESTADO (STATUS)

A continuación, se describen los bits del registro ESTADO que se muestran en la Figura 12.

IRP: bit de selección de banco (usado para direccionamiento indirecto).

0 = Banco 0, 1 (00h - FFh)

1 = Banco 2, 3 (100h - 1FFh)

Este bit no se utiliza efectivamente en el PIC16F84, debe programarse como 0.

RP1: RP0: Selector de banco (usado para direccionamiento directo).

00 = Banco 0 (00h - 7Fh)

01 = Banco 1 (80h - FFh)

10 = Banco 2 (100h - 17Fh)

11 = Banco 3 (180h - 1FFh)

Solamente RP0 se utiliza en el PIC16F84, RP1 se debe programar como cero.

$\overline{T0}$: Time Out (Bit de finalización del temporizador)

1 = después de energizarse, de instrucción CLRWD, o instrucción SLEEP

0 = Cuando el circuito de vigilancia Watchdog finaliza la temporización.

$\overline{PD}$: Power-down (Bit de bajo consumo)

1 = Se pone automáticamente en 1 después de la conexión de la alimentación al microcontrolador o al ejecutar la instrucción clwdd.

0 = Se pone automáticamente en 0 mediante la ejecución de la instrucción sleep.

Z: Zero (Bit de cero)

1 = El resultado de una operación lógica o aritmética es cero.

0 = El resultado de una operación lógica o aritmética no es cero.

DC: Digit Carry/Borrow (Bit de acarreo de dígito)

1 = En operaciones aritméticas se activa cuando hay acarreo entre el bit 3 y 4

0 = No hay acarreo entre el bit 3 y 4

C: Carry (bit de acarreo)

1 = Indica que se ha producido un acarreo en el bit de más peso del resultado al ejecutar las instrucciones de ADDWF y ADDLW.

0 = No se ha producido acarreo.

Registro OPTION (dirección 81h)

El registro OPTION es un registro legible y escribible que contiene varios bits para configurar: los temporizadores TMR0/WDT, las interrupciones INT y TMR0 y la activación de las resistencias de pull-ups del puerto B, además de asignar el valor de preescala con la que va actuar el Temporizador asignado.

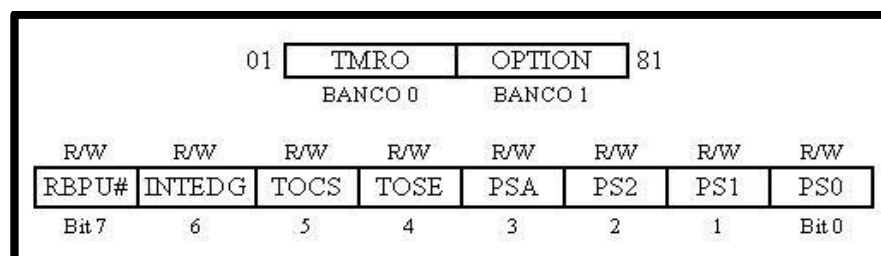


Figura 13. Dirección y mapeo de bits del registro OPTION y TMR0

A continuación, se describen los bits del registro OPTION/TMR0 que se muestran en la Figura 13.

RBPU: Resistencia Pull-Up Puerto B

1 = Desactivadas

0 = Activadas

INTEDG: Flanco activo interrupción externa

1 = Flanco ascendente

0 = Flanco descendente

TOCS: Tipo de reloj para el TMR0

1 = Pulsos introducidos a través de T0CKI (contador)

0 = Pulsos de reloj interno Fosc/4 (temporizador)

TOSE: Tipo de flanco en T0CKI

1 = Incremento de TMR0 cada flanco descendente

0 = Incremento de TMR0 cada flanco ascendente

PSA: Asignación del Divisor de frecuencia

1 = El Divisor de frecuencia se le asigna al WDT

0 = El Divisor de frecuencia se le asigna al TMR0

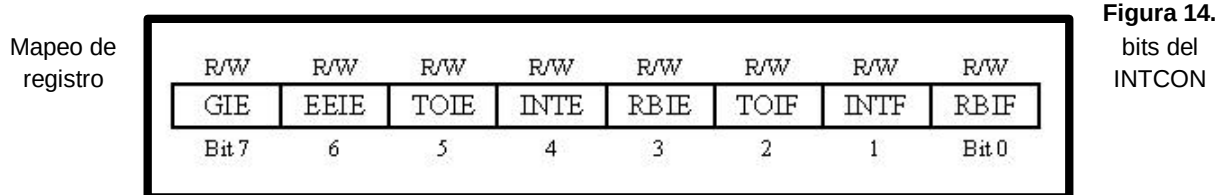
PS2-PS0: Prescaler Value (Valor con el que actúa el Divisor de frecuencia que se muestra en la Tabla 7).

Tabla 7. Valor de preescala asignado al divisor de frecuencia

PS2	PS1	PS0	Divisor del TMR0	Divisor del WDT
0	0	0	1 : 2	1 : 1
0	0	1	1 : 4	1 : 2
0	1	0	1 : 8	1 : 4
0	1	1	1 : 16	1 : 8
1	0	0	1 : 32	1 : 16

2.7.2.4 REGISTRO INTCON

El registro INTCON se localiza en la dirección 0Bh y en la dirección 8Bh de la memoria de datos de la SFR, es un registro de lectura / escritura que contiene los bits de habilitación de las fuentes de interrupción. Las banderas de interrupción se ponen en alto cuando una interrupción ocurre, sin tener en cuenta el estado de su correspondiente bit de habilitación o el permiso global de interrupciones, GIE (INTCON<7>).



A

continuación, se describen los bits del registro INTCON que se muestran en la Figura 14.

GIE: Habilitador global de interrupciones (Global Interrupt Enable)

1 = Habilita todas las interrupciones

0 = Deshabilita todas las interrupciones

EEIE: Habilitación de interrupción por escritura de la EEPROM (EEPROM Write Interrupt Enable)

1 = Habilita la interrupción EEIE

0 = Deshabilita la interrupción EEIE

TOIE: Habilitación de la interrupción del temporizador TMRO (TMR0 Interrupt Enable)

1 = Habilita la interrupción TOIE

0 = Deshabilita la interrupción TOIE

INTE: Habilitación de la interrupción INT (INT Interrupt Enable)

1 = Habilita la interrupción INTE

0 = Deshabilita la interrupción INTE

RBIF: Habilitación de la interrupción RBIF (RBIF Interrupt Enable)

1 = Habilita la interrupción RBIF

0 = Deshabilita la interrupción RBIF

TOIF: Bandera de interrupción por sobreflujo del TMR0 (TMR0 Overflow Interrupt Flag)

1 = Ha ocurrido un sobreflujo en el TMR0 (solo es limpiado por software)

0 = No hay sobreflujo en el TMR0

INTF: Bandera de interrupción INT (RB0/INT Interrupt Flag)

1 = Ha ocurrido la interrupción externa RB0/INT

0 = No hay interrupción externa por RB0/INT

RBIF: Bandera de interrupción por cambio en el puerto B (RBPort Change Interrupt Flag)

1 = Indica el cambio de estado de los bits RB7:RB4 del puerto B (se limpia por software)

0 = No hay cambios de estado en los bits RB7:RB4 del puerto B.

2.7.2.5 CONTADOR DE PROGRAMA (PCL Y PCLATH)

El contador de programa es un registro que nos indica la dirección de la instrucción próxima a ejecutar, este registro se puede leer y escribir, dicho registro se localiza en la posición 02h del banco 0 y en la 82 del banco 1 de la memoria de datos (RAM).

Al igual que todos los registros específicos que controlan la actividad del procesador, el Contador de Programa está implementado sobre un par de posiciones de la memoria RAM. Cuando se escribe el Contador de Programa como resultado de una operación de la ALU, los 8 bits de menos peso del PC residen en el registro PCL,

que se encuentra también en la posición 2 de los dos bancos de memoria de datos. Los bits de más peso, PC<12:8>, residen en los 5 bits de menos peso del registro PCLATH, que ocupa la posición 0Ah de los dos bancos de la memoria RAM. En las instrucciones GOTO y CALL de la gama media, los 11 bits de menos peso del PC provienen del código de la instrucción y los otros dos de los bits PCLATH <4:3>, como se muestra en la siguiente Figura 15.

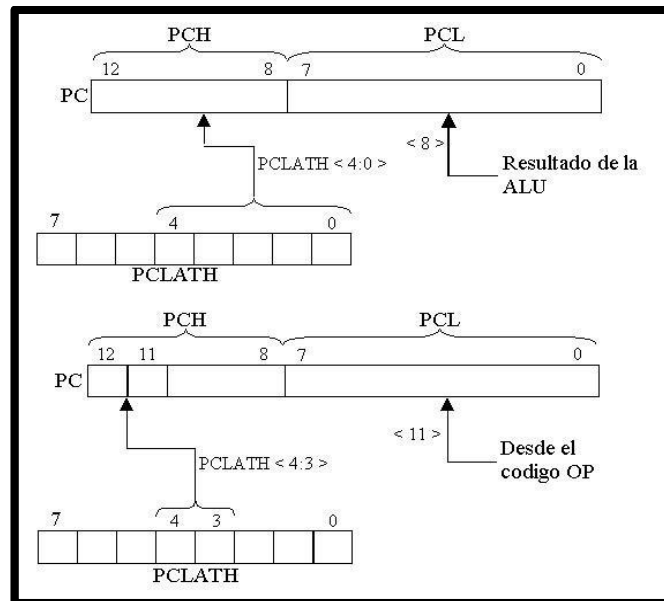


Figura 15. Carga del contador de programa

En la parte superior de la figura 15 se muestra cómo se carga el PC cuando una instrucción deposita en él el resultado que se obtiene de la ALU: en la parte inferior, se indica la carga del PC en las instrucciones GOTO y CALL.

Con los 11 bits que se cargan en el PC desde el código de las instrucciones GOTO y CALL se puede direccionar una página de 2 K de la memoria. Los bits restantes PC <12:11> tienen la misión de apuntar una de las 4 páginas del mapa de memoria y en los modelos de PIC que alcanzan ese tamaño, proceden de PCLATH <4:3>.

La pila es una zona aislada de las memorias de instrucciones y datos. Tiene una estructura LIFO, en la que el último valor guardado es el primero que sale. Tiene 8

niveles de profundidad cada uno con 13 bits. Funciona como un «buffer» circular, de manera que el valor que se obtiene al realizar el noveno «desempilado» (pop) es igual al que se obtuvo en el primero.

La instrucción CALL y las interrupciones originan la carga del contenido del PC en el nivel superior o «cima» de la pila. El contenido del nivel superior se saca de la Pila al ejecutarse las instrucciones RETURN, RETLW y RETFIE. El contenido del registro PCLATH no es afectado por la entrada o salida de información de la pila.

A diferencia de los PIC de primera generación, el 16F84 ante una condición de reset inicia el contador de programa con todos sus bits en “cero”. Durante la ejecución normal del programa, y dado que todas las instrucciones ocupan sólo una posición de memoria, el contador se incrementa en uno con cada instrucción, a menos que se trate de alguna instrucción de salto.

2.7.3 DIRECCIONAMIENTO DE LA MEMORIA DE DATOS RAM

La memoria de datos RAM de los PIC de la Gama media está organizada en cuatro bancos de 128 posiciones con 8 Bits de longitud cada una. En el caso concreto del PIC16F84 sólo están implementadas las 80 primeras posiciones (00h - 4Fh) de los bancos 0 y 1, como se puede apreciar en la siguiente Figura 16.

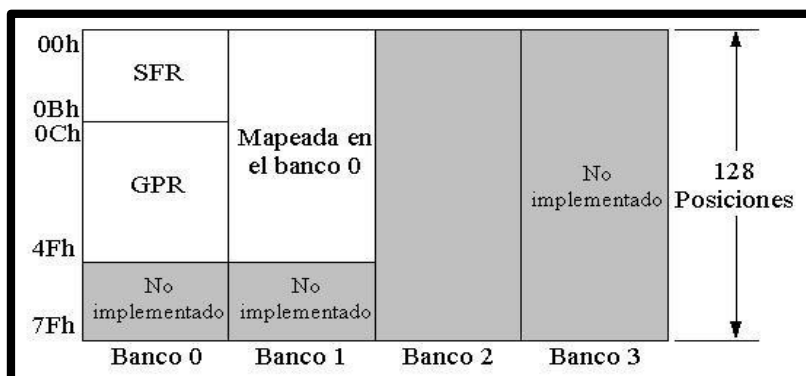


Figura 16. Mapeo general de bancos de los dispositivos PIC

Para direccionar una posición de la memoria RAM en un PIC hay que especificar, en primer lugar, el banco en el que se encuentra y, después, la dirección relativa en el

mismo. Para seleccionar el banco existen dos bits en el registro STATUS del área de la RAM, que se llaman RP0 y RP1, ocupando las posiciones 5 y 6 de dicho registro. La forma de direccionar la RAM, es decir, el modo de direccionamiento, puede tomar dos alternativas: direccionamiento directo y direccionamiento indirecto.

2.7.3.1 DIRECCIONAMIENTO DIRECTO

Es la manera de apuntar a la posición de la memoria RAM donde se encuentra el dato que se busca, se empieza escribiendo en los bits RP0 y RP1 del registro STATUS el código correspondiente al banco. La dirección relativa dentro del banco está incluida en el código de la propia instrucción, pues de los 14 bits de longitud, los 7 de menor peso se reservan para esta labor.

Para cargar el código del banco en los bits RP0 y RP1, existen, un par de instrucciones muy eficaces, capaces de poner a 1 o 0 un bit de cualquier registro.

En el PIC16F84 el direccionamiento directo es aún más sencillo, pues la memoria RAM sólo está implementada sobre los dos primeros bancos, el banco 0 y el banco 1. Esto supone que con un sólo bit se elija el banco, concretamente el bit usado es el RP0, siendo indiferente el estado que tenga el bit RP1. Si RP0 = 0, se accede al banco 0 y si RP0 = 1 se accede al banco 1, como se muestra en la siguiente Figura 17.

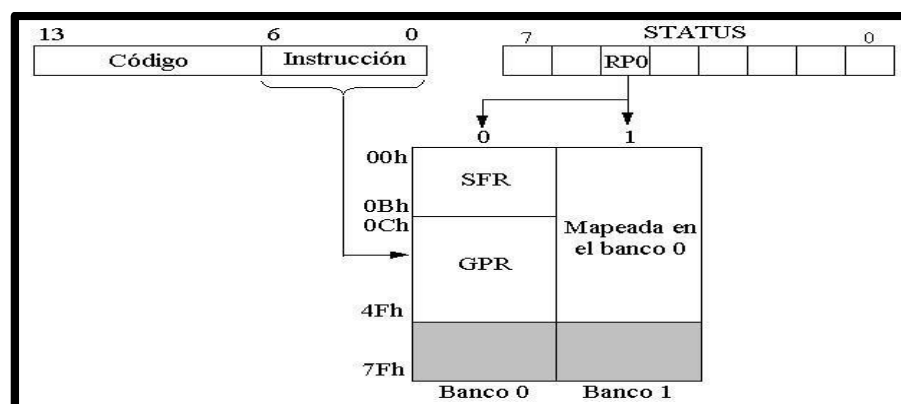


Figura 17. Direccionamiento directo en el PIC16F84

2.7.3.2 DIRECCIONAMIENTO INDIRECTO

En los microcontroladores PIC las instrucciones que manejan el direccionamiento

indirecto emplean el registro INDF, que ocupa la dirección 00H de los dos bancos del área SFR. En realidad, el registro INDF no está implementado físicamente, por lo tanto, cuando se hace referencia a él, en realidad se accede a otro registro específico del área SFR que se llama FSR. En los 7 bits de menos peso del FSR se encuentra el valor de la dirección relativa del registro buscado en el banco seleccionado.

La instrucción con modo de direccionamiento indirecto utiliza como registro operando el INDF, pero el PIC accede al contenido del registro FSR, que ocupa, por duplicado, la dirección 04h de los dos bancos de la RAM.

No sólo es diferente la forma de obtener los 7 bits de la dirección, sino que en el modo indirecto también cambia el modo de especificar el banco. En el direccionamiento directo se usaban los bits RP0 y RP1 para elegir el banco; en el modo indirecto se usa el bit 7, de más peso, del registro FSR (recuerde que los 7 bits de menos peso contenían la dirección relativa). Como en el PIC16F84 sólo están disponibles los bancos 0 y 1, el bit IRP del registro STATUS siempre valdrá 0, justo como se observa en la siguiente Figura 18.

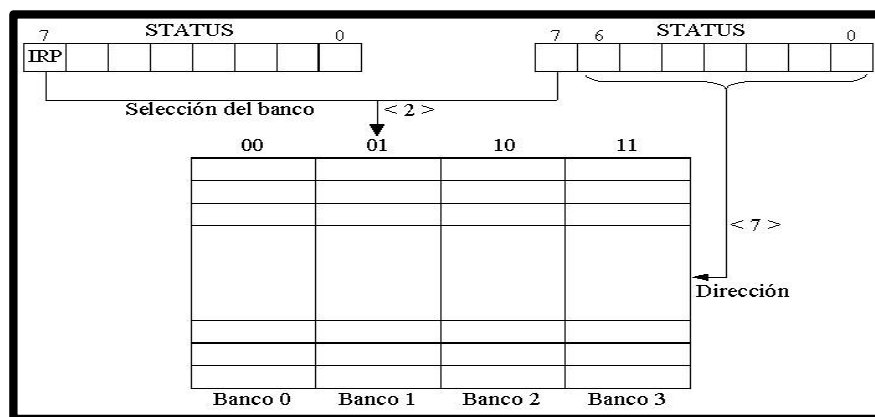


Figura 18. Direccionamiento indirecto en los dispositivos PIC

El direccionamiento indirecto es muy utilizado en el manejo de tablas, pues nos facilita la lectura y la escritura de la misma, dicho procedimiento consiste en cargar el registro FSR con un valor inicial, que nos va a servir de offset para después incrementar o decrementar el mismo y de esta manera acceder a una posición específica de memoria.

2.7.4 LA PILA

Una pila o stack es un conjunto de registros que guardan información de una forma determinada, ya que tanto la carga como la descarga son especial. En el caso del PIC16F84 la pila está formada por 8 registros de 13 bits cada uno y dotados de una estructura de carga/descarga tipo LIFO, que significa “último en entrar, primero en salir”. Cuando se carga información a la pila siempre se introduce por el nivel 1. Si anteriormente había información en el nivel 1, se traslada al nivel 2 y lo de éste al nivel 3 y así sucesivamente. Es decir, cuando se carga información en el nivel 1 lo que había almacenado en cada nivel se desplaza al siguiente, dicho proceso se puede apreciar en la figura 19.

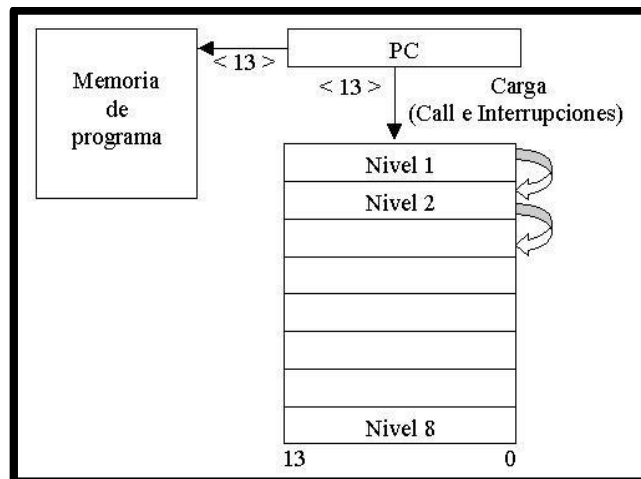


Figura 19. La instrucción CALL y la interrupción, provocan la carga automática del contenido del PC en la pila

Si la pila está llena y contiene información en los ocho niveles, la carga de un nuevo dato produce el desplazamiento de la información de cada nivel al siguiente, perdiéndose la que existía en el nivel 8. Para recuperar la información de la pila el desplazamiento de la información se realiza al revés. Sale la que está ocupando el nivel 1 y a éste se le carga con lo del nivel 2. La información se desplaza al nivel anterior y el dato que sale desde el nivel 1 habrá sido el último que se había cargado, dicho proceso se muestra en la siguiente figura 20.

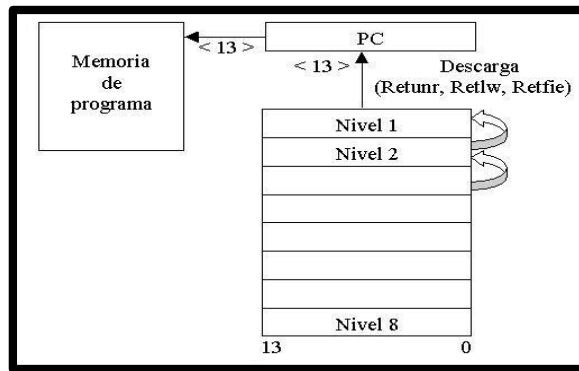


Figura 20. Las instrucciones de RETORNO de subrutina o de interrupción descargan automáticamente el contenido del nivel 1 de la pila sobre el PC

La pila del PIC funciona automáticamente y se halla relacionada directamente con el contenido del contador de programa (PC). La pila se carga en su nivel 1 con el contenido del PC. Cuando la pila descarga la información del nivel 1 pasa al PC. Cada vez que se ejecuta una instrucción Call o que se produce una interrupción, es necesario salvar el contenido del PC para luego saber retornar al punto donde se salió en el programa principal. La ejecución de una instrucción Call o una interrupción producen el traslado automático del contenido del PC al nivel 1 de la Pila.

La última instrucción de una subrutina o de una rutina de interrupción es la de retorno (Return), que al ejecutarse produce la descarga automática del nivel 1 de la pila sobre el PC, devolviendo el flujo de control al programa principal en el punto que se abandonó.

2.8 LOS PUERTOS DE ENTRADA/SALIDA

Los PIC16X8X sólo disponen de dos Puertos de E/S. El Puerto A posee 5 líneas RA0-RA4, y una de ellas soporta dos funciones multiplexadas. Se trata de la RA4/T0CK1, que puede actuar como línea de E/S o como terminal por la que se reciben los impulsos que debe contar TMR0. El Puerto B tiene 8 líneas, RB0-RB7, y también tiene una con funciones multiplexadas, la RB0/INT, que además de línea típica de E/S, también sirve como terminal por la que se reciben los impulsos externos que provocan una interrupción.

Cada línea de E/S puede configurarse independientemente como entrada o como salida, según se ponga a 1 o a 0, respectivamente el bit asociado del registro de configuración de cada Puerto (TRISA y TRISB). Se llaman PUERTO A y PUERTO B a los registros que guardan la información que entra o sale por el Puerto, y ocupan las direcciones 5 y 6 del banco 0 de la memoria de datos. Los registros de configuración TRISA y TRISB ocupan las mismas direcciones, pero en el banco 1, como se aprecia en la siguiente Figura 21.

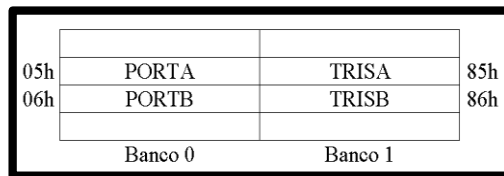


Figura 21. Mapeo de direcciones de los puertos

La información que entra o sale por los puertos reside en las posiciones 5 y 6 del banco 0, mientras que TRISA y TRISB, que son los registros de configuración, ocupan las direcciones 5 y 6 del banco 1.

2.8.1 CONSIDERACIONES DE LOS PUERTOS

Como el PIC16F84 es de tecnología CMOS. Todos los pines deben estar conectados a alguna parte, nunca dejarlos al aire porque se puede dañar el integrado. Los pines que no se estén usando se deben conectar a la fuente de alimentación de +5V, como se muestra en la siguiente Figura 22.

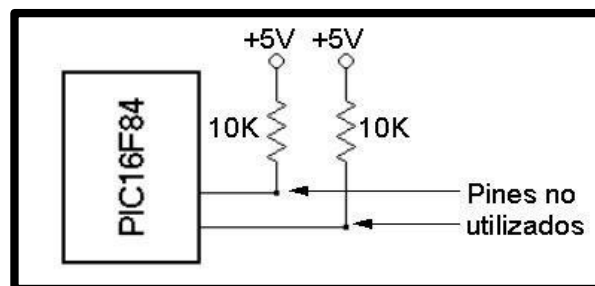


Figura 22. Los puertos no utilizados se deben conectar a la fuente

La máxima capacidad de corriente de cada uno de los pines de los puertos en modo sumidero (sink) es de 25mA y en modo fuente (source) es de 20mA. La máxima capacidad de corriente total de los puertos se muestra en la siguiente Figura 23.

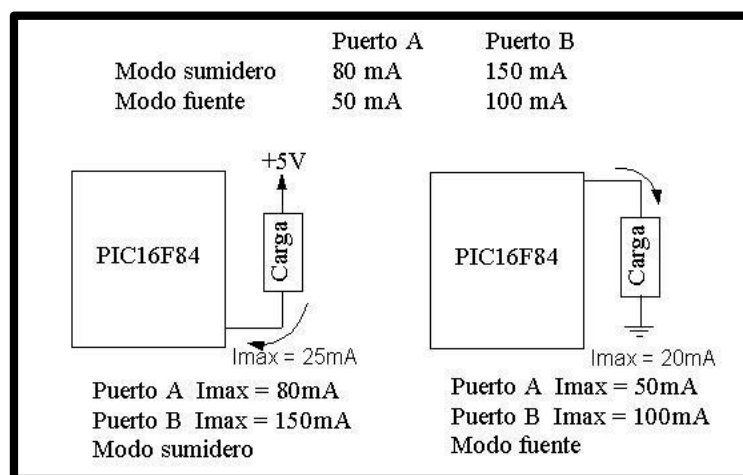


Figura 23. Capacidad máxima de corriente que soportan los puertos

El consumo de corriente del microcontrolador para su funcionamiento depende del voltaje de operación, la frecuencia y de las cargas que tengan sus pines. Para un reloj de 4 MHz el consumo es de aproximadamente 2mA; aunque este se puede reducir a 40 microamperios cuando se está en el modo sleep (en este modo el microcontrolador se detiene y disminuye el consumo de potencia).

2.8.2 PUERTO A (REGISTROS PORTA Y TRISA)

Las líneas RA3-RA0 admiten niveles de entrada TTL y de salida CMOS. La línea RA4/TOCK1 dispone de un circuito Disparador Schmith (Schmith Trigger) que proporciona una buena inmunidad al ruido y la salida tiene drenador abierto. RA4 multiplexa su función de E/S con la de entrada de impulsos externos para el TMR0.

En la Figura 24 se muestra el registro TRISA, el cual se encarga de configurar las líneas del Puerto A como entradas si están a 1 y como salidas si están a 0. En el circuito de la Figura 25 se muestra la adaptación de los pines RA3-RA0 a las señales del procesador.

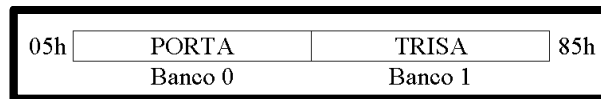


Figura 24. Ubicación de PORTA y TRISA en los bancos del PIC

Cuando se lee una línea de él Puerto A se toma el nivel lógico que tiene en ese momento. Las líneas cuando actúan como salidas están «sujetadas», lo que significa que sus pines sacan el nivel lógico que se haya cargado por última vez en el registro PUERTO A. La escritura de un Puerto implica la operación «lectura/modificación/escritura». Primero se lee el Puerto, luego se modifica el valor y finalmente se escribe en el «latch» de salida.

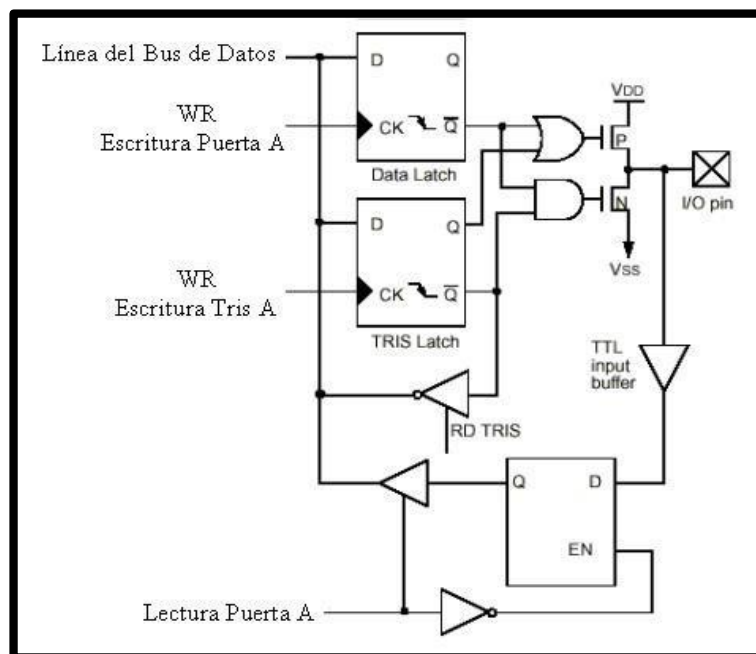


Figura 25. Diagrama de conexiones de los pines RA3-RA0 a las señales del procesador

De la Figura 25 se desprende que cuando se saca un nivel lógico por una línea del Puerto A, primero se deposita en la línea correspondiente del bus interno de datos y se activa la señal WRITE, lo que origina el almacenamiento de dicho nivel en el Flip-Flop de datos. En esta situación, el FF de configuración debería contener un 0 para que actuase como salida. Con estos valores el puerto OR tendría un 0 en su salida y la AND también. Estos valores producen la conducción del transistor PMOS superior y el bloqueo del NMOS. Así, la terminal de E/S queda conectada a la VDD y tiene un nivel alto. Como la línea de salida está sujeta conserva su valor hasta que no es reescrita en el FF tipo D.

Si una línea actúa como entrada, el nivel lógico depositado en ella desde el exterior pasa a la línea correspondiente del bus interno de datos cuando se activa la señal READ y se hace conductor el dispositivo triestado que les une. Al programarse como entrada, los dos transistores MOS de salida quedan bloqueados y la línea en alta impedancia.

Así, que cuando se lee una línea de entrada se obtiene el estado actual que tiene su terminal correspondiente y no el valor que haya almacenado en el FF de datos. La información presente en una línea de entrada se muestrea al iniciarse el ciclo de instrucción y debe mantenerse estable durante su desarrollo.

Al iniciarse el PIC todos los bits de los registros TRIS quedan a 1, con lo que las líneas de los Puertos quedan configuradas como entradas.

Cada línea de salida puede suministrar una corriente máxima de 20 mA y si es entrada puede absorber hasta 25 mA. Al existir una limitación en la disipación máxima de la potencia del chip se restringe la corriente máxima de absorción del Puerto A 80 mA y la de suministro a 50 mA. El Puerto B puede absorber un máximo de 150 mA y suministrar un total de 100 mA.

Con `movf puerto, w` se lee un puerto y con la instrucción `movwf puerto` se escribe.

También existen instrucciones para modificar el valor de un bit particular correspondiente a una línea de un puerto con las instrucciones bsf puerto, bit (pone a 1 el bit indicado de el puerto) y bcf puerto, bit (pone a cero el bit indicado). Existen instrucciones de salto condicionales que verifican el valor de un bit de un puerto y brincan si vale 1 (btfss) o si vale 0 (btfsc).

En la siguiente Tabla se muestra una descripción de cada uno de los pines del puerto A.

Tabla 8. Descripción de las terminales del puerto A

Nombre	Bit	Tipo de Buffer	Función
RA0	Bit 0	TTL	Entrada / salida
RA1	Bit 1	TTL	Entrada / salida
RA2	Bit 2	TTL	Entrada / salida
RA3	Bit 3	TTL	Entrada / salida
RA4/TOCK1	Bit 4	ST	Entrada / salida o entrada de reloj externa para el TMR0. La salida es de tipo drenaje abierto.
TTL = entrada TTL, ST = entrada Schmitt Trigger.			

2.8.3 PUERTO B (REGISTROS PORTB Y TRISB)

Consta de 8 líneas bidireccionales de E/S, RB7-RB0, cuya información se almacena en el registro PORTB, que ocupa la dirección 6 del banco 0, tal como se muestra en la figura 26. El registro de configuración TRISB ocupa la misma dirección en el banco 1. La línea RB0/INT tiene dos funciones multiplexadas. Además de pines de E/S, actúa como terminal para la repetición de una interrupción externa, cuando se autoriza esta función mediante la adecuada programación del registro INTCON, del que se hablará más adelante.

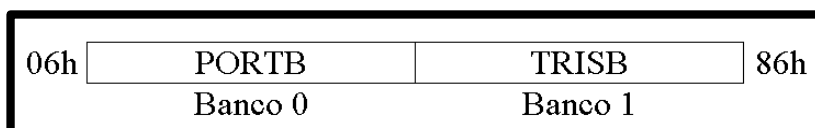


Figura 26. Ubicación de PORTB y TRISB en los bancos del PIC

A todas las líneas de este puerto se les permite conectar una resistencia Pull-Up de elevado valor con el positivo de la alimentación. Para este fin hay que programar en el registro OPTION el bit RBP_U# = 0, afectando la conexión de la resistencia a todas las líneas. Con el Reset todas las líneas quedan configuradas como entradas y se desactivan las resistencias Pull-Up.

Las 4 líneas de más peso, RB7-RB4, pueden programarse para soportar una misión especial. Cuando las 4 líneas actúan como entradas se les puede programar para generar una interrupción si alguna de ellas cambia su estado lógico. Esta posibilidad es muy práctica en el control de teclados.

En la figura 27 se muestra el esquema de conexionado entre los pines RB7-RB4 y las líneas correspondientes del bus de datos interno.

El estado de los pines RB7-RB4 en modo entrada se compara con el valor antiguo que tenían y que se había sujetado durante la última lectura del Puerto B. El cambio de estado en alguna de esas líneas origina una interrupción y la activación del señalizador RBIF.

La línea RB6 también se utiliza para la grabación serie de la memoria de programa y sirve para soportar la señal de reloj. La línea RB7 constituye la entrada de los datos en serie.

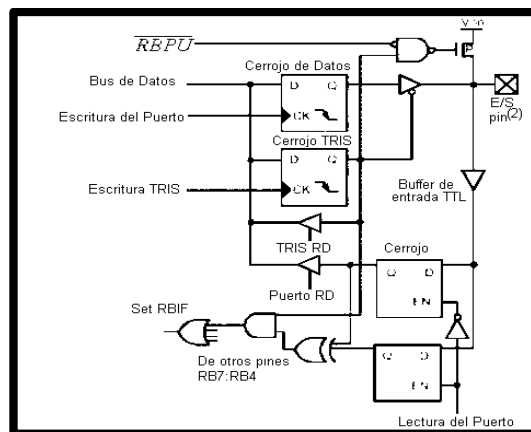


Figura 27. Conexionado de los pines RB7-RB4 y las líneas correspondientes al bus interno de datos y las señales de control

Los pines RB3 al RB0 pueden ser programados como líneas de entrada o salida, dependiendo de su configuración en el registro TRISB. El conexionado de dichos pines se muestra en la siguiente Figura 28.

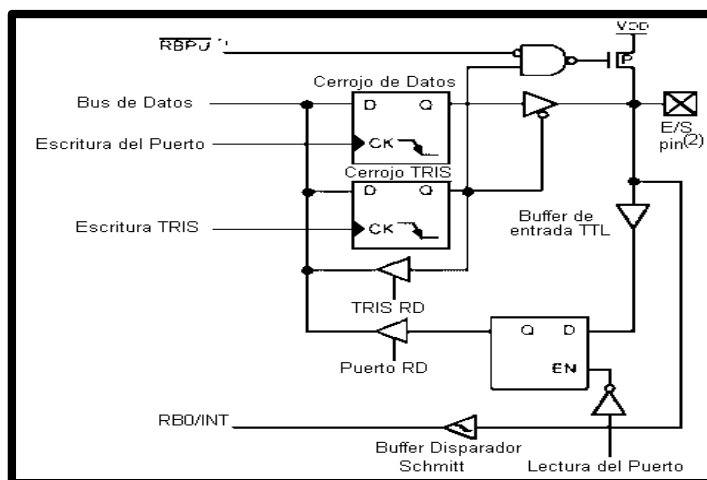


Figura 28. Conexionado de los pines RB3 – RB0 y las líneas correspondientes al bus interno de datos y las señales de control

En la siguiente Tabla se muestra una descripción de cada uno de los pines del puerto B.

Tabla 9. Descripción de las terminales del puerto B

Nombre	Bit	Tipo de Buffer	Función
RB0/INT	Bit 0	TTL/ST	Entrada / salida o entrada de interrupción externa. Resistencias de Pull-Up por software
RB1	Bit 1	TTL	Entrada / salida (Resistencia de Pull-Up por software)
RB2	Bit 2	TTL	Entrada / salida (Resistencias de Pull-Up por software)
RB3	Bit 3	TTL	Entrada / salida (Resistencias de Pull-Up por software)
RB4	Bit 4	TTL	Entrada/ salida (con interrupción por cambio) (Resistencias de Pull-Up por software)

RB5	Bit 5	TTL	Entrada / salida (con interrupción por cambio) (Resistencias de Pull-Up por software)
RB6	Bit 6	TTL/ST	Entrada / salida (con interrupción por cambio) (Resistencias de Pull-Up por software)
RB7	Bit 7	TTL/ST	Entrada / salida (con interrupción por cambio) (Resistencias de Pull-Up por software)
TTL = entrada TTL, ST = entrada Schmitt Trigger.			

2.9 MÓDULO TEMPORIZADOR/CONTADOR TMR0

Una de las tareas de los programas de control de dispositivos es determinar intervalos de tiempo al elemento encargado de realizar esta función se le denomina temporizador (timer). También suele ser frecuente contar los impulsos que se producen en el exterior del sistema y el elemento destinado a este fin se denomina contador.

Si las tareas del temporizador o contador se asignarán al programa principal robarían mucho tiempo al procesador. Por este motivo se diseñan recursos específicamente orientados a estas misiones.

Los PIC16X8X poseen un temporizador/contador de 8 bits, llamado TMR0, que actúa de dos maneras diferentes. Como contador de sucesos, que están representados por los impulsos que se aplican a la terminal RA4/T0CKI. Al llegar al valor FFh se

desborda el contador y con el siguiente impulso, pasa a 00h, activando un señalizador y/o provocando una interrupción. Como temporizador, cuando se carga en el registro que implementa al recurso un valor inicial se incrementa con cada ciclo de instrucción ($F_{osc}/4$) hasta que se desborda, o sea, pasa de FFh a 00h y avisa poniendo a 1 un bit (bandera) y/o provocando una interrupción.

El temporizador se carga con un valor inicial que se va incrementando hasta el desbordamiento. Como contador va incrementando su valor con cada pulso que se le aplica. Al alcanzar el máximo valor binario se «desborda» y pasa a cero, circunstancia que indica una bandera.

Para que el TMR0 funcione como contador de impulsos aplicados a la terminal TOCK1 hay que poner a 1 el bit T0CS, que es el que ocupa la posición 5 del registro OPTION. En esta situación, el registro TMR0 que es el ubicado en la dirección 1 del banco 0 de la memoria de datos, se incrementa con cada flanco activo aplicado en la terminal TOCK1.

El tipo de flanco activo se elige programando el bit T0SE, que es el que ocupa la posición 4 del registro OPTION, si T0SE = 1, el flanco activo es el descendente, y si T0SE = 0, es el ascendente. Cuando se desea que TMR0 funcione como temporizador el bit T0CS = 0. En el diagrama a bloques de la Figura 29 se describe gráficamente el funcionamiento del TMR0.

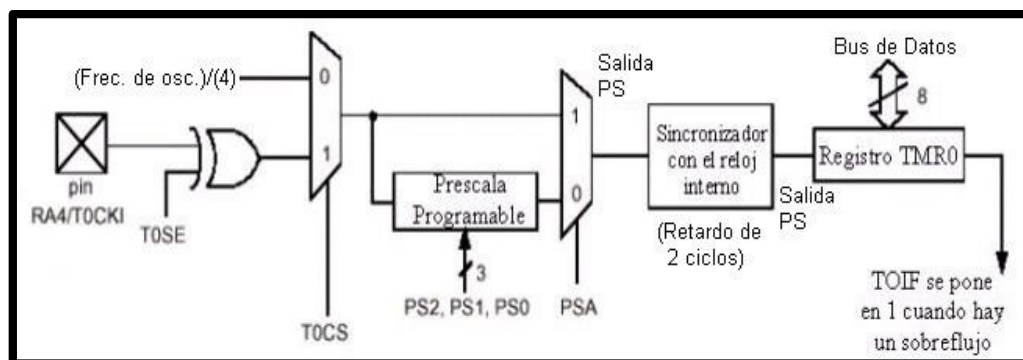


Figura 29. Diagrama simplificador del temporizador TMR0

En realidad, los PIC16X8X y los de la gama baja disponen de dos temporizadores, el TMR0 y el Perro Guardián (watchdog). El primero actúa como principal y sobre él recae el control de tiempos y el conteo de pulsos. El otro vigila que el programa no se «cicle», y para ello cada cierto tiempo comprueba si el programa se está ejecutando normalmente. En caso contrario, si el control está detenido en un bucle infinito a la espera de algún acontecimiento que no se produce, el perro guardián «interrumpe», lo que se traduce en un reset que reinicializa todo el sistema.

A menudo el TMR0 y el Perro Guardián precisan controlar largos intervalos de tiempo y necesitan aumentar la duración de los impulsos de reloj que les incrementa. Para cubrir este requisito se dispone de un circuito programable denominado divisor de frecuencia, que divide la frecuencia utilizada por diversos rangos.

Para programar el comportamiento del TMR0, el Perro Guardián (WDT) y el divisor de frecuencia se utilizan algunos bits del registro OPTION y de la palabra de configuración, que se aplicará más adelante. En la Figura 30 se proporciona un esquema simplificado de la arquitectura del circuito de control de tiempos usado en los PIC16X8X.

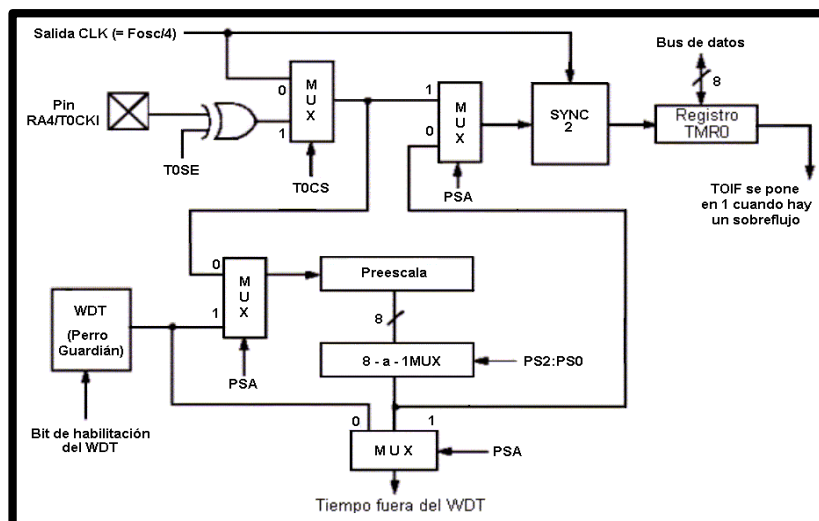


Figura 30. Esquema simplificado del circuito de control de tiempos usado en los PIC16X8X

El divisor de frecuencia puede usarse con el TMR0 o con el WDT. Con el TMR0 actúa como pre-divisor, es decir, los impulsos pasan primero por el divisor y luego se

aplican al TMR0, una vez aumentada su duración. Con el perro guardián actúa después, realizando la función de Post-divisor.

Los pulsos, que divide por un rango el divisor de frecuencia, pueden provenir de la señal de reloj interna ($F_{osc}/4$) o de los que se aplican a la terminal T0CK1.

El TMR0 se comporta como un registro de propósito especial (SFR) ubicado en la dirección 1 del banco 0 de la memoria de datos. En igual dirección, pero en el banco 1, se halla el registro OPTION. TMR0 puede ser leído y escrito en cualquier momento al estar conectado al bus de datos. Funciona como un contador ascendente de 8 bits.

Cuando funciona como temporizador conviene cargarle con el valor de los impulsos que se quiere temporizar, pero expresados en complemento a 2. De esta manera, al llegar el número de impulsos deseados se desborda y al pasar por 00h se activa el señalizador TOIF y/o se produce una interrupción.

Para calcular los tiempos a controlar con TMR0, se utilizan las siguientes fórmulas prácticas.

Temporización = $4 \cdot T_{osc} \cdot (\text{valor cargado en TMR0}) \cdot (\text{rango del divisor})$

Valor a cargar en a TMR0 = $(\text{Temporización} / 4) \cdot T_{osc} \cdot \text{rango del divisor}$

En cualquier momento se puede leer el valor que contiene TMR0, sin detener su conteo. La instrucción adecuada al caso es `movf tmr0, w`.

En la Figura 31 se muestra un esquema de funcionamiento del TMR0. Obsérvese que hay un bloque que retrasa 2 ciclos el conteo para sincronizar el momento del incremento producido por la señal aplicada en T0CK1 con el que producen los impulsos externos de reloj. Cuando se escribe TMR0 se retrasa 2 ciclos su incremento y se pone a 0 el divisor de frecuencia.

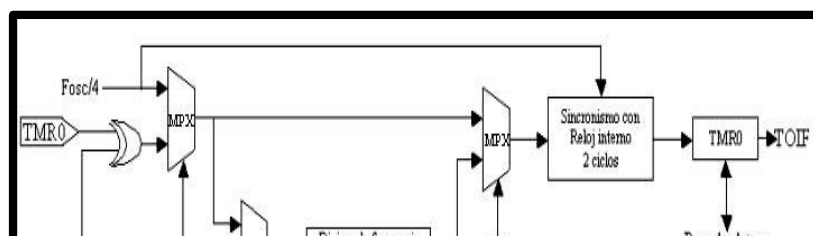


Figura 31. Esquema general del funcionamiento del TMR0

La misión principal del registro OPTION es controlar TMR0 y el Divisor de frecuencia. Ocupa la posición 81h de la memoria de datos, que equivale a la dirección 1 del banco 1. El bit T0CS (Timer 0 Clock Edge Select) selecciona en el multiplexor MPX1 la procedencia de los impulsos de reloj, que pueden ser los del oscilador interno ($F_{osc}/4$) o los que se aplican desde el exterior por la terminal T0CK1. El bit T0SE (Timer 0 Clock Source Select) elige el tipo de flanco activo en los impulsos externos. Si T0SE = 1 el flanco activo es el descendente y si T0SE = 0 el flanco es ascendente.

El bit PSA del registro OPTION asigna el divisor de frecuencia al TMR0 (PSA = 0) o al WDT (PSA = 1). Los 3 bits de menos peso de OPTION seleccionan el rango por el que divide el Divisor de frecuencia los pulsos que se le aplican en su entrada. La figura 32 muestra la distribución de los bits del registro OPTION.

El bit 6 INTEDG (Interrupt Edge) sirve para determinar el flanco activo que provocará una interrupción externa al aplicarse a la terminal RB0/INT. Un 1 si es ascendente y un 0 descendente.

El bit 7 RBPU# (RB Pull-Up) activa, si sale 0 o desactiva, cuando vale 1, las resistencias Pull-Up que pueden conectarse en las líneas del puerto B.

2.9.1 REGISTRO OPTION

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
RBPU#	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
Bit 7	6	5	4	3	2	1	Bit 0

Figura 32. Mapeo de bits del registro OPTION

A continuación, se describen los bits del registro OPTION que se muestran en la Figura 32.

RBPU: Resistencia Pull-Up Puerto B.

1=Desactivadas

0 = Activadas

INTEDG: Flanco activo interrupción externa

1 = Flanco ascendente

0 = Flanco descendente

TOCS: Tipo de reloj para el TMR0

1 = Pulsos introducidos a través de T0CKI (contador)

0 = Pulsos de reloj interno Fosc/4 (temporizador)

TOSE: Tipo de flanco en T0CKI

1 = Incremento de TMR0 cada flanco descendente

0 = Incremento de TMR0 cada flanco ascendente

PSA: Asignación del Divisor de frecuencia

1 = El Divisor de frecuencia se le asigna al WDT

0 = El Divisor de frecuencia se le asigna al TMR0

PS2-PS0: Prescaler Value (Valor con el que actúa el Divisor de frecuencia, el cual se muestra en la Tabla 10)

Tabla 10. Distribución y asignación de funciones de los bits del registro OPTION

PS2	PS1	PS0	Divisor del TIMER 0	Divisor del WDT
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

2.10 MEMORIA EEPROM

El microcontrolador PIC16F84 tiene implementada una memoria de datos de tipo EEPROM, con una capacidad de 64 posiciones de 8 bits cada una. Su principal inconveniente es que la duración del ciclo de escritura o borrado de una posición es muy lento, comparado con la velocidad del procesador, ya que puede llegar a los 10 milisegundos, en los que un PIC16F84, trabaja a 10 Mhz, ejecuta más instrucciones que las 1024 que caben en su memoria de programa. Además, este tiempo es crítico y hay que esperar a que termine completamente la operación para iniciar otra nueva.

Los ciclos de lectura y escritura de la EEPROM se realizan sobre la posición de un byte. El ciclo de escritura conlleva el borrado previo y automático de la información que contenía la posición accedida. El tiempo del ciclo de escritura está controlado por un temporizador integrado en el chip. El gran problema es que dicho tiempo no sólo es largo, unos 10ms como máximo, sino que es variable según la temperatura y el voltaje aplicado. Como este tiempo es crítico, el final de la operación está controlado por un señalizador (EEIF), que puede causar una interrupción al completarse la escritura de una posición.

Si con los bits correspondientes de la palabra de configuración se protege el código

del microcontrolador, se puede continuar leyendo y escribiendo la EEPROM.

2.10.1 MAPA DE MEMORIA EEPROM

Aunque la EEPROM es una parte de la memoria de datos, sus posiciones no están mapeadas junto a los registros SFR y GPR de la RAM, que es volátil. Ocupa un espacio de memoria independiente, tal y como se muestra en la siguiente Figura 33.

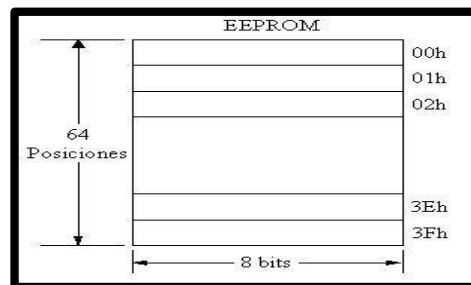


Figura 33. Mapeo de memoria EEPROM del PIC16F84

2.10.2 MANEJO DE LA EEPROM

Para manipular la lectura y escritura de las 64 posiciones de la EEPROM del PIC16F84 se emplean cuatro registros del área de SFR:

- EECON1
- EECON2
- EEDATA
- EEADR

Para manejar el espacio de la EEPROM hay que utilizar dos registros de control. Se trata de los registros EEADR y EEDATA. El registro EEADR ocupa la posición 9 del banco 0 de la RAM, mientras que EEDATA se ubica en la dirección 8 del banco 0, como se muestra gráficamente en la siguiente Figura 34.

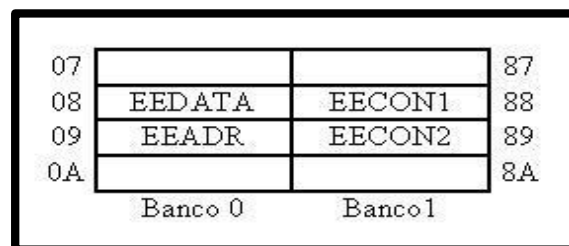


Figura 34. Ubicación de los registros EEDATA, EEADR, EECON1 y EECON2

En la Figura 34 se muestra la ubicación de los registros EEADR y EEDATA que se emplean en el manejo del espacio de memoria de la EEPROM. También aparecen los registros de control EECON1 y EECON2.

Ocupando una posición correlativa a EEDATA, pero en el banco 1, se encuentra el registro de control de la EEPROM, EECON1. De manera similar, el registro de control EECON2 ocupa una posición correlativa a EEADR, en el banco 1.

Para acceder a una posición de la EEPROM hay que cargar en el registro EEADR la dirección de la misma, mientras que en EEDATA se carga el dato a escribir, o bien, se deposita el dato leído en él.

Como los registros EEADR y EEDATA son de 8 bits, los dos bits de más peso del registro EEADR siempre valen cero, ya que el mayor valor que se puede cargar es 3Fh.

En la posición relativa 8 del banco 1 de la zona SFR de la RAM está situado el registro EECON1, cuyos bits tiene la misión de controlar las operaciones de la EEPROM. Los tres bits de más peso de este registro no tienen asignada ninguna función.

Los bits que controlan la lectura y escritura son RD y WR respectivamente. Estos bits no pueden limpiarse, solo ponerse en alto por software. Estos bits se limpian por hardware después del proceso de lectura y escritura.

En la siguiente Figura 35 se describe cada uno de los bits del registro EECON1.

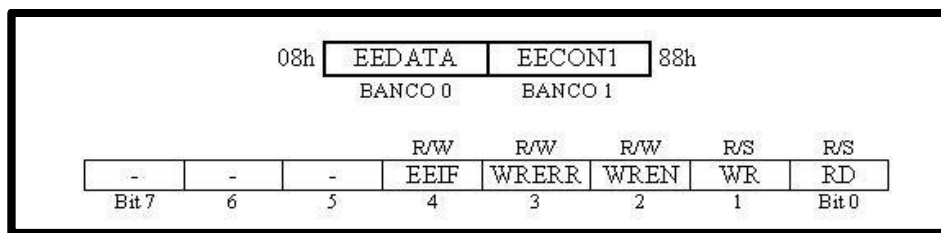


Figura 35. Bits correspondientes al registro EECON1

Los bits 5, 6 y 7 no están implementados, son leídos como '0'.

A continuación se describe la función de cada uno de los bits que integran el registro EECON1.

EEIF: Bandera de interrupción de la operación de escritura de la EEPROM

1 = El proceso de escritura se ha completado (debe ser limpiado por software).

0 = El proceso de escritura no se ha completado o no ha empezado

WRERR: Bandera de error de escritura en la EEPROM

1 = El proceso de escritura ha terminado prematuramente.

0 = La operación de escritura se ha completado correctamente.

WREN: Permiso de escritura de la EEPROM

1 = Permite la escritura de la EEPROM

0 = Inhabilita la escritura en la EEPROM

WR: Bit de control de escritura

1 = Se inicializa el ciclo de escritura (El bit es limpiado por hardware una vez completada la escritura). El bit solo puede ponerse en 1 (no en cero) por software).

0 = Se ha completado el ciclo de escritura de la EEPROM. (Cuando se completa el ciclo pasa '0' automáticamente).

RD: Bit de control de lectura

1 = Inicializa el ciclo de lectura de la EEPROM (Toma un ciclo de lectura. RD se limpia por hardware). El bit RD solo puede ponerse en '1' (no puede limpiarse) por software.

0 = No se ha iniciado la lectura de la EEPROM (cuando se ha completado el ciclo de escritura de la EEPROM este bit se pone automáticamente en cero).

El registro de control EECON2 en realidad no está físicamente implementado en la

RAM. Se utiliza como un dispositivo de seguridad para validar la operación de escritura de la EEPROM y evitar interferencias que pudieran producirse en el largo intervalo de tiempo que dura esta operación. Si se lee todos sus bits valen 0.

2.10.3 OPERACIÓN DE LECTURA EN LA EEPROM

La lectura de una de las posiciones de la EEPROM comienza cargando en EEADR la dirección a acceder. Luego se pone a 1 el bit RD del registro EECON1. En el siguiente ciclo el dato leído estará disponible en el registro EEDATA. Dicho dato permanecerá en este registro hasta que se realice otra nueva operación de lectura o escritura en la EEPROM.

2.10.4 OPERACIÓN DE ESCRITURA EN LA EEPROM

Para escribir un dato en una posición de la EEPROM hay que seguir una secuencia de instrucciones en la que participa de una manera especial el registro EECON2. Este registro, que en realidad no está implementado físicamente, sólo asume una función de seguridad al cargarse con los valores 55h y AAh, sucesivamente. Esto evita interferencias durante la larga operación de escritura, que a veces llega a los 10 ms.

El ciclo comienza cargando en EEADR la dirección de la posición y en EEDATA el byte a escribir. Si durante el proceso de escritura no se desea admitir interrupciones hay que poner el bit de permiso global de interrupciones GIE = 0. Así se evita que durante la operación de escritura la CPU se desvíe a otra rutina si ocurre una interrupción.

Después hay que autorizar la escritura poniendo el bit WREN = 1 en el registro EECON1. A continuación hay que incluir en el programa una secuencia de instrucciones que cargan en primer lugar el valor de 55h en EECON2 y luego AAh en el mismo registro. Una vez ejecutada esta secuencia se pone el bit WR = 1 en EECON1 y se inicia la larga operación de escritura.

Cuando finaliza la escritura el bit WR es igual a 0 y el señalizador EEIF a 1 en el registro EECON1. Este señalizador se pone a 1 automáticamente en cuanto finaliza

el ciclo de escritura, y leyendo puede saberse si el acontecimiento ha sucedido. Luego debe ser el programa el que pase a 0 el bit EEIF.

Detrás de cada ciclo de escritura es recomendable comprobar que la posición de la EEPROM se ha grabado correctamente.

2.11 INTERRUPCIONES

Las interrupciones constituyen el procedimiento más rápido y eficaz del microcontrolador para el tratamiento de los acontecimientos del mundo exterior. Si algún parámetro o circunstancia tiene influencia en el desarrollo del programa, la interrupción es el medio por el cual el microcontrolador lo atiende inmediatamente. Al programa especial que trata dicho acontecimiento se le llama “rutina de interrupción”.

Las interrupciones son desviaciones del flujo de control del programa originadas asincrónicamente por diversos sucesos que no se hallan bajo la supervisión de las instrucciones. Dichos sucesos pueden ser externos al sistema, como la generación de un flanco o nivel activo en una terminal del microcontrolador, o bien internos, como el desbordamiento de un contador.

El comportamiento del microcontrolador ante la interrupción es similar al de la instrucción CALL de llamada a subrutina. Cuando se utiliza la instrucción CALL, se detiene la ejecución del programa en curso, se guarda la dirección actual del PC en la pila y éste se carga con una nueva dirección que se especifica en dicha instrucción.

En el caso de una interrupción, el proceso inicia de la misma forma; se detiene la carga con una dirección «reservada» de la memoria de programa (0004h), llamada Vector de Interrupción.

En los PIC16X8X el Vector de interrupción se halla situado en la dirección 0004h, en donde comienza la Rutina de Servicio a la Interrupción (RSI). En general, en dicho

Vector se suele colocar una instrucción de salto incondicional (GOTO), que traslada el flujo de control a la zona de la memoria de código destinada a contener la rutina de atención a la interrupción.

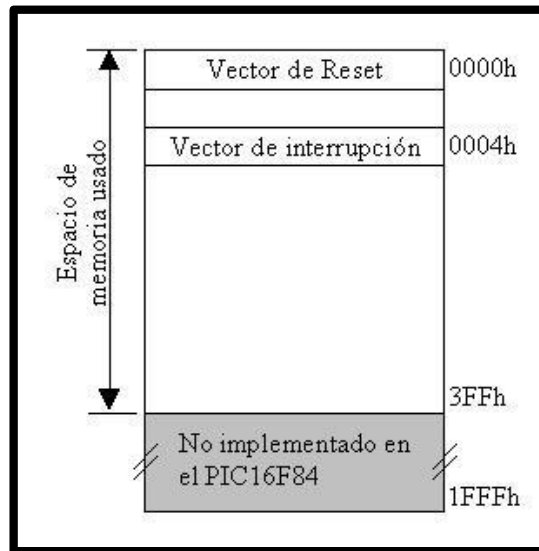


Figura 36. Localización del vector de interrupción

En la Figura 36, se muestra la dirección del vector de interrupción que está localizada en la posición 0004h de la memoria de programa del PIC16F84.

La RSI suele comenzar guardando en la memoria de datos algunos registros específicos del procesador. Concretamente aquellos que la RSI va a emplear y va a alterar su contenido. Antes del retorno al programa principal se recuperan los vectores guardados y se restaura completamente el estado del procesador.

Algunos procesadores salvan estos registros en la pila, pero los PIC no disponen de instrucciones para introducir (push) y extraer (pop) información de la pila, utilizando para este fin registros de propósito general de la memoria de datos.

Los PIC 16X8X pueden ser interrumpidos por 4 causas diferentes, pero todas ellas desvían el flujo de control a la dirección 0004h, por lo que otra de las operaciones iniciales de la RSI es averiguar cuál de las posibles causas ha sido la responsable de la interrupción en curso. Para ello se exploran los señalizadores de las fuentes de interrupción.

Otro detalle importante de la RSI de los PIC 16X8X es que estos microcontroladores poseen un bit GIE (Global Interrupt Enable) que cuando vale 0 inhabilita todas las interrupciones. Pues bien, al comenzar la RSI dicho bit GIE se pone automáticamente a 0, con objeto de no atender nuevas interrupciones hasta que se termine la que ha comenzado. En el retorno final de la interrupción, GIE pasa a valer automáticamente 1 para volver a tener en cuenta las interrupciones. Dicho retorno de interrupción se realiza mediante la instrucción RETFIE. Antes del retorno conviene borrar el señalizador de la causa de interrupciones que se ha atendido, porque si bien los señalizadores se ponen a 1 automáticamente en cuanto se produce la causa que indican, la puesta a 0 se hace por programa, la Figura 37 muestra un organigrama de las fases más importantes que se desarrollan durante el proceso de ejecución de una interrupción.

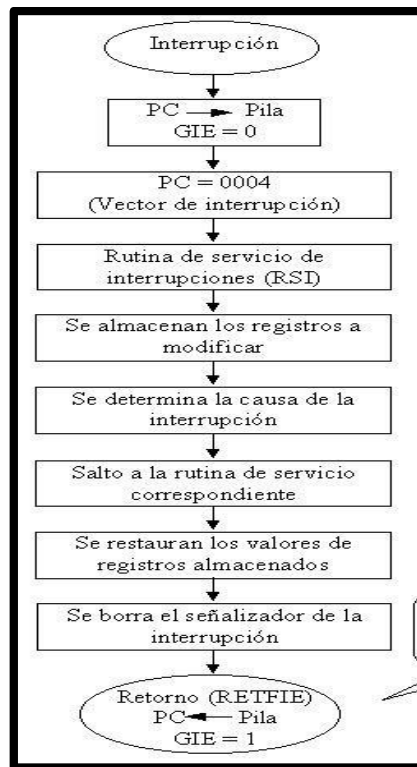


Figura 37. Organigrama del desarrollo de una interrupción, teniendo en cuenta el papel del bit GIE

2.11.1 CAUSAS DE UNA INTERRUPCIÓN

Hay cuatro causas que pueden ocasionar una interrupción del PIC16F84

- Activación del pin RB0/INT
- Desbordamiento del temporizador TMR0
- Cambio de estado de uno de los 4 pines de más peso del PUERTO B (RB7:RB4)
- Finalización de la escritura en la EEPROM de datos

Una dificultad inicial que presentan estas interrupciones es que sea cual sea la que ha originado la interrupción, la rutina de atención siempre empieza en la dirección del vector de interrupción que es la 0004h.

Otros procesadores disponen de un vector de interrupción para cada causa que la produce. Con el PIC16F84 no queda otro remedio que comenzar averiguando cuál ha sido la causa de la interrupción, probando los señalizadores correspondientes, y desviando hacia el procedimiento particular que atiende a cada una.

Cuando ocurre cualquiera de los 4 sucesos indicados, se origina una petición de interrupción, que si se acepta y se atiende comienza depositando el valor del PC actual en la pila, poniendo el bit GIE = 0 y cargando en el PC el valor 0004h, que es el vector de interrupción donde se desvía el flujo de control.

Cada fuente de interrupción dispone de un señalizador o flag, que es un bit que se pone automáticamente a 1 cuando se produce. Además, cada fuente de interrupción tiene otro bit de permiso, que según su valor permite o prohíbe la realización de dicha interrupción.

2.11.2 REGISTRO DE CONTROL DE INTERRUPCIONES INTCON

A veces, muchos programas no precisan la ayuda de las interrupciones y no las usan. Por este motivo debe existir una llave general que permita o prohíba las interrupciones en su conjunto. En el PIC16F84 existe un registro de control en el área SFR de la RAM, llamado INTCON, que regula el funcionamiento de las

interrupciones, la dirección que ocupa dicho registro es la 0Bh del banco 0 y la dirección 8Bh del banco 1, justo como se muestra en la figura 38. El bit más significativo de INTCON (INTCON<7>) se denomina GIE (Permiso Global de Interrupciones). Si se escribe este bit y se pone GIE = 0, todas las interrupciones están prohibidas y el procesador no admite ninguna. En cambio, si escribimos GIE = 1 el procesador admite cualquier interrupción que se produzca y que el programador haya habilitado para que actúe. Es decir, cada interrupción dispone de un bit particular de permiso que también debe habilitarse si se quiere que cuando la causa de la interrupción se produzca el procesador lance las fases de su desarrollo.

El registro INTCON es un registro de lectura/escritura que contiene los bits de habilitación de las fuentes de interrupción. Las banderas de interrupción se ponen en alto cuando una interrupción ocurre, sin tener en cuenta el estado de su correspondiente bit de habilitación o el permiso global de interrupciones, GIE (INTCON<7>).

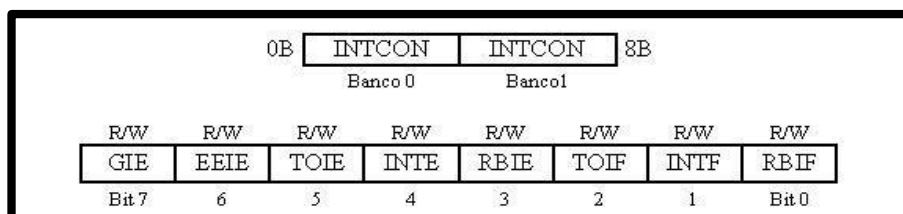


Figura 38. Bits que conforman el registro INTCON

A continuación, se describen los bits que conforman el registro INTCON y que se muestran en la Figura 38.

GIE: Habilitador global de interrupciones (Global Interrupt Enable)

1 = Habilita todas las interrupciones

0 = Deshabilita todas las interrupciones

EEIE: Habilitación de interrupción por escritura de la EEPROM (EEPROM Write Interrupt Enable)

1 = Habilita la interrupción EEIE

0 = Deshabilita la interrupción EEIE

TOIE: Habilitación de la interrupción del temporizador TMRO (TMRO Interrupt Enable)

1 = Habilita la interrupción TOIE

0 = Deshabilita la interrupción TOIE

INTE: Habilitación de la interrupción INT (INT interrupt Enable)

1 = Habilita la interrupción INTE

0 = Deshabilita la interrupción INTE

RBIE: Habilitación de la interrupción RBIF (RBIF Interrupt Enable)

1 = Habilita la interrupción RBIE

0 = Deshabilita la interrupción RBIE

TOIF: Bandera de interrupción por sobreflujo del TMRO (TMRO Overflow Interrupt Flag)

1 = Ha ocurrido un sobreflujo en el TMRO (solo es limpiado por software)

0 = No hay sobreflujo en el TMRO

INTF: Bandera de interrupción INT (RB0/INT Interrupt Flag)

1 = Ha ocurrido la interrupción externa RB0/INT

0 = No hay interrupción externa por RB0/INT

RBIF: Bandera de interrupción por cambio en el puerto B (RB Port Change Interrupt Flag)

1 = Indica el cambio de estado de los bits RB7:RB4 del puerto B (se limpia por software)

0 = No hay cambios de estado en los bits RB7:RB4 del puerto B.

En el organigrama de la figura 39 se amplía la operatividad del desarrollo de la

interrupción al tener en cuenta el papel del bit GIE. Obsérvese que en cuanto se admite una interrupción el procesador pone automáticamente $GIE = 0$, para evitar que cuando está atendiéndose a una interrupción se produzcan otras nuevas.

Al completarse la rutina de atención a la interrupción y ejecutarse la última instrucción RETFIE, que es la de retorno de la rutina de interrupción, el bit GIE vuelve a tomar el valor 1 de manera automática, para que al retomar el programa principal queden habilitadas las interrupciones y puedan atenderse.

Cabe señalar que los señalizadores de interrupción deben ponerse a 0 por programa antes del retorno de la interrupción y son operativos, aunque la interrupción esté prohibida con su bit de permiso correspondiente. En la Figura 39 se ofrece el esquema de la lógica de control que origina la interrupción.

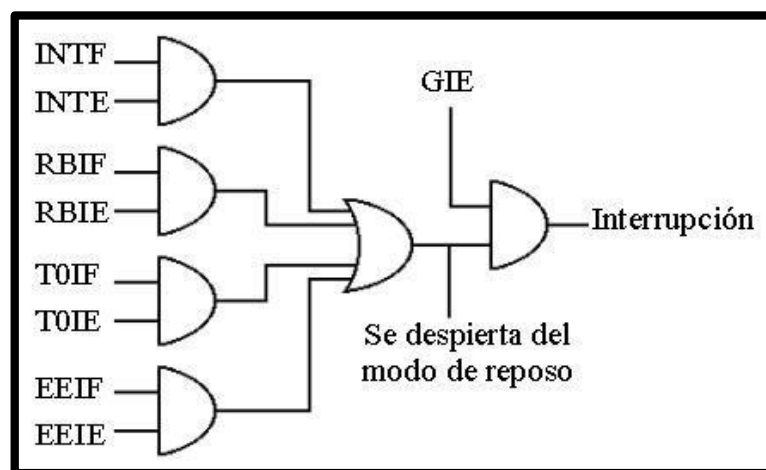


Figura 39. Lógica de control para la generación de una interrupción en los PIC16X8X

2.11.3 INTERRUPCIÓN EXTERNA INT

De las cuatro posibles causas de interrupción que admite el PIC16F84, hay una que le permite estar en contacto permanente con los acontecimientos que ocurren en el mundo exterior.

Esta interrupción está soportada por el pin 6 del microcontrolador, y se llama RB0/INT. Se trata de un pin multifunción que puede actuar de dos formas diferentes.

En primer lugar puede funcionar como línea de entrada/salida digital (RB0), también puede actuar como línea de petición de interrupción externa (INT).

Para que este pin pueda recibir las peticiones externas de interrupción hay que habilitarla para tal trabajo, para lo cual hay que comenzar poniendo a 1 el bit de permiso global de interrupciones GIE. Además, también habrá que habilitar esta interrupción poniendo a 1 el bit INTE. Tanto GIE como INTE se ubican en el registro INTCON.

Mediante el bit 6, llamado INTEG, del registro OPTION se puede seleccionar cuál será el flanco activo en RB0/INT. Si se desea en flanco ascendente se escribe un 1 en dicho bit, y si se desea que sea el flanco descendente se escribe un 0.

Dentro de la rutina de interrupción y antes de retornar al programa principal, el programador debe encargarse de poner el señalizador INTF = 0, pues de lo contrario, al ejecutarse la instrucción de retorno RETFIE volveríamos a repetir el proceso de interrupción, ya que el flag INTF se encontraría a 1.

2.11.4 INTERRUPCIÓN POR DESBORDAMIENTO DEL TMR0

El temporizador TMR0 es un contador ascendente de 8 bits. Cuando TMR0 se desborda y pasa del valor FFh al 00h, el señalizador T0IF se pone automáticamente a 1.

En el caso del TMR0 su valor se incrementa al ritmo de los impulsos de reloj que se le aplican por el pin RA4/T0CKI o desde el oscilador principal interno con la frecuencia $F_{osc}/4$. Cuando el valor del TMR0 pasa de FFh a 00h el señalizador T0IF, que es el bit 2 de INTCON, se pone a 1 automáticamente.

Para que el procesador sepa cuando se desborda el TMR0 debe explorar cada poco tiempo el estado del señalizador T0IF. El chequeo de dicho bit supone una constante atención y pérdida de rendimiento. Para conocer inmediatamente cuándo se ha

desbordado el TMR0 es mucho más eficaz generar una interrupción cuando ese hecho ocurra. Para ello es necesario habilitar la interrupción del temporizador, lo cual implica inicialmente poner el bit GIE = 1, que es el permiso global de interrupciones, y después activar el bit de permiso de la interrupción del TMR0, o sea, escribir un 1 en TOIE.

En la Figura 40 se muestra el diagrama a bloques de la interrupción por desbordamiento del temporizador TMR0.

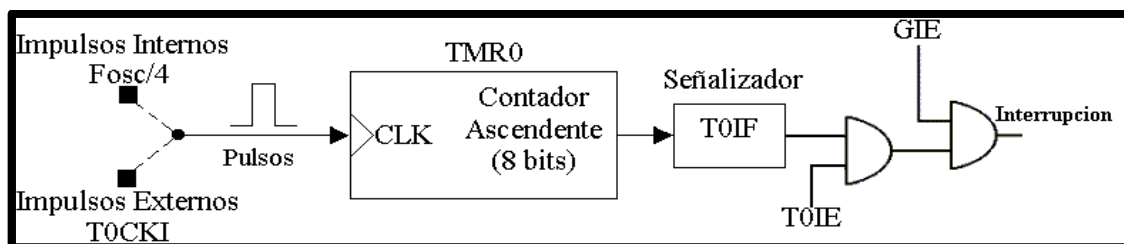


Figura 40. Diagrama a bloques de la Interrupción por desbordamiento del TMR0

Si no se recarga el TMR0 cuando se desborda, sigue contando desde 00h a FFh. En cualquier momento se puede leer y escribir este registro, pero cada vez que se escribe se pierden dos ciclos de reloj para la sincronización.

Cuando se carga inicialmente TMR0 con el valor de N_{10} , cuenta $256 - N$ impulsos, siendo el tiempo que tarda en hacerlo el que expresa la siguiente formula:

$$\text{Temporización} = 4 * T_{osc} * (256 - N_{10}) * \text{Rango del divisor de frecuencia}$$

Para entender lo anterior se propondrá un ejemplo:

Calcular el tiempo para que el TMR0 genere una interrupción si se le carga continuamente con el valor 4010, el divisor de frecuencia está programado en el rango 1:8 y el microcontrolador dispone de un cristal de 1 MHz.

$$T_{osc} = 1/1\text{MHz}$$

$$T_{osc} = 1\mu\text{s}$$

$$\text{Temporización} = 4 * T_{osc} * (256 - N_{10}) * \text{Rango del divisor de frecuencia}$$

$$\text{Temporización} = 4 * 1 \mu\text{s} * (256 - 40) * 8 = 6.912 \mu\text{s}$$

2.11.5 INTERRUPCIÓN POR CAMBIO DE ESTADO EN RB7:RB4

Una de las cuatro causas que producen la interrupción del PIC16F84 es el cambio del estado lógico en alguna de las cuatro líneas de más peso del puerto B (RB7:RB4).

Si en un momento una de las mencionadas líneas, que actúan como entradas digitales, cambia el estado lógico que recibe del mundo exterior, el señalizador RBIF se pone a 1, y si el bit de permiso de esta interrupción también está activo (RBIE = 1) se genera una interrupción, para que esta ocurra es necesario que el bit de permiso global de interrupciones, GIE valga 1, y además, que el bit de permiso particular de la interrupción por cambio de estado en RB7:RB4 (RBIE) también esté a 1.

La razón por la que existe esta extraña causa de interrupción es la de atender a los teclados matriciales, tan usuales en las aplicaciones con microcontroladores. El teclado es un periférico de entrada de información fácil de manejar, de reducido tamaño y economía, tal como se muestra en la siguiente Figura 41.

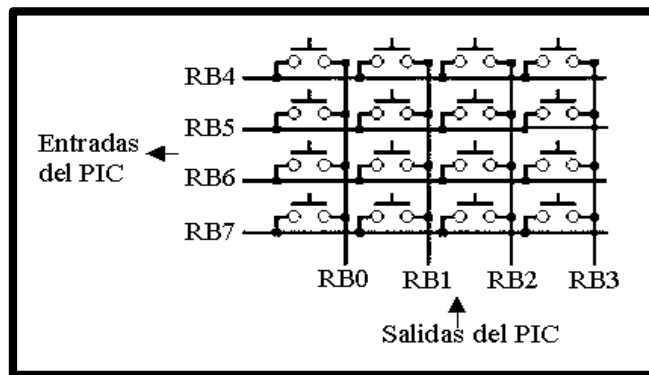


Figura 41. Estructura y conexionado del teclado matricial a los pines del PUERTO B del PIC

2.11.6 INTERRUPCIÓN POR FINALIZACIÓN DE LA ESCRITURA EN LA EEPROM

El tiempo típico que tarda en desarrollarse una operación de escritura en la EEPROM de datos de los PIC16X8X es de 10 ms, que es considerable comparado con la velocidad a la que el procesador ejecuta instrucciones. Para asegurarse que se ha completado la escritura y puede continuarse con el flujo de control del programa es aconsejable manejar la interrupción que se origina al finalizar la escritura, que pone automáticamente el señalizador EEIF a 1, y se autoriza siempre que los bits de permiso EEIE = 1 y GIE = 1.

Cuando se describió el proceso de escritura de la EEPROM de datos se indicó que se usaba un registro no real para asegurar la misma. Se trataba del EECON2, en él se grababan dos valores, el 55h y el AAh. Durante la escritura de este registro debe prohibirse la aceptación de interrupciones para salvaguardar la operación de escritura, por eso en ese módulo se pone GIE = 0.

En los PIC16C84 y PIC16F8X se puede leer y escribir la EEPROM de datos, aunque se haya protegido el código. En los PIC16CF8X, que disponen de memoria ROM para el código, existen dos bits para el código de protección: uno dedicado a la ROM de código y el otro a la EEPROM de datos.

2.12 EL PERRO GUARDIÁN (WDT)

El perro guardián, llamado abreviadamente WDT (Watchdog Timer) es un contador ascendente de 8 bits que tiene la propiedad especial de producir el Reset al microcontrolador cada vez que se desborda. A diferencia del TMR0, el Perro guardián no dispone de señalizadores, ni tampoco genera interrupción en el desbordamiento, lo único que hace es provocar un Reset que inicializa el procesador y comienza a ejecutar el programa principal desde la primera instrucción que se halla en la dirección 0000h de la memoria de programa (Vector Reset).

El objetivo del Perro Guardián es comprobar el correcto procesamiento de las instrucciones del programa. Si todo va bien, el programa proporciona, antes de

terminar su temporización, una instrucción que borra e inicia de nuevo el conteo, algo así como si se “refrescara”. Si el programa se ha colgado y no se han ejecutado las instrucciones según las previsiones, el programa no suministrará la instrucción de refresco, con lo que se desbordará el WDT y se generará un Reset. Con esta acción se pretende que el microcontrolador comience de nuevo el programa y evita la causa que lo bloqueó, por ejemplo, entra en bucle infinito, o porque precisa una señal para salir de él que no llega, etc.

Para evitar que se desborde el Perro Guardián hay que refrescarlo previamente. En realidad, este refresco consiste en ponerle a cero mediante las instrucciones `clrwdt` y `sleep`. El programador debe analizar las instrucciones de la tarea y situar alguna de esas dos en sitios estratégicos por los que pase el flujo de control antes de que transcurra el tiempo asignado al WDT. De esta manera si el programa se «cicla» no se refresca al Perro Guardián y se produce la reinicialización del sistema.

La instrucción `clrwdt` borra al WDT y reinicia su cuenta. Sin embargo, la instrucción `sleep`, además de borrar WDT detiene al sistema y lo mantiene en un estado de «reposo» o «de bajo consumo». Si no se desactiva el Perro Guardián al entrar en el modo de reposo, al completar su ciclo de espera provocará un Reset y sacará al microcontrolador del modo de bajo consumo. Para desactivar el Perro Guardián hay que escribir un 0 en el bit 2 (WDTE) de la Palabra de Configuración.

En el registro ESTADO existe un bit denominado $\overline{T0}$ que toma el valor de 0 después del desbordamiento del WDT.

El Perro Guardián se activa y se desactiva con el valor del bit WDTE de la palabra de configuración. La frecuencia de trabajo de este temporizador siempre es interna e independiente, originada por un oscilador RC propio. Cuando se asigna el divisor de frecuencia al Perro Guardián actúa como postdivisor, al revés de cómo actuaba con el TMR0. De esta manera, los impulsos que genera el WDT pasan al divisor de frecuencia que los divide por el rango que seleccionamos con los tres bits de menos

peso del registro OPTION.

Si el bit PSA es 1, en el registro OPTION, el Divisor de frecuencia se aplica al WDT. Es curioso apreciar que el Perro Guardián no dispone de ningún registro en el área SFR de la RAM. La razón se debe a que no se carga con ningún valor inicial, empieza desde 0 hasta desbordarse. Cada vez que se refresca con clrwdt vuelve a ponerse a 0 y a reiniciar el conteo. En el registro STATUS existe un bit llamado $\overline{TO}$ que toma el valor de 0 cuando se desborda el WDT, así discrimina la forma en que se ha realizado el Reset. En la figura 42 se muestra un diagrama a bloques que describe el funcionamiento del WDT.

El temporizador WDT se halla preparado para que controle un tiempo de 18ms, pero se puede actuar sobre el divisor de frecuencia para que dicho tiempo se eleve hasta un máximo de 2.3 segundos.

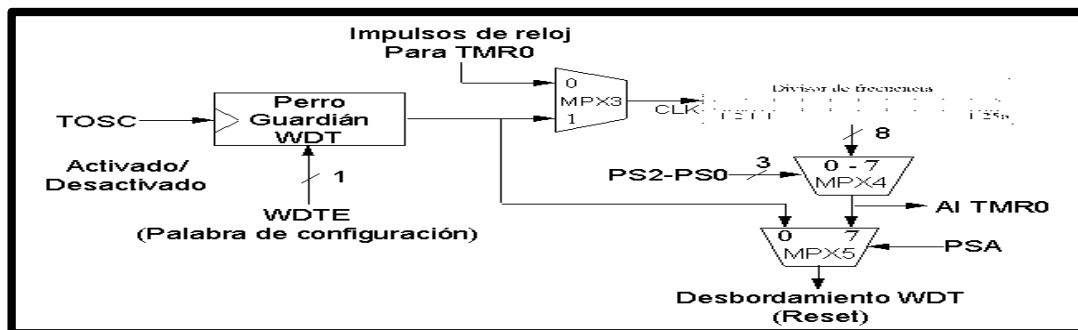


Figura 42. Diagrama a bloques de la operación del Perro guardián (WDT)

2.13 REINICIALIZACIÓN O RESET

Cuando se aplica un nivel bajo durante el tiempo suficiente al pin $\overline{MCLR}$, se produce la reinicialización, que conlleva la ejecución de dos acciones importantes.

- La puesta a cero del Contador de Programa, que así pasa a direccionar la primera instrucción del programa.
- Colocar en un determinado estado la mayoría de los bits de los registros de control del procesador.

En la Figura 43 se muestra un sencillo esquema, a base de un pulsador, que provoca

el Reset externo.

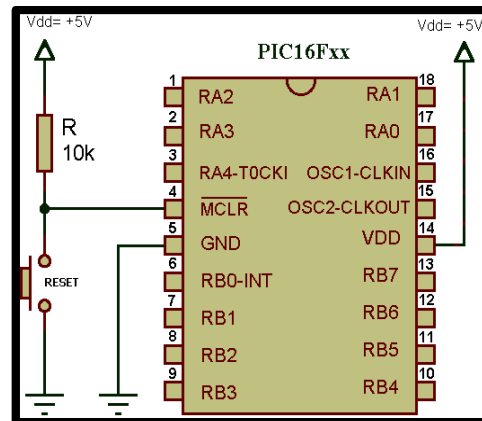


Figura 43. Diagrama de conexión para controlar el reset del PIC

Cuando se presiona el pulsador de Reset el pin $\overline{MCLR}$ pasa a nivel lógico bajo. Otro esquema habitual para el pulsador de Reset incluye una resistencia de absorción de 10K, un diodo que impide la circulación de corriente hacia el pin 4 y otra pequeña resistencia de 100 Ω , como se muestra en la figura 44.

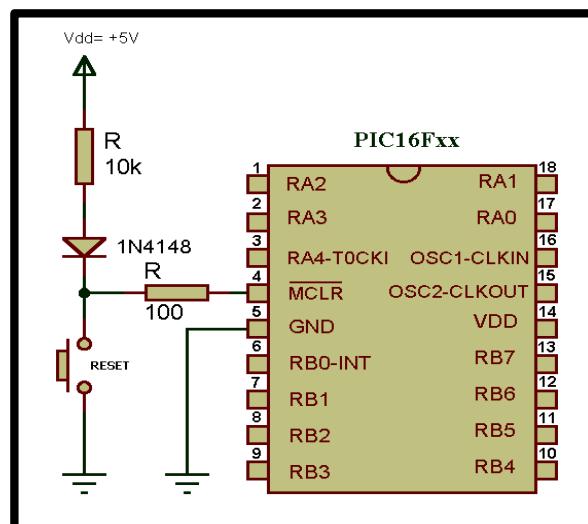


Figura 44. El diodo impide que circule corriente del positivo al interior del microcontrolador

2.13.1 CAUSAS QUE PROVOCAN EL RESET

El Reset pone a cero el PC y coloca en un estado conocido la mayoría de los bits de los registros de control. Existen tres causas principales que originan un Reset:

- Conexión de la alimentación (por: Power On Reset). Se producen al detectarse la subida de la tensión de alimentación desde 1.2 a 1.7 V.
- Activación del pin $\overline{MCLR}$. Esto puede suceder tanto cuando el PIC funciona normalmente como cuando esté en estado de reposo.
- Desbordamiento de Perro Guardián (WDT). Igualmente puede ocurrir en el modo de funcionamiento normal del PIC y en su estado de reposo.

En cualquiera de las cinco posibilidades después del Reset el PC se carga con cero, excepto cuando se desborda el WDT estando el PIC en estado de reposo, donde el PC se incrementa una unidad para ejecutar la siguiente instrucción a la que lo introdujo en ese estado de bajo consumo. En la Tabla 11 se presenta el valor que toma cada uno de los bits de los registros SFR tras cada una de las posibles causas de Reset.

Tabla 11. Valores que toman los bits de los registros SFR tras los posibles Reset

Registro	Dirección	Conexión de la Alimentación	Desbordamiento perro guardián modo normal	Desbordamiento perro guardián modo reposo	MCLR modo normal	MCLR modo reposo
W	----	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
INDIR	00 h	----	----	----	----	----
TMR0	01 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	02 h	0000 h	0000 h	PC + 1	0000 h	0000 h
STATUS	03 h	0001 1xxx	0000 1uuu	uuu 0 0uuu	000u uuuu	0001 0uuu
FSR	04 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PORT A	05 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PORT B	06 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TRIS A	85 h	---1 1111	---1 1111	----u uuuu	---1 1111	---1 1111
TRIS B	86 h	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
OPTION	81 h	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
EEDATA	08 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
EEADR	09 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu

EECON 1	88 h	-- -0 0000	-- -0 ?000	-- -u uuuu	-- -0 ?000	-- -0 ?000
EECON 2	89 h	----	----	----	----	----
PCLATH	0A h	-- -0 0000	-- -0 0000	-- -u uuuu	-- -0 0000	-- -0 0000
INTCON	0B h	0000 000x	0000 000u	uuuu uuuu	0000 000u	0000 0000
u=No cambia x=Independiente ----=No implementado ?=Dependiente de otras condiciones						

2.13.2 ESQUEMA ELÉCTRICO DEL RESET

En la Figura 45 se ofrece el esquema eléctrico que controla la actividad del Reset. En él destacan los temporizadores OST y PWRT.

El temporizador Power-up Timer (PWRT) añade 72 milisegundos que mantienen al microcontrolador en la situación de Reset, para dar tiempo a que se establezca el voltaje de alimentación. Para su funcionamiento hay que poner a nivel bajo el bit de permiso $\overline{PWRT}$, que reside en la palabra de configuración.

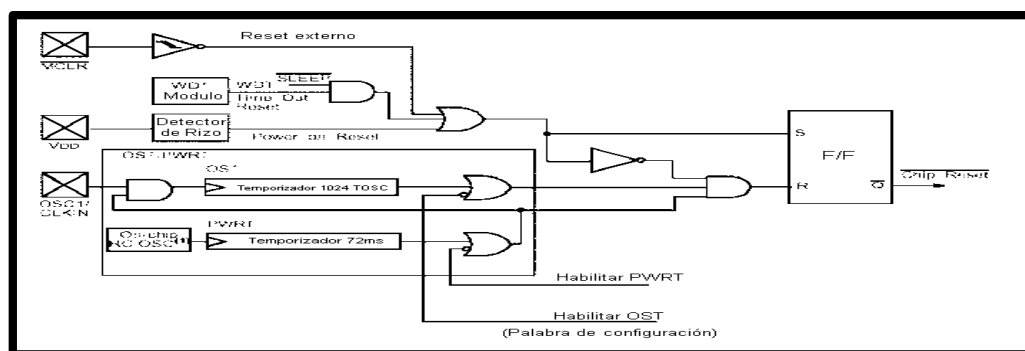


Figura 45. Diagrama simplificado a bloques del Reset

2.13.3 DETERMINACIÓN DEL ORIGEN DEL RESET

Dados los diferentes efectos que provoca el Reset según la causa que lo ha originado, es muy importante averiguar cuál es ésta. Para ello se utilizan dos bits del registro ESTADO, que ocupa la dirección 3 de los dos bancos de la RAM. Se trata del bit $\overline{TO}$ (Timer Out) que se activa con el desbordamiento del WDT y del bit $\overline{PD}$ (Power Down) que se activa cuando el PIC está en estado de reposo. En la Tabla 12 se muestran los estados de estos dos bits, al aplicarse algún tipo de reset.

Tabla 12. Los bits $\overline{TO}$ y $\overline{PD}$ del registro de STATUS determinan la causa que ha originado el Reset

TO	PD	Condición de Reset
1	1	Por (Reset por conexión VDD)
0	1	Desbordamiento WDT en funcionamiento normal
0	0	Desbordamiento WDT en estado de reposo
1	1	Activación MCLR en funcionamiento normal
1	0	Activación MCLR en reposo

2.13.4 RESET POR FALLO EN LA ALIMENTACIÓN

A veces es muy importante inicializar el procesador cuando se produce un fallo en la alimentación y el voltaje desciende por debajo de un valor determinado sin llegar a cero, recuperándose después. Para dicho proceso se propone el circuito de la siguiente Figura 46.

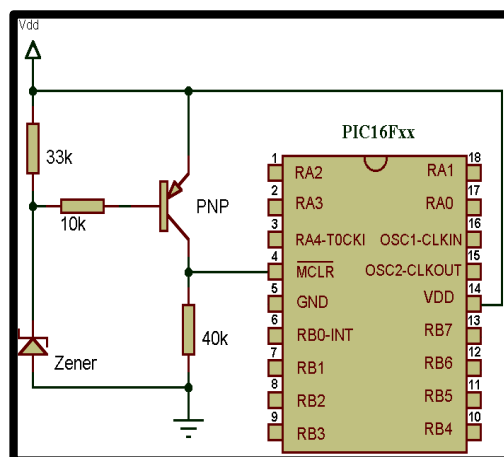


Figura 46. Cuando la tensión de alimentación desciende por debajo del valor $V_{zener} + 0.7V$, se produce un Reset por la activación del pin $\overline{MCLR}$

2.14 MODO DE REPOSO O BAJO CONSUMO DE ENERGÍA (SLEEP)

El microcontrolador PIC16F84 puede estar mucho tiempo consumiendo una cantidad apreciable de energía eléctrica. El modo de reposo o de bajo consumo de energía consiste en reducir la corriente de alimentación, que típicamente oscila alrededor de los 2 miliamperios, y que puede llegar a menos de 10 microamperios, lo que permite alimentarlo con una pequeña pila durante casi dos años. Esta característica lo hace muy recomendable en aquellas aplicaciones en las que hay largos periodos de inactividad a la espera de que suceda algún acontecimiento asíncrono, como la

pulsación de una tecla. En dicho estado se dice que el microcontrolador se ha “dormido”.

Para dormir el microcontrolador basta ejecutar la instrucción SLEEP, entonces el primer elemento que se detiene es el oscilador principal, dejando de producirse los impulsos de trabajo a la frecuencia $F_{osc}/4$. En estas circunstancias el procesador no ejecuta instrucciones. Tampoco funciona el TMR0 al no recibir los impulsos internos. Para que el TMR0 no funcione con impulsos externos se lleva el pin RA4/TOCLKI al positivo. En este estado hay que destacar que el Perro Guardián si queda en funcionamiento al entrar en modo reposo. Se refresca o inicializa, pero sigue funcionando al disponer de un oscilador propio e independiente. En este caso los señalizadores del registro ESTADO quedarán $\overline{TO}=1$ y $\overline{PD}=0$.

Las líneas de E/S mantienen el estado anterior a la entrada en modo reposo, y las que no se hallan conectadas a periféricos y actúan como entradas de alta impedancia se recomienda conectarlas a positivo o a tierra, para evitar posibles fugas de corriente. Véase la Tabla 13.

Tabla 13. Consecuencias del Modo de reposo

Estado Recurso	Consecuencia
Oscilador principal detenido	-No se ejecutan instrucciones -No funciona el TMR0 (RAM/T0CKI conectado a VDD)
WDT	-Se resetea pero sigue funcionando
Líneas E/S congeladas	-Mantienen el estado previo
Consumo mínimo	-Pasa de 2mA a menos de 10µA

2.14.1 SALIDA DEL ESTADO DE REPOSO

El PIC se “duerme” al ejecutarse la instrucción SLEEP en el programa y puede ser “despertado” para reanudar su funcionamiento normal por varias causas:

- Por la activación del pin $\overline{MCLR}$
- Por el desbordamiento del perro guardián
- Cuando se genera una interrupción

Cuando el PIC16F84 está “dormido” y se activa el pin $\overline{MCLR}$, el microcontrolador se “despierta” y comienza a ejecutar la primera instrucción del programa, que reside en el Vector de Reset. Lo mismo sucede cuando se desborda el WDT: si el microcontrolador estaba dormido se despierta y se reinicializa.

Para determinar cuál de las dos causas ha originado el Reset se exploran los señalizadores $\overline{TO}$ y $\overline{PD}$ del registro STATUS, de acuerdo con los valores que se indican en la Tabla 14.

Tabla 14. STATUS y significado de los señalizadores $\overline{PD}$ y $\overline{TO}$

Valor del señalizador	Indicación
PD = 0	- Ejecución SLEEP. Modo de reposo
PD = 1	- Se ha despertado
TO = 0	- Se ha desbordado WDT
TO = 1	- No se ha desbordado WDT

Cuando se despierta el PIC se desarrolla la secuencia del temporizador OST que retarda 1024 Tosc el paso al funcionamiento normal, con objeto de dar tiempo a que se establezca la frecuencia de funcionamiento.

2.14.2 SALIDA DEL ESTADO DE REPOSO MEDIANTE UNA INTERRUPCIÓN

La tercera causa que puede hacer despertar al PIC sucede cuando se produce una interrupción. De las cuatro posibles interrupciones que admite el PIC16F84 la correspondiente al TMR0 no es posible, al carecer de entrada de impulsos, por lo tanto sólo son válidas las otras tres interrupciones:

- Activación del pin RB0/INT
- Cambio de estado de uno de los pines RB7:RB4
- Fin de la escritura de la EEPROM

Cuando se ha ejecutado SLEEP, el PC se queda cargado con la dirección de la siguiente instrucción (PC + 1).

Para que se admita una de las tres interrupciones es necesario que sus bits de permiso (INTE, RBIE y EEIE) estén a 1. En este caso cuando se activa uno de los señalizadores de interrupción se explora el valor del bit GIE, que es el de Permiso Global de Interrupciones. Si $GIE = 1$ y se permiten globalmente las interrupciones, se despierta el PIC, se ejecuta la instrucción (PC + 1), que es la siguiente a SLEEP y, finalmente, se carga en el PC el valor 0004h, saltando a la rutina de interrupción. Como a veces no se desea ejecutar ninguna instrucción después de SLEEP y antes de saltar a la rutina de interrupción, se coloca una instrucción NOP.

Si se produce una interrupción permitida individualmente, pero el bit $GIE = 0$ (Prohibición Global), se despierta el PIC del estado de reposo y continúa la ejecución del programa a partir de la siguiente instrucción a SLEEP, como se muestra en la tabla 15.

Tabla 15. Instrucciones que ejecuta el PIC cuando “despierta” por una interrupción según el valor de bit GIE

Estado de Bits	Siguiente Instrucción
$GIE = 0$ (Prohibición) $RBIE = INTE = EEIE = 1$	PC + 1 (Siguiente a SLEEP)
	PC + 2 (Siguiente)
$GIE = 1$ (Permiso) $RBIE = INTE = EEIE = 1$	PC + 1 (Siguiente a SLEEP)
	0004 (Vector de interrupción)

Cuando $GIE = 0$ y se activa la bandera de una interrupción que tiene a 1 su bit individual de permiso, puede suceder uno de estos dos acontecimientos:

- Si la interrupción ocurre antes de la ejecución de la instrucción SLEEP, ésta se completará como si se tratase de una instrucción NOP. No se borra ni el WDT ni su divisor de frecuencia. El bit TO no se pone a 1 y el bit PD no se borra.
- Si la interrupción ocurre durante o después de la instrucción SLEEP, el microcontrolador se despertará inmediatamente, una vez ejecutada SLEEP. Se borra el WDT y su divisor de frecuencia bit $TO = 1$ y el bit $PD = 0$.

Para asegurar el borrado del WDT cuando se ejecuta una instrucción SLEEP se

recomienda poner delante de ella una instrucción CLR WDT.

En la Figura 47 se presenta un cronograma que muestra la forma de despertarse del PIC16F84 a través de una interrupción.

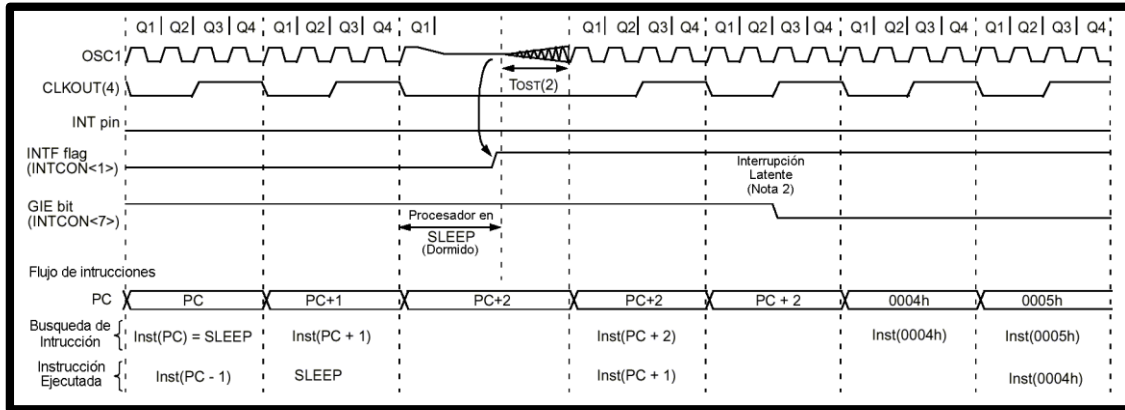


Figura 47. Cronograma del despertar PIC al producirse una interrupción

Notas:

- 1.-XT, HS o LP son los modos asumidos del oscilador
- 2.-TOST = 1024 TOST. Este retardo (delay) no sucede en modo RC.
- 3.-Si se supone que GIE = 1, al despertar el procesador salta a la rutina de interrupción. Si GIE = 0 la ejecución continua detrás de SLEEP.
- 4.-CLKOUT no se precisa en estos modos, pero se usa como referencia.

2.15 TIPOS DE OSCILADORES

Los microcontroladores PIC disponen de un oscilador interno que es el que determina la frecuencia de los impulsos de reloj. Sin embargo, para controlar el valor de la frecuencia y estabilizarla, es necesario colocar algunos elementos externos en los pines 15 y 16 (OSC1/CLKIN y OSC2/CLKOUT). Según el tipo de circuitería externa los osciladores se clasifican en cuatro tipos:

- Oscilador RC
- Oscilador LP

- Oscilador XT
 - Oscilador HS

Para iniciar el trabajo del microcontrolador hay que especificar el tipo de oscilador que se ha implementado. Para este fin existen dos bits FOSC1 y FOSC2 en una posición de la memoria de programa, llamada Palabra de Configuración, especificación que debe elegirse con la combinación correspondiente.

2.15.1 OSCILADOR RC

En este caso se coloca en el pin 16 del PIC16F84 (OSC1/CLKIN) una resistencia y un condensador. Este procedimiento es muy barato, pero la estabilidad de la frecuencia es mediocre y sólo se recomienda en sistemas en los que el tiempo no sea un factor crítico y la economía sea un objetivo prioritario.

Por el pin 15 (OSC2/CLKOUT) sale la cuarta parte de la frecuencia de oscilación ($F_{osc}/4$) que es el tiempo del ciclo de instrucción normal. La figura 48, muestra un oscilador de tipo RC.

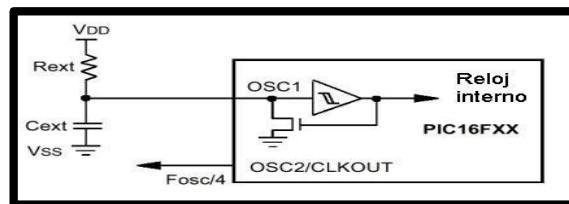


Figura 48. Oscilador de tipo RC

En la Tabla 16, se especifican los valores de capacitor y resistencia, recomendados por el fabricante para obtener una determinada frecuencia.

Valores recomendados: $5K\Omega \leq R_{ext} \leq 100K\Omega$

$C_{ext} > 20pF$

Tabla 16. Valores recomendados para los componentes del oscilador RC con respecto a la frecuencia deseada

Fosc	Rext	Cext
625 KHz	10 K	20pF
80 KHz	10 K	220Pf
80 Hz	10 K	0.1Pf

2.15.2 OSCILADORES DE CRISTAL / RESONADORES CERÁMICOS

En los modos XT, LP, y HS debe conectarse un cristal o un resonador cerámico en los pines OSC1/CLKIN y OSC2/CLKOUT para establecer la oscilación.

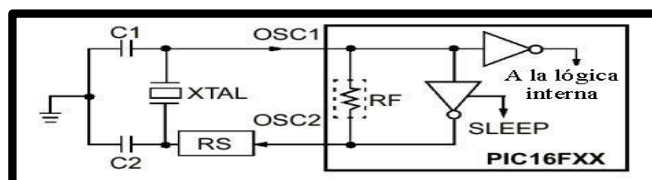


Figura 49. Oscilador de tipo HS, XT ó LP

Oscilador de cristal/Resonador cerámico (configuración de oscilador XT, LP o HS). En la Figura 49 se muestra el esquema de los osciladores HS, XT y LP que usan un cristal de cuarzo y dos condensadores de desacoplo, cuyos valores especifica el fabricante en función de la frecuencia.

Según el tipo de Oscilador y la frecuencia de trabajo, se emplean diferentes valores en los condensadores C1 y C2 que acompañan al cristal de cuarzo. Para frecuencias comprendidas entre 4 MHz y 10 MHz, los condensadores tienen una capacidad de 15 pF a 33 pF. La resistencia RS sólo es necesaria en algunas versiones tipo HS.

En la Tabla 17, se muestran los valores de cristal y de capacitor especificado para cada tipo de oscilador.

Tabla 17. Tabla de selección de cristal y capacitor

Modo	Frecuencia	OSC1/C1	OSC2/C2
LP	32 KHz	68 – 100 pF	68 – 100 pF
	200 KHz	15 – 33 pF	15 – 33 pF
XT	100 KHz	100 – 150 pF	100 – 150 pF
	2 MHz	15 – 33 pF	15 – 33 pF
	4 MHz	15 – 33 pF	15 – 33 pF
HS	4 MHz	15 – 33 pF	15 – 33 pF
	20 MHz	15 – 33 pF	15 – 33 pF

El oscilador diseñado requiere el uso de un cristal de corte paralelo. El uso de una serie de corte de cristal puede dar una frecuencia fuera de la especificada por el fabricante. En los modos XP, LP, HS, el dispositivo puede tener una fuente del reloj externa para manejar el pin OSC1/CLKIN, justo como se observa en la Figura 50.

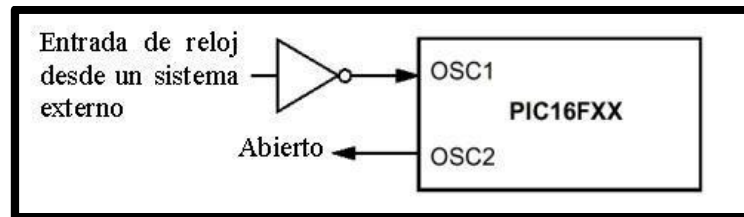


Figura 50. Entrada de reloj externa (configuración de oscilador HS, XT o LP)

2.15.3 OSCILADOR LP

Entre los pines 15 y 16 del microcontrolador se coloca un cristal de cuarzo de la frecuencia a la que se desee funcione. También puede instalarse en lugar del cristal de cuarzo un resonador cerámico. En cada uno de dichos pines se sitúa un condensador de desacoplo.

Este tipo de oscilador está preparado para trabajar con bajas frecuencias, comprendidas entre 35 y 250 KHz. Su principal interés radica en el bajo consumo que necesita el PIC para trabajar a estas frecuencias tan bajas, detalle muy importante en los sistemas que se alimentan con pilas.

2.15.4 OSCILADOR XT

En este tipo de oscilador se coloca un cristal de cuarzo o un resonador cerámico entre los pines 15 y 16, junto a los correspondientes condensadores de desacoplo, cuyos valores se encuentran en la tabla 18, y dependen de la frecuencia seleccionada. El oscilador XT está preparado para trabajar en un rango de frecuencia de 100 KHz – 4 MHz. En la Figura 51 se muestra la configuración típica para este tipo de oscilador.

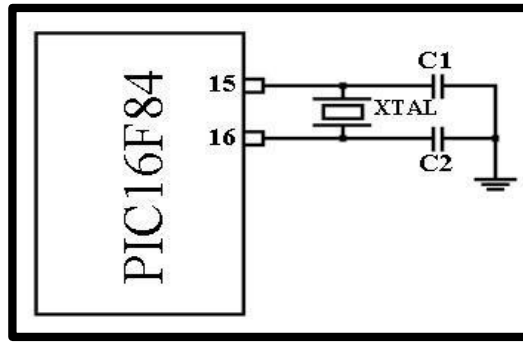


Figura 51. Circuito típico para el oscilador XT

2.15.5 OSCILADOR HS

Es un oscilador de alta velocidad y muy estable. Funciona en frecuencias comprendidas entre 4 y 20 MHz. Utiliza un cristal de cuarzo o un resonador cerámico entre los pines 15 y 16, junto a dos condensadores de desacoplo. No todos los modelos de PIC admiten este tipo de oscilador. Debido su alta frecuencia de oscilación, el costo es muy elevado.

2.15.6 OTROS CIRCUITOS OSCILADORES

Empleando algunas puertas lógicas TTL muy sencillas pueden construirse osciladores de un inmejorable comportamiento y amplio rango de frecuencias. Hay dos tipos de osciladores con cristal: de resonancia en paralelo y de resonancia en serie.

2.15.6.1 OSCILADORES DE RESONANCIA EN PARALELO

El circuito oscilador con resonancia en paralelo, como se muestra en la Figura 52, se usa para la frecuencia del cristal que incorpora. La puerta inversora 74AS04 realiza una inversión de fase de 180°.

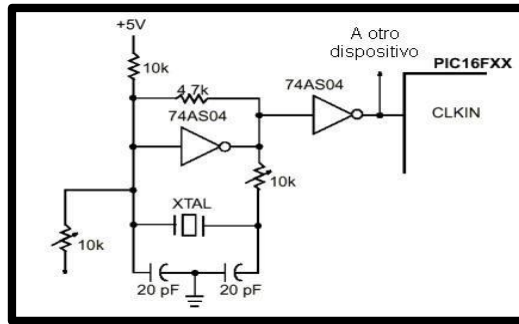


Figura 52. Oscilador de cristal y resonancia en paralelo

2.15.6.2 OSCILADOR DE RESONANCIA SERIE

El circuito oscilador resonante serie que se muestra en la Figura 53 permite una enorme precisión en la frecuencia que, en general, es la que determina el cristal de cuarzo.

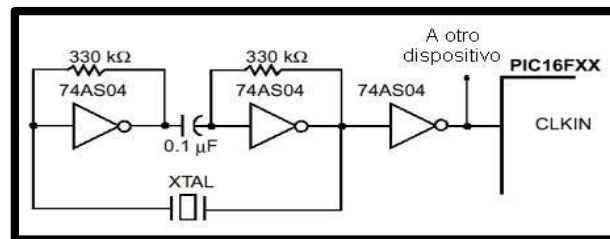


Figura 53. Oscilador de cristal y resonancia en serie

2.16 PALABRA DE CONFIGURACIÓN

La palabra de configuración ocupa una posición especial de la memoria de programa FLASH, la dirección 2007h, cuyos 14 bits regulan algunas funciones vitales del procesador. La organización de esta posición de la memoria se presenta en la Figura 54.

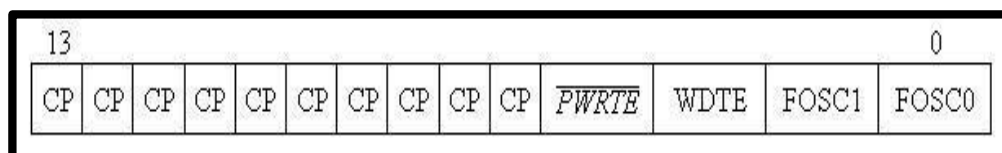


Figura 54. Estructura interna de la palabra de configuración

Los 10 bits de más peso, denominados todos ellos CP (Código de protección), se encargan de proteger el acceso a la memoria de código. Si CP es 0 el programa de instrucción no puede leerse, por el contrario, si CP es 1 el programa grabado en la FLASH se puede leer. Además el código de protección evita que pueda ser accedida la EEPROM de datos y, finalmente, si se modifica el bit CP de 0 a 1, se borra completamente la EEPROM.

Con los dos bits de menos peso, FSOC0 y FOSC1, se determina el tipo de oscilador que emplea el procesador.

El bit WDTE es el que habilita o inhabilita el funcionamiento del perro guardián. Si WDTE es 1 el perro guardián entra en funcionamiento. Por último, el bit $\overline{PWRT\overline{E}}$ es el de habilitación del temporizador PWRT, que retarda 72 ms el Reset para dar tiempo a estabilizar el voltaje de alimentación. Si $\overline{PWRT\overline{E}}$ es 0, actúa el temporizador.

A continuación, se muestra la descripción de cada uno de los bits que conforman la palabra de configuración del PIC16F84A.

CP: Bits de protección de la memoria de código

1 = No protegida

0 = El programa no se puede leer, evitando copias.

PWRT $\overline{E}$: Activación del temporizador <<POWER-UP>>

1 = Desactivada

0 = Activado

WDTE: Activación del perro guardián (WDT)

1 = Activado el WDT

0 = Desactivado

FOSCI-FOSC0: selección del oscilador utilizado, el cual se muestra en la tabla 18.

Tabla 18. Combinaciones para elegir el tipo de oscilador

FOSC1	FOSC2	Oscilador
1	1	RC
1	0	HS
0	1	XT
0	0	LP

Una de las pocas diferencias que tiene el PIC16C84 con el PIC16F84 es la actuación del bit PWRTE de la Palabra de Configuración, que tiene invertida su función en ambos. En el primero cuando este bit vale 1 el temporizador de Power-Up está activado, mientras que sucede lo contrario en el segundo. En ambos microcontroladores el temporizador retrasa 72 ms el funcionamiento del microcontrolador al conectarle la alimentación para asegurar la estabilidad del voltaje aplicado.

2.17 PALABRA DE IDENTIFICACIÓN (ID)

Son cuatro posiciones de la memoria FLASH que ocupan las direcciones comprendidas entre la 2000h y 2003h. Sólo son accesibles en lectura y escritura durante el proceso de grabación del programa.

De los 14 bits de las palabras de identificación sólo son válidos los 4 de menos peso, y en ellos el diseñador puede escribir códigos de identificación, números de serie, datos sobre la fabricación, modelos, etc. En la figura 55 se destacan los bits válidos de cada posición

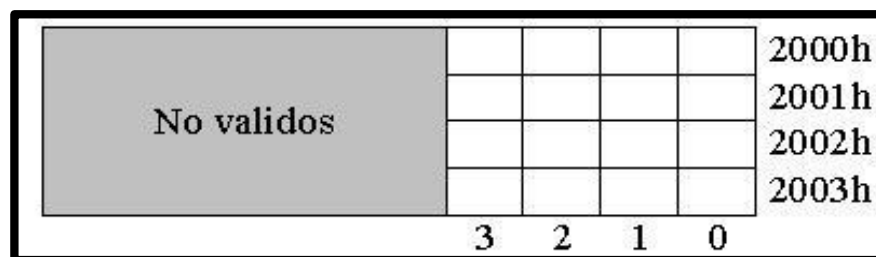


Figura 55. Palabra de identificación, solo son válidos los 4 bits de menos peso

2.18 CIRCUITERÍA FIJA

Para comenzar a funcionar el PIC16F84 precisa de tres recursos:

- Voltaje de alimentación
- Frecuencia de referencia
- Reinicialización o Reset

El voltaje de alimentación es imprescindible para el funcionamiento de todos los componentes electrónicos, y su valor puede estar comprendido entre +2 y +6 VDC, siendo +5V la tensión típica, compatible con el resto de la circuitería del sistema. Este voltaje se aplica a los pines 15 y 5, positivo y tierra, respectivamente. El consumo de corriente depende de la frecuencia de funcionamiento y del voltaje de alimentación, no sobrepasando los 10 mA. La frecuencia de referencia sirve para estabilizar y fijar la frecuencia del oscilador interno, que determina la de trabajo del procesador. Se obtiene de un cristal de cuarzo situado entre los pines 16 (OSC1/CLKIN) y 15 (OSC2/CLKOUT).

Por último, un microcontrolador debe disponer de diversas formas de inicializar su programa de trabajo. Para provocar un Reset externo se utiliza el pin 4 ($\overline{MCLR}/V_{pp}$), que también tiene la función de introducir la tensión V_{pp} de 12 a 14 VDC que se necesita durante el proceso de grabación del programa. En la figura 56 se muestran los cinco pines del PIC16F84 que soportan los tres recursos principales.

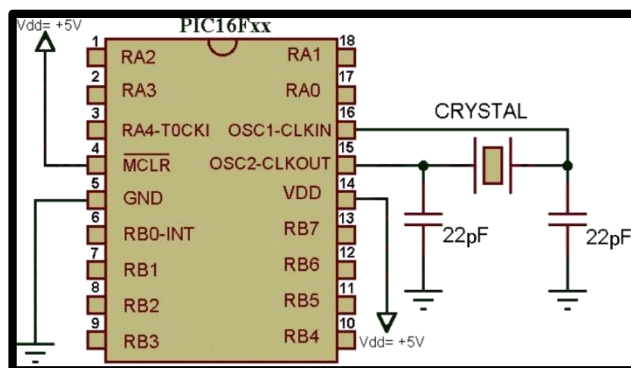


Figura 56. Se resaltan los cinco pines del PIC16F84 que soportan las tres funciones principales: alimentación, frecuencia de referencia y Reset

2.19 JUEGO DE INSTRUCCIONES

Todos los modelos de PIC responden a la arquitectura RISC, que significa «Computadora de Juego de Instrucciones Reducido». No sólo implica que el número de instrucciones máquina que es capaz de interpretar y ejecutar el procesador es pequeño, como sucede con los PIC16X8X que constan de 35, sino también que posee las siguientes características. Las instrucciones son simples y rápidas. La falta de complejidad en la operación que realizan las instrucciones de los procesadores RISC permite que sean ejecutadas, mayoritariamente, en un solo ciclo de instrucción. Los PIC's tardan un ciclo en ejecutar la mayoría de las instrucciones, excepto las de salto, que tardan el doble. Las instrucciones son ortogonales. Apenas tienen restricciones en el uso de operando. Cualquier instrucción puede usar cualquier operando. La longitud de los datos y las instrucciones es constante. Todas las instrucciones tienen la misma longitud, 14 bits en los PIC16X8X, y todos los datos de un byte. La arquitectura Harvard del procesador aísla la memoria de instrucciones de datos, pudiendo tener diferente tamaño sus palabras.

2.19.1 CLASIFICACIÓN DE FORMATOS

Las instrucciones de los PIC de la gama media, entre los que se encuentran los modelos PIC16X8X, tienen 14 bits de longitud. Dicho formato se divide en diferentes campos de bits, cada uno de los cuales referencia a operandos o elementos que maneja la instrucción en la operación que realiza el procesador. A continuación, se describen dichos campos.

a) Campo de código OP

Los bits de este campo sirven para definir la operación que realiza la instrucción.

b) Campo de los operandos fuente (f) y destino (d)

Estos campos de bits definen los registros que actúan como operandos en la instrucción. Suelen referenciar la dirección que ocupan en la memoria de datos.

c) Campo de operando inmediato o literal (k)

Es un campo de bits que contiene el valor del operando inmediato.

d) Campo de referencia a un bit (b)

Suele ser un campo de 3 bit que indica la posición de un bit concreto dentro de un registro de 8 bits.

e) Campo de la dirección de salto

En las instrucciones de salto CALL y GOTO hay un campo de bits que contiene la dirección de la siguiente instrucción que hay que ejecutar. Dicho campo de bits se carga en el PC en las instrucciones de salto incondicional.

La clasificación de los formatos se muestra en la tabla 19.

Tabla 19. Clasificación de formatos

<i>Símbolo</i>	<i>Descripción</i>
f	Dirección del registro
W	Registro de trabajo
k	Campo que contiene un valor inmediato, que puede ser un operando (8 bits) o una dirección para el PC (11 bits).
x	Valor indeterminado de un bit. Puede ser 1 o 0.
d	Selección del destino. Cuando d=0, el resultado se almacena en W. Si d=1, el resultado se almacena en el archivo de registro f. Por default d=1.
label	Nombre de etiqueta
PC	Contador de Programa
PCLATH	Retenedor de Contador de Programa
GIE	Permiso global de interrupciones (Global Interrupt Enable bit)
WDT	Temporizador Perro Guardián (Watchdog Timer/Counter)
TO	Bit Time-out
PD	Bit Power-down
[]	Opciones
()	Contenido
→	Asignado a
< >	Campo de un bit de un registro. Por ejemplo: ESTADO <5>.
∈	En el conjunto de ejemplo: $d \in [0,1]$

2.19.2 INSTRUCCIONES DEL PIC16F84A

En la tabla 20. Se muestra el juego de 35 instrucciones del PIC16F84A.

Tabla 20. Juego de Instrucciones del PIC16F84A

Mnemonicos, Operandos		Descripción	Ciclos	Opcode 14 bits		Banderas Afectadas
				MSb	LSb	
ADDLW	k	Suma la literal con W	1	11 111x	kkkk kkkk	C,DC,Z
ADDWF	f,d	Suma W y f	1	00 0111	dfff ffff	C, DC, Z
ANDLW	k	AND entre la literal y W	1	11 1001	kkkk kkkk	Z
ANDWF	f,d	AND W con f	1	00 0101	dfff ffff	Z
BCF	f,d	Limpia el bit f	1	01 00bb	bfff ffff	
BSF	f,d	Pone a 1 el bit f	1	01 01bb	bfff ffff	
BTFS	f,d	Explora un bit de f y brinca si vale 0	1 (2)	01 10bb	bfff ffff	
BTFS	f,d	Explora un bit de f y brinca si vale 1	1 (2)	01 11bb	bfff ffff	
CALL	k	Llamada a Subrutina	2	10 0kkk	kkkk kkkk	
CLRF	f	Borra f	1	00 0001	1fff ffff	Z
CLRW	-	Borra W	1	00 0001	0xxx xxxx	Z
CLRWD	k	Limpia Watchdog Timer	1	00 0000	0110 0100	TO, PD
COMF	f,d	Complementa f	1	00 1001	dfff ffff	Z
DECF	f,d	Decrementa f	1	00 0011	dfff ffff	Z
DECFSZ	f,d	Decrementa f y salta si es 0	1 (2)	00 1011	dfff ffff	
GOTO	k	Salto incondicional	2	10 1kkk	kkkk kkkk	
INCF	f,d	Incrementa f	1	00 1010	dfff ffff	Z
INCFSZ	f,d	Incrementa f y salta si es 0	1 (2)	00 1111	dfff ffff	
IORLW	k	OR inmediata con W	1	11 1000	kkkk kkkk	Z
IORWF	f,d	Or entre W y f	1	00 0100	dfff ffff	Z
MOVF	f,d	Mueve f	1	00 1000	dfff ffff	Z
MOVLW	k	Mueve la literal a W	1	11 00xx	kkkk kkkk	
MOVWF	f	Mueve W a f	1	00 0000	1fff ffff	
NOP	-	No opera	1	00 0000	0xx0 0000	
RETFIE	k	Retorno de interrupción (GIE = 1)	2	00 0000	0000 1001	
RETLW	k	Retorno de subrutina y carga W = k	2	11 01xx	kkkk kkkk	
RETURN	k	Retorno de subrutina	2	00 0000	0000 1000	
RLF	f,d	Rota f a la izq. A través del acarreo	1	00 1101	dfff ffff	C
RRF	f,d	Rota f a la der. A través del acarreo	1	00 1100	dfff ffff	C
SLEEP	k	Paso a modo SLEEP	1	00 0000	0110 0011	TO, PD
SUBLW	k	Resta W de la literal	1	11 110x	kkkk kkkk	C,DC,Z
SUBWF	f,d	Resta W y f	1	00 0010	dfff ffff	C,DC,Z
SWAPF	f,d	Intercambia nibbles en f	1	00 1110	dfff ffff	
XORLW	k	OR exclusiva entre la literal y W	1	11 1010	kkkk kkkk	Z
XORWF	f,d	OR exclusiva W con f	1	00 0110	dfff ffff	Z

2.19.3 INSTRUCCIONES QUE MANEJAN REGISTROS

Responden a la sintaxis nemónico *f, d*, siendo *f* y *d* los dos operandos fuente y destino que se hallan implementados por los registros de 8 bits de la memoria de datos.

El registro *f* viene referenciado por la dirección de 7 bits que ocupa, mientras que el destino sólo por 1, que si vale 0 es el *W* y si vale 1 es el fuente. En la tabla 21 se muestran las instrucciones de este grupo con sus características más interesantes.

Tabla 21. Principales características de las instrucciones de los PIC16X8X

INSTRUCCIONES QUE MANEJAN REGISTROS				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 BITS	SEÑALIZADORES
ADDWF <i>f,d</i>	Suma <i>W</i> y <i>f</i>	1	00 0111 dfff ffff	C,DC,Z
ANDWF <i>f,d</i>	AND <i>W</i> y <i>f</i>	1	00 0101 dfff ffff	Z
CLRF <i>f</i>	Borra <i>f</i> (pone todos los bits a 0)	1	00 0001 dfff ffff	Z
CLRW ---	Borra <i>W</i>	1	00 0001 0xxx xxxx	Z
COMF <i>f,d</i>	Complementa <i>f</i> (Invierte)	1	00 1001 dfff ffff	Z
DECF <i>f,d</i>	Decrementa <i>f</i>	1	00 0011 dfff ffff	Z
INCF <i>f,d</i>	Incrementa <i>f</i>	1	00 1010 dfff ffff	Z
IORWF <i>f,d</i>	OR entre <i>W</i> y <i>f</i>	1	00 0100 dfff ffff	Z
MOVF <i>f,d</i>	Mueve <i>f</i>	1	00 1000 dfff ffff	Z
MOVWF <i>f</i>	Mueve <i>f</i>	1	00 0000 dfff ffff	----
NOP ---	No opera	1	00 0000 0xx0 0000	----
RLF <i>f,d</i>	Rota <i>f</i> a la izq. A través del acarreo	1	00 1101 dfff ffff	C
RRF <i>f,d</i>	Rota <i>f</i> a la der. A través del acarreo	1	00 0100 dfff ffff	C
SUBWF <i>f,d</i>	Resta <i>W</i> a <i>f</i>	1	00 0110 dfff ffff	C,DC,Z
SWAPF <i>f,d</i>	Intercambia nibbles	1	00 1110 dfff ffff	----
XORWF	XOR de <i>W</i> con <i>f</i>	1	00 0110 dfff ffff	Z

2.19.4 INSTRUCCIONES QUE MANEJAN BITS

Sólo hay dos instrucciones en este grupo, pero son muy flexibles. Una de ellas pone a 1 (*bsf*) cualquier bit de un registro, mientras que la otra pone a 0 (*bcf*), tal como se muestra en la tabla 22.

Tabla 22. Características más importantes de las dos instrucciones que manejan un bit determinado de un registro

INSTRUCCIONES QUE MANEJAN BITS				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 BITS	SEÑALIZADORES
BCF f,d	Borra bit de f	1	01 00bb bfff ffff	----
BSF f,d	Pone a 1 el bit de f	1	01 01bb bfff ffff	----

2.19.5 INSTRUCCIONES DE BRINCO (SKIP)

Sólo hay 4 instrucciones de salto condicional en los PIC de la gama media. Dos de ellas examinan un bit de un registro y según valga 1 o 0, brincan o no. Recuérdese que un brinco es un «salto» pequeño, sólo se salta la instrucción siguiente a la condicional. Las otras dos instrucciones incrementan o decrementan un registro y la posibilidad del brinco se efectúan si con esa operación el valor del registro ha llegado a cero. Cuando estás instrucciones no brincan por que no se cumple la condición, tardan un ciclo de instrucción en ejecutarse. Para el caso en que brinquen tardan el doble, tal como se muestra en la tabla 23.

Tabla 23. Características más relevantes de las cuatro instrucciones condicionales de brinco

INSTRUCCIONES DE BRINCO				
SINTAXIS	OPERACIÓN	CICLO	FORMATO 14 BITS	SEÑALIZADORES
BTFSC f,d	Explora un bit de f y brinca si vale 0	1(2)	01 10bb bfff ffff	----
BTFSS f,d	Explora un bit de f y brinca si vale 1	1(2)	01 11bb bfff ffff	----
DECFSZ f,d	Decrementa f y si es 0, brinca	1(2)	00 1011 dfff ffff	----
INCFSSZ f,d	Decrementa f y si es 1, brinca	1(2)	00 1111 dfff ffff	----

2.19.6 INSTRUCCIONES QUE MANEJAN OPERANDOS INMEDIATOS

Se trata de media docena de instrucciones que realizan una operación con un valor inmediato de 8 bits que se proporcionan dentro del formato de la instrucción, el cual

sólo tiene dos campos: el del Código OP (6 Bits) y el de operando inmediato (8 bits), como se puede observar en la tabla 24.

Tabla 24. Características más importantes de las instrucciones que manejan operandos inmediatos (k)

INSTRUCCIONES QUE MANEJAN OPERANDOS INMEDIATOS				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 BITS	SEÑALIZADORES
ADDLW k	Suma inmediata con W	1	11 111x kkkk kkkk	C,DC,Z
ANDLW k	AND inmediato con W	1	11 1001 kkkk kkkk	Z
IORLW k	OR inmediato con W	1	11 1000 kkkk kkkk	Z
MOVLW k	Mueve a W un valor inmediato	1	11 00xx kkkk kkkk	----
SUBLW k	Resta W de un inmediato	1	11 110x kkkk kkkk	C,DC,Z
XORLW k	OR exclusiva con W	1	11 1010 kkkk kkkk	----

2.19.7 INSTRUCCIONES DE CONTROL Y ESPECIALES

En este grupo se incluyen las instrucciones que rompen la secuencia normal del programa porque alteran el contenido del PC y también las instrucciones especiales.

La instrucción de salto incondicional goto carga en el PC la dirección de la nueva instrucción. La instrucción call de llamada a Subrutina, antes de cargar el PC con la dirección de la instrucción a saltar, salva la dirección de la partida guardando en la cima de la pila el valor actual del PC. De esta manera, al retornar de la subrutina se saca de la pila la dirección de regreso del programa principal.

Para realizar un retorno de una subrutina se pueden emplear dos instrucciones. La más habitual es return, que se limita a extraer de la cima de la pila el valor que carga en el PC. Otra más completa es retlw k, que además de hacer lo mismo que return, carga en W el valor inmediato que contiene k. Es decir, devuelve un parámetro desde la subrutina.

Para el final de las interrupciones hay otra instrucción cuyo nemónico es retfie. La operatividad de esta instrucción consiste en cargar en el PC el contenido de la cima de la pila y poner el bit GIE = 1, pues al comenzar la interrupción este bit se pone automáticamente a 0 para evitar que cuando se atiende una interrupción se produzca

otra. GIE es el bit de permiso de todas las interrupciones.

En cuanto a las interrupciones especiales, se han incluido dos en este grupo: clrwtd y sleep. La primera pone a 0 el contenido del Perro Guardián, es decir, lo refresca o lo reinicializa. El Perro Guardián si se desborda (pasa de 0xff a 0x00) provoca un Reset. La instrucción clrwtd hay que colocarla estratégicamente en ciertos puntos del programa para evitar la reinicialización.

La instrucción sleep introduce al procesador en un modo de funcionamiento que se llama de Reposo o de Bajo Consumo. Detiene el oscilador y el procesador queda congelado, no ejecutando instrucciones y manteniendo el mismo valor los Puertos de E/S. También pone los bits $\overline{PD} = 0$ y $\overline{TO} = 1$ y borra al Perro Guardián y al Divisor de frecuencia, como se muestra en la tabla 25.

Es recomendable incluir la instrucción clrwtd para que no se desborde el WDT (Watch Dog Timer o Perro Guardián). Las instrucciones de salto cuentan el doble de ciclos que las instrucciones normales. (ITO, 2018)

Tabla 25. Principales características de las instrucciones del control del flujo del programa y de las especiales.

INSTRUCCIONES QUE MANEJAN OPERANDOS INMEDIATOS				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 BITS	SEÑALIZADORES
CALL k	Llamada a subrutina	2	10 0kkk kkkk kkkk	TO#,PD#
CLRWDT	Borra o refresca el PeroGuardián	1	00 0000 0110 0100	-----
GOTO k	Salto incondicional	2	10 1kkk kkkk kkkk	-----
RETFIE	Retorno de interrupción (GIE = 1)	2	00 0000 0000 1001	-----
RETLW k	Retorno subrutina y carga W = k	2	11 01xx kkkk kkkk	-----
RETURN	Retorno de subrutina	2	00 0000 0000 1000	-----
SLEPP	Pasa al modo de reposo	1	00 0000 0110 0011	TO#,PD#

2.20 SENSORES Y TRANSDUCTORES.

El término sensor se refiere a un elemento que produce una señal relacionada con la cantidad que se está midiendo. Por ejemplo, en el caso de un elemento para medir temperatura mediante resistencia eléctrica, la cantidad que se mide es la temperatura y el sensor transforma una entrada de temperatura en un cambio de resistencia. Con frecuencia se utiliza el término transductor en vez de sensor. Los transductores se definen como el elemento que al someterlo a un cambio físico experimenta un cambio relacionado. Es decir, los sensores son transductores. Sin embargo, en un sistema de medición se pueden utilizar transductores, además de sensores, en otras partes del sistema para convertir las señales de una forma dada en otra distinta.

Dado que hay seis tipos de señales: mecánicas, térmicas, magnéticas, eléctricas, ópticas y moleculares (químicas), cualquier dispositivo que convierta una señal de un tipo en una señal de otro tipo debería considerarse un transductor, y la señal de salida podría ser de cualquier forma física útil. En la práctica, no obstante, se consideran transductores por antonomasia aquellos que ofrecen una señal de salida eléctrica. Ello se debe al interés de este tipo de señales en la mayoría de los procesos de medida. (*Bolton, 2001*)

2.20.1 SENSOR DE LUZ

Las resistencias dependientes de la luz, LDR (Light Dependent Resistor) o fotorresistencias, son dispositivos que varían sus resistencias en función de la luz que incide sobre su superficie. Cuanto mayor sea la intensidad luminosa que incide sobre ella menor será la resistencia entre extremos de la misma.

Funcionamiento

Cuando la LDR no está expuesta a radiaciones luminosas los electrones están firmemente unidos en los átomos que la conforma pero cuando sobre ella inciden radiaciones luminosas esta energía libera electrones con lo cual el material se hace más conductor, y de esta manera disminuye su resistencia.

Las resistencias LDR solamente reducen su resistencia con una radiación luminosa

situada dentro de una determinada banda de longitudes de onda. Las construidas con sulfuro de cadmio son sensibles a todas las radiaciones luminosas visibles, las construidas con sulfuro de plomo solamente son sensibles a las radiaciones infrarrojas.

Si medimos entre sus extremos nos encontraremos que pueden llegar a medir en la oscuridad valores cercanos al MegaOhm ($1M\Omega$) y expuestas a la luz mediremos valores en el entorno de los $100\ \Omega$.

El tiempo de respuesta típico de un LDR está en el orden de la décima de segundo. (*Electronics, 2012*).

2.20.2 SENSOR CNY70

El dispositivo CNY70 es un sensor óptico infrarrojo, de un rango de corto alcance (menos de 5 cm) que se utiliza para detectar colores de objetos y superficies. Contiene un emisor de radiación infrarroja -fotodiodo- y un receptor -fototransistor-. El fotodiodo emite un haz de radiación infrarroja, el fototransistor recibe ese haz de luz cuando se refleja sobre alguna superficie u objeto.

Para distinguir los terminales a fin de poder hacer el conexionado correctamente, tienes que colocar el sensor con la parte del fotodiodo y del fototransistor hacia arriba y los terminales hacia abajo. Además, la cara del sensor que está serigrafiada con el nombre del dispositivo ha de estar situada en tu derecha, tal y como se muestra en la siguiente imagen. (*Tecnosefarad 2014*)

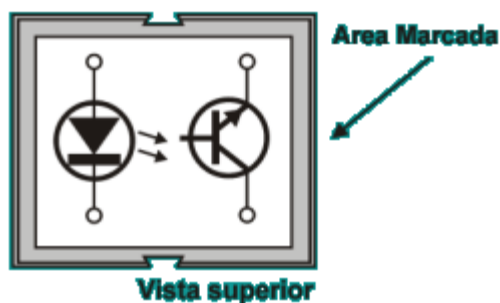


Figura 57. Esquema interno del CNY70 (*Tecnosefarad 2014*)

2.20.3 SENSOR ULTRASÓNICO

Es un sensor de distancias por ultrasonidos capaz de detectar objetos y calcular la distancia a la que se encuentra en un rango de 2 a 450 cm. El funcionamiento es simple, envía una señal ultrasónica inaudible y nos entrega el tiempo que demora en ir y venir hasta el obstáculo más cercano que detecta.

Generalmente están conformados por dos cilindros puestos uno al lado del otro, uno de ellos es quien emite la señal ultrasónica, mientras que el otro es quien la recibe.

Características

- Dimensiones del circuito: 43 x 20 x 17 mm
- Tensión de alimentación: 5 Vcc
- Frecuencia de trabajo: 40 KHz
- Rango máximo: 4.5 m
- Rango mínimo: 1.7 cm
- Duración mínima del pulso de disparo (nivel TTL): 10 μ S.
- Duración del pulso eco de salida (nivel TTL): 100-25000 μ S.
- Tiempo mínimo de espera entre una medida y el inicio de otra 20 mS.

Pines de conexión:

- VCC
- Trigger (Disparo del ultrasonido)
- Echo (Recepción del ultrasonido)
- GND
- Distancia = {(Tiempo entre Trig y el Echo) * (V.Sonido 340 m/s)}/2 (Cristian Veloso 2016 Sensor us-100)

2.20.4 INTERRUPTORES MECÁNICOS

Se trata de un módulo de 4 relés (o relays) que funcionan a 5 Voltios, capaces de manejar cargas de hasta 10 Amperes en 250 Voltios, convenientemente aislados mediante optoacopladores de las entradas, las que cuentan con leds individuales que sirven como indicadores de estado.

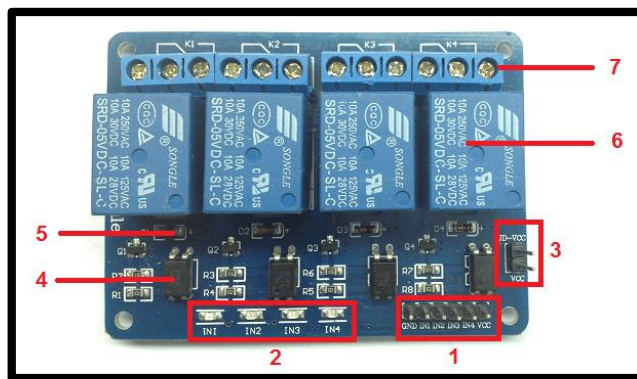


Figura 58. Partes del módulo (profeetolocka 2015)

Como se puede apreciar, la placa tiene un conector de entradas (IN1 a IN4) y alimentación (GND es masa o negativo y Vcc es el positivo) [1], cuatro leds que indican el estado de la entradas [2], un jumper selector para la alimentación de los relés [3], cuatro optoacopladores del tipo FL817C [4], cuatro diodos de protección [5], cuatro relés marca SONGLE con bobinas de 5V y contactos capaces de controlar hasta 10 Amperes en una tensión de 250V [6] y cuatro borneras, con tres contactos cada una (Común, Normal abierto y Normal cerrado), para las salidas de los relés [7]. (Profetolocka 2015)

2.20.5 INTERRUPTORES DE ESTADO SÓLIDO

Para realizar la conmutación electrónica de los circuitos se utilizan diversos dispositivos de estado sólido.

TRANSISTOR BD135

Es un transistor bipolar, compuesto por silicio. Es un amplificador de audio con una corriente máxima de colector de 1.5 ampere con tres regiones consecutivas de diferente conductibilidad eléctrica los cuales forman dos uniones n-p-n, las dos regiones extremas tienen un mismo tipo de conductibilidad, la intermedia, conductibilidad de otro tipo, estas son llamadas emisor, colector y base. Este es muy difundido en las industrias por el amplio uso al que es sometido en diversos aparatos electrónicos.

Características

- Polaridad (N-P-N)
- Amplificador de audio, driver
- Corriente máxima de colector (I_c) 1.5 Ampere
- De colector a base (CBO) 180 Voltios
- De colector a emisor (CEO) 160 Voltios
- De emisor a base (EBO) 5 Voltios
- Ganancia típica de la corriente directa (h_{fe}) 60 Min
- Máxima disipación de potencia en colector (P_d) 20* (Watts)
- Frecuencia MHz 140 Min (*EcuRed 2013*)

2.20.6 INTEGRADO LM358

Consisten de dos amplificadores operacionales independientes, con compensación de frecuencia de ganancia alta diseñados para operar un solo suministro o suministro dividido en una amplia gama de voltajes. La operación desde suministros divididos también es posible si la diferencia entre las dos fuentes es de 3 V a 32 V y V_{CC} es por lo menos 1.5 V más positivo que el voltaje de modo común. Los dispositivos ofrecen una corriente de bajo consumo independiente de la magnitud del voltaje de la fuente de alimentación.

Las aplicaciones incluyen transductor amplificador, bloques de amplificación de CC y todos los circuitos de amplificador operacional convencional que ahora pueden ser implementados más fácilmente en los sistemas de tensión de alimentación única. Por ejemplo, estos dispositivos pueden operarse directamente desde fuentes de alimentación de 5 V estándar utilizadas en sistemas digitales y pueden proporcionar fácilmente la electrónica de interfaz requerida sin fuentes adicionales de ± 5 V.

Características

- Amplio rango de suministro
 - Fuente única: 3 V a 32 V
 - Fuentes duales: ± 1.5 V a ± 16 V

- Drenaje de corriente de suministro baja independiente del voltaje de suministro: 0.7 mA típica
- Ancho de banda de ganancia de unidad: 0.7 MHz
- El rango de voltaje de modo común de entrada incluye tierra, lo que permite detección directa cerca de tierra.
- Rango de voltaje de entrada diferencial igual al voltaje de alimentación de rango máximo: 32 V
- Ganancia de voltaje diferencial de lazo abierto: 100 dB típica
- Compensación de frecuencia interna
- Parámetros de desviación y de polarización de entrada bajos
- Bajo voltaje de desviación de entrada: 3 mV típica
 - Versiones A: 2 mV típica
- Corriente de desviación de entrada: 2 nA típica.
- Corriente de polarización de entrada: 20 nA
 - Versiones A: típico 15 nA típica (*Ti 1986- 2016*).

2.20.7 DRIVER L293D

Los dispositivos L293 y L293D son cuádruples de alta Drivers Half-H actuales. El L293 está diseñado para suministro separado de lógica de entrada proporciona corrientes de accionamiento bidireccionales de hasta 1 A en voltaje de alimentación: 4.5 V a 36 V protección interna contra ESD para proporcionar corrientes de accionamiento bidireccionales de hasta 600 mA a voltajes de 4.5 V a 36 V. El L293D está diseñado a voltajes de 4.5 V a 36 V. Ambos dispositivos están diseñados para impulsar cargas inductivas, como relés, solenoides, motores paso a paso bipolares y de corriente continua, así como otras cargas de alta corriente / alto voltaje en positivo.

Cada salida es un circuito completo de accionamiento de tótem, Supresión (L293D) con un disipador de transistores Darlington y un pseudo fuente de Darlington. Los controladores están habilitados en pares, con los controladores 1 y 2 habilitados por 1,2EN y los controladores 3 y 4 habilitado por 3,4EN.

Los modelos L293 y L293D se caracterizan por su funcionamiento De 0 ° C a 70 °

Características

- Amplio rango de voltaje de alimentación 4.5 V a 36 V
- Suministro separado de lógica de entrada
- Protección interna contra ESD
- Entradas de alta inmunidad al ruido
- Corriente de salida 1 A por canal (600 mA para L293D)
- Corriente de salida pico 2 A por canal (1.2 A para L293D)
- Diodos de pinza de salida para transitorios inductivos (*Ti 1986-2016*)

2.20.8 MOTOR CD

Un motor de corriente directa cuenta con una armadura, cepillos, conmutador y bobinas de campo. La armadura es el conductor giratorio en un motor de CD. Los cepillos son componentes estacionarios del conmutador, que suministran corriente a los segmentos rotatorios del conmutador. El Conmutador es un dispositivo utilizado en un motor CD para revertir la corriente en la armadura cada media rotación de tal manera que los campos magnéticos funcionen juntos para mantener la rotación. Las bobinas de Campo son los devanados estacionarios del motor CD que generan el campo electromagnético.

La velocidad de un motor de cd puede variarse mediante el cambio de una de las variables de la ecuación fundamental de la velocidad. Los siguientes son métodos utilizados para el control de la velocidad de un motor de CD.

1. Modificación del flujo de excitación, por medio de un reóstato variable serie o shunt. Este método se conoce como control de campo.
2. Variación de la tensión en bornes del inducido mediante el empleo de una de la resistencia del inducido.
3. Variación de la tensión en bornes del inducido, y la corriente en el mismo, mediante combinación de dos resistencias variables en paralelo y en serie con

el inducido. Este método se denomina control de la resistencia serie y shunt del inducido.

4. Empleo de una fuente controlada de tensión de CD variable para modificar la tensión, en bornes del inducido de un motor de excitación independiente. Este método se conoce como control de la tensión de inducido. (Loera, E. S., & Díaz, I. A. F., 1988)

2.20.9 LCD 16X2 LMO16L

Las pantallas de crystal líquido o display LCD (Liquid Crystal Display) tienen la capacidad de mostrar cualquier carácter alfanumérico, permitiendo representar la información que genera cualquier equipo electrónico de una forma fácil y económica. La pantalla consta de una matriz de caracteres (normalmente de 5x7 puntos) distribuidos en una, dos o cuatro líneas de 16, 20 y hasta 40 caracteres cada línea.

Las características generales del módulo LM016L son:

- Consumo muy reducido, del orden de 7.5 mW.
- Pantalla de caracteres ASCII, además de los caracteres japoneses Kanji, caracteres griegos y símbolos matemáticos.
- Desplazamiento de los caracteres hacia la izquierda o a la derecha.
- Memoria de 40 caracteres por línea de pantalla, visualizándose 16 caracteres por línea.
- Movimiento del cursor y cambio de su aspecto.
- Permite que el usuario pueda programar ocho caracteres.
- Pueden ser gobernados de dos formas principales.
 - Conexión con bus de 4 bits.
 - Conexión con bus de 8 bits.

El módulo LM016L tiene 14 pines descritos en la figura 58 y la tabla 26. La alimentación es de +5V. La regulación de contraste mediante el voltaje obtenido al dividir los 5V con una resistencia ajustable de 10 k Ω y aplicárselo al pin V_{LC} a masa fijando el máximo contraste permanentemente. (Palacios, Remiro, López 2009)

Tabla 26. Función de los pines en un LM016

SEÑAL	DEFINICIÓN	PINES	FUNCIÓN
DB0...DB7	Data bus	7..14	Bus de datos
E	Enable	6	E=0, LCD no habilitado. E=1.LCD Habilitado.,
R/W	Read/Write	5	R/W=0, escribe en LCD. R/W=1, lee del LCD.
RS	Register Select	4	R/S=0, modo comando. R/S=1, modo carácter.
V _{LC}	Liquid Crystal driving Voltage	2	Tensión para ajustar el contraste.
V _{DD}	Power Supply Voltage	2	Alimentación de +5.
V _{SS}	Ground	1	Masa.

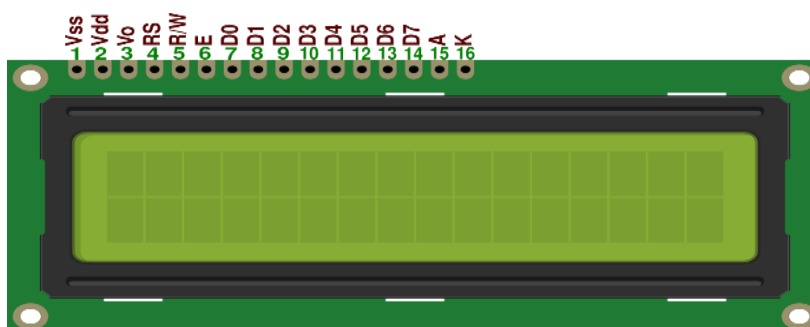


Figura 59. Patillaje del LCD LM016

2.20.10 MOTORES PAP

El motor paso a paso es un dispositivo que produce una rotación en ángulos iguales, denominados pasos, por cada impulso digital que llega a su entrada. Existen diversos tipos de motor paso a paso:

1. Motor paso a paso de reluctancia variable.
2. Motor paso a paso de imán permanente.
3. Motor paso a paso híbrido.

Los siguientes son algunos de los términos más utilizados para especificar motores paso a paso:

1. Fase.
2. Ángulo de paso.
3. Par de retención.

4. Par máximo de enganche.
5. Par máximo de desenganche.
6. Frecuencia de enganche.
7. Frecuencia de desenganche.
8. Velocidad de progresión. (*Bolton 2001*)

CAPÍTULO 3

MARCO

METODOLÓGICO

3.1 TIPO DE ESTUDIO

El presente trabajo es de tipo experimental, se desarrolló un manual de 12 prácticas con su código de programación, circuitos de simulación y de implementación en el Modulo DIGIAC PIC3000.

3.2 FUENTES DE INFORMACIÓN

Para la realización de estas etapas es necesario contar con las teorías relacionadas con la temática de microcontroladores, sensores y actuadores, las cuales son posibles de encontrar en distintas fuentes, dentro de las cuales destacan libros y manuales cuyo contenido esté relacionado con este proyecto. Como fuentes secundarias, el Internet y artículos científicos. Estas fuentes pueden ser vistas de manera detallada en la bibliografía de este documento.

3.3 METODOLOGÍA

Para el desarrollo del presente proyecto se siguió la siguiente metodología:

- ✓ Instalación de software (MPLAB IDE v8.92, PROTEUS 7.8, Oracle VM VirtualBox 5.2.22 y PIC C Compiler)
- ✓ Compilación del código en Software MPLAB IDE v8.92
- ✓ Compilación del código en Software PIC C Compiler
- ✓ Simulación del circuito en software PROTEUS 7.8
- ✓ Implementación de circuitos en el Modulo DIGIAC PIC3000

La metodología mencionada se describe con mayor detalle en los Capítulos 4 y 5.

CAPÍTULO 4.

MARCO

OPERATIVO

4.1 INSTALACIÓN DEL PROGRAMA MPLAB IDE v8.92

Se presenta los pasos a seguir para la instalación del software utilizado así como también los conceptos básicos para el funcionamiento e importancia del presente proyecto.

Si aún no cuenta con el archivo MPLAB IDE 8.92.ZIP, haga el download directamente del sitio <https://pic-microcontroller.com/mplab-ide-v8-92-free-download/> elija la opción “download” y después el archivo indicado.

Abrir el archivo MPLAB IDE v8.92.ZIP para descomprimir. Dentro de la carpeta MPLAB IDE v8.92 elige el setup y selecciona “next” tal como se muestra en la figura 60.

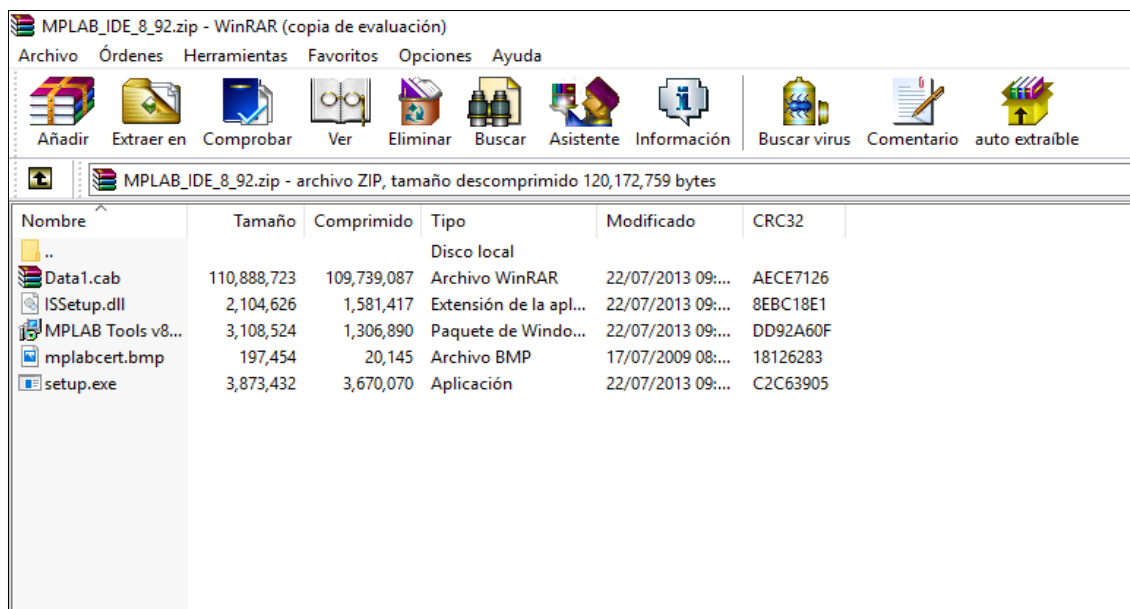


Figura 60. Descomprimir el archivo setup



Figura 61. Bienvenido a la instalación del programa MPLAB

Antes de ver la nota 1. Selecciona la opción “I accept” y luego oprima “next”, tal como se muestra en la figura 62.



Figura 62. Términos y condiciones de Microchip

Selecciona instalación “complete” tal como se muestra en la figura 63.

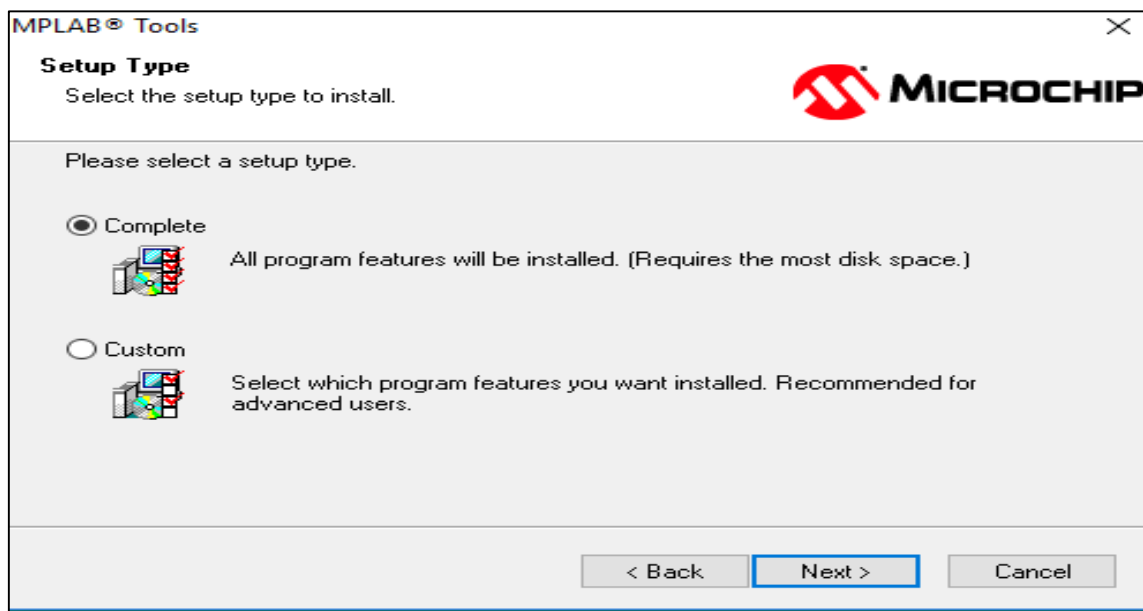


Figura 63. Instalación de todas las funciones del programa

No modifique el path de instalación y solo oprima “next” como se muestra en la figura 64.

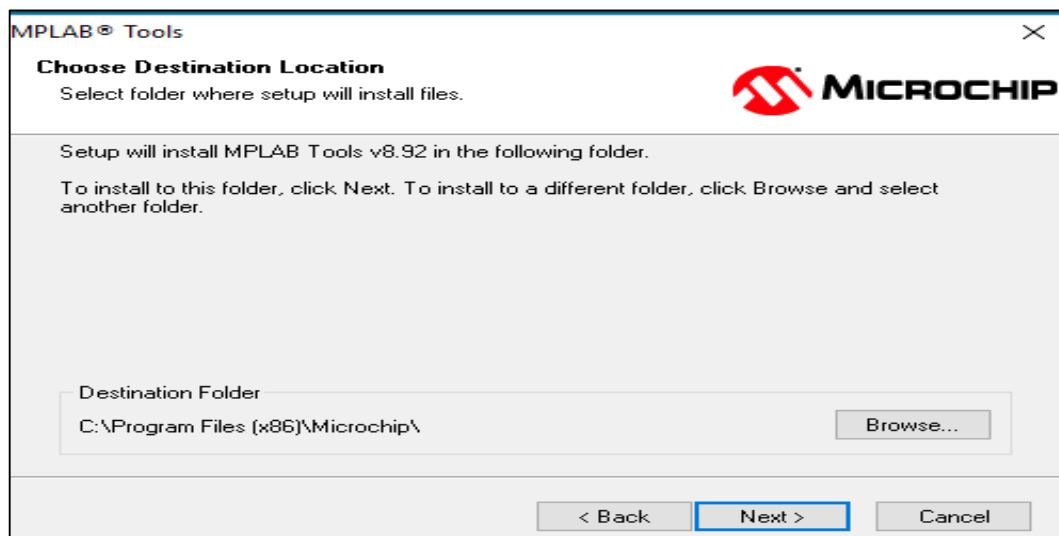


Figura 64. Selección del directorio de instalación

Antes de ver la nota 2. Selecciona la opción “I accept” y en seguida oprima “next” como se muestra en la figura 65.

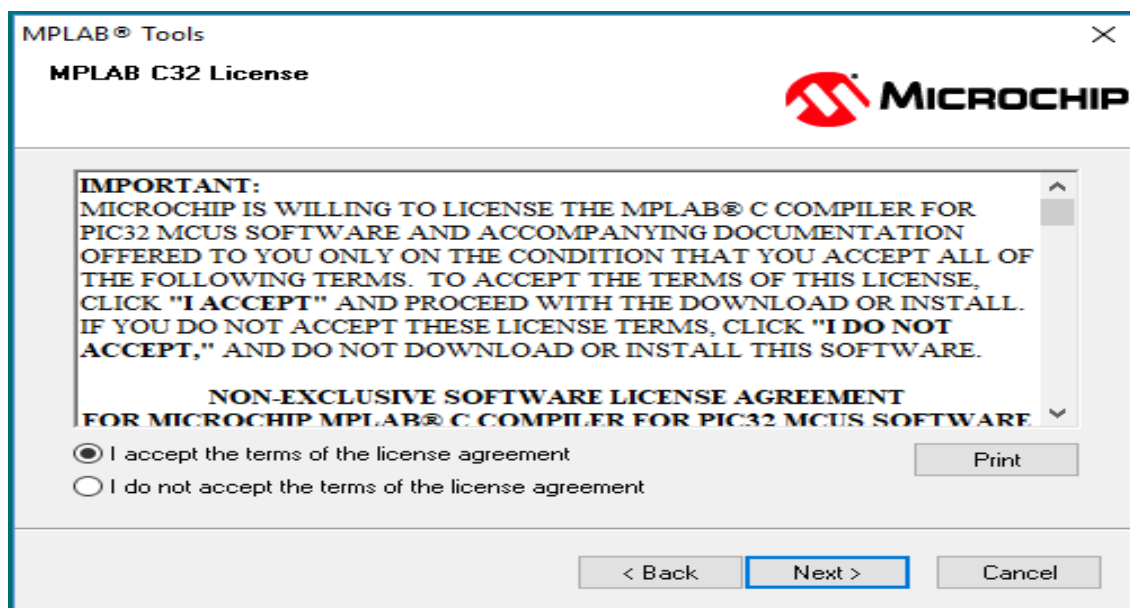


Figura 65. Términos de licencia Microchip

En esta ventana se muestran los componentes que se instalarán, oprima “next” sin hacer modificaciones tal como se muestra en la figura 66.

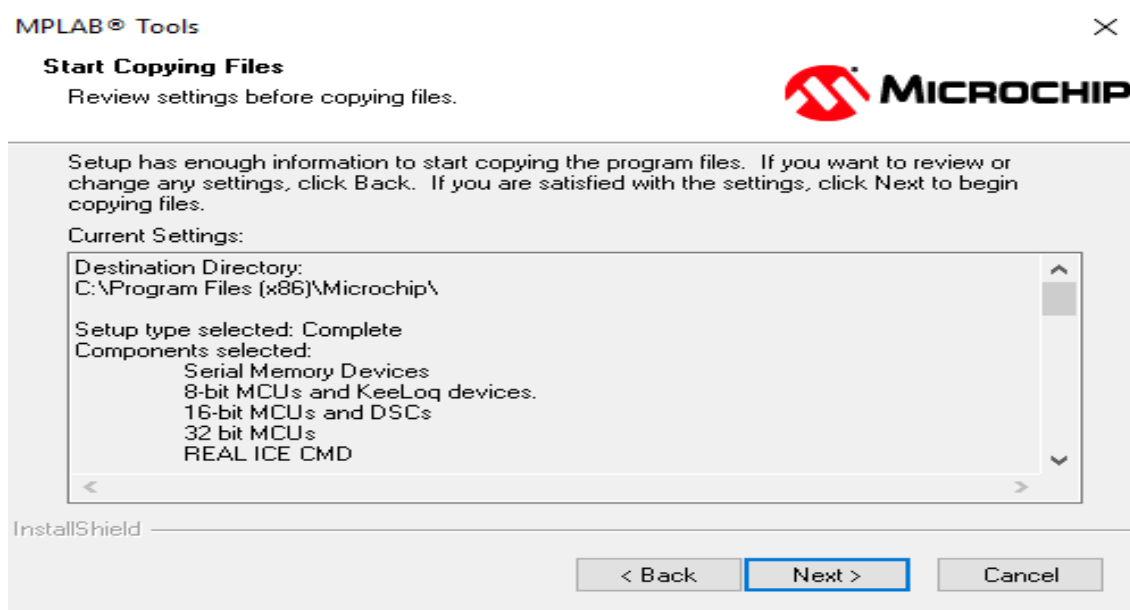


Figura 66. Componentes que se instalarán en el programa

El proceso de instalación ha iniciado, espera unos minutos como se muestra en la figura 67.

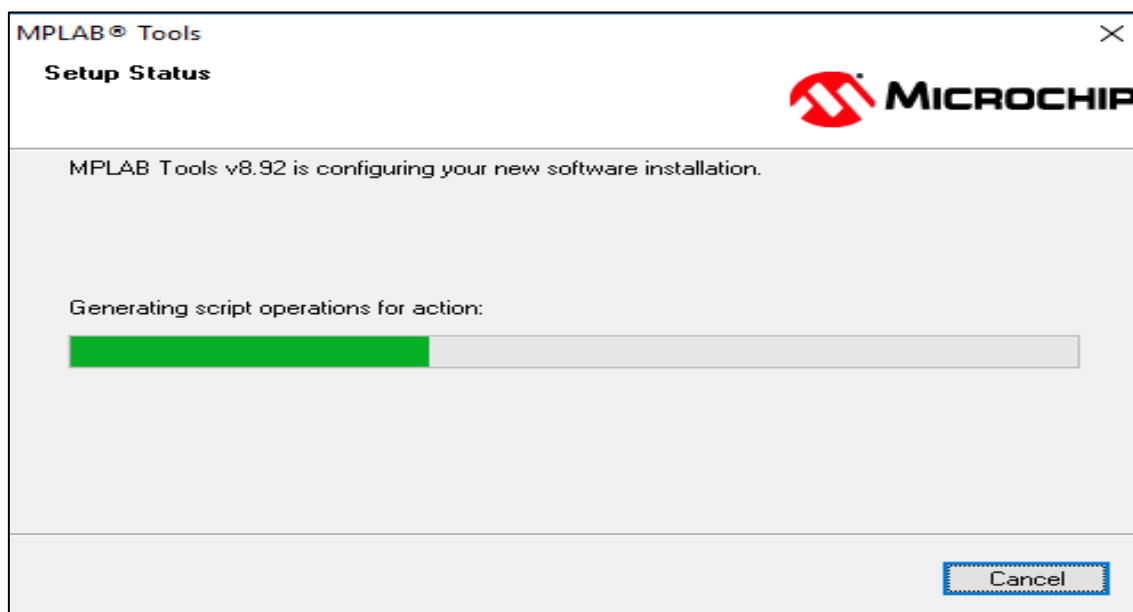


Figura 67. Proceso de instalación

La ultima ventana muestra la instalacion finalizada por ultimo selecciona la opción finish como se muestra en la figura 68.

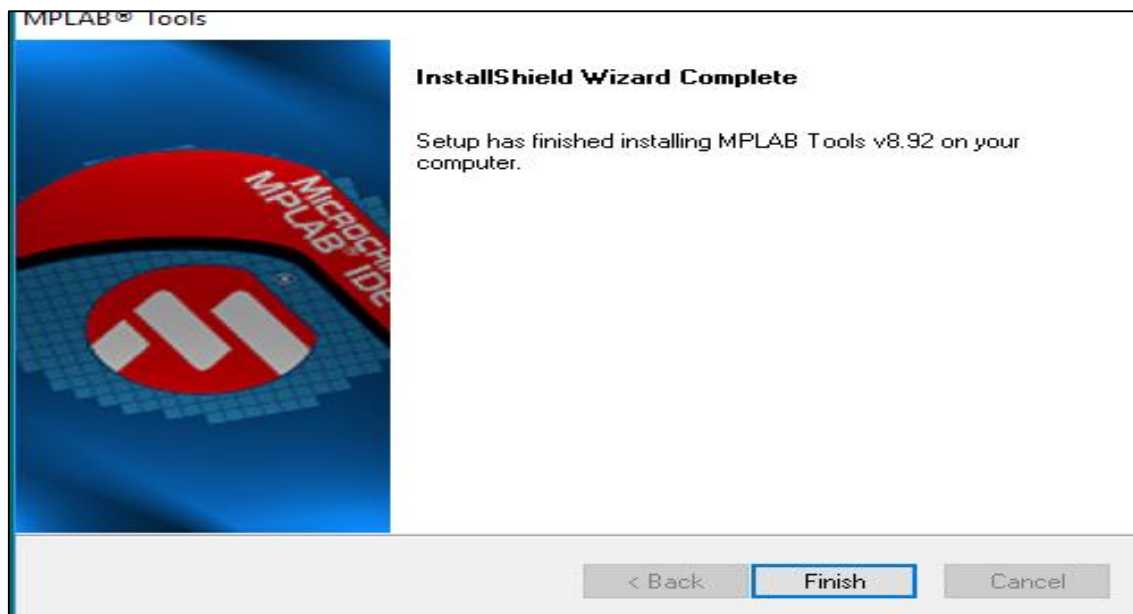


Figura 68. La instalación se ha terminado

4.2 INTALACIÓN DE PROTEUS 7.8

Nota el antivirus debe estar desactivado.

Se ejecuta el programa y oprima el botón next como se muestra en la figura 69.

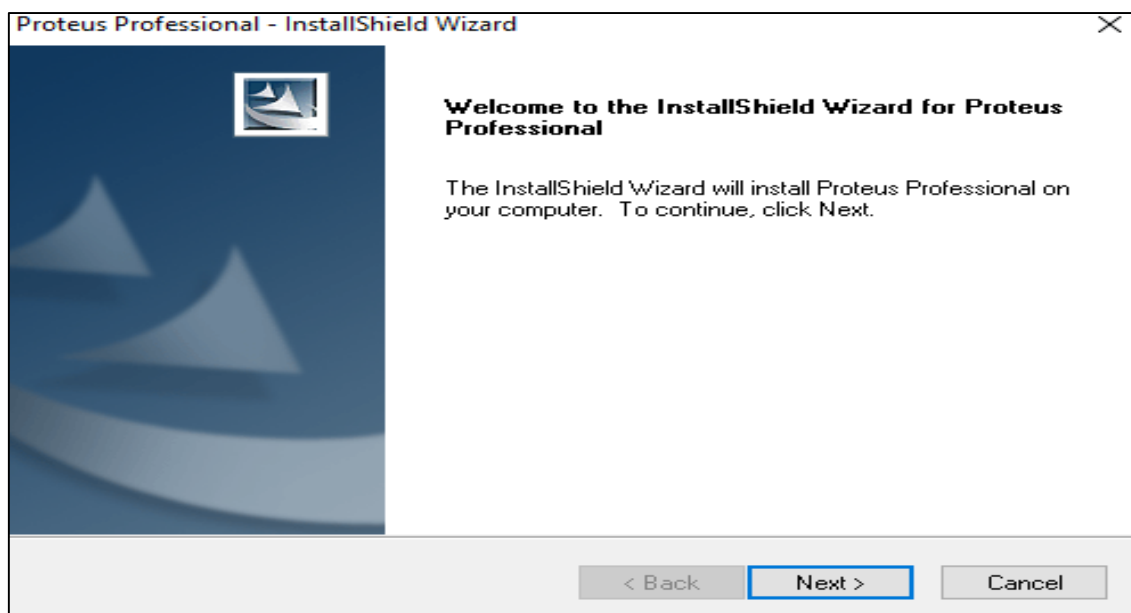


Figura 69. Mensaje de bienvenida a la instalación del programa Proteus 7.8

Le das en “ Use a locally installed licence key” y después oprima el botón next como se muestra en la figura 70.

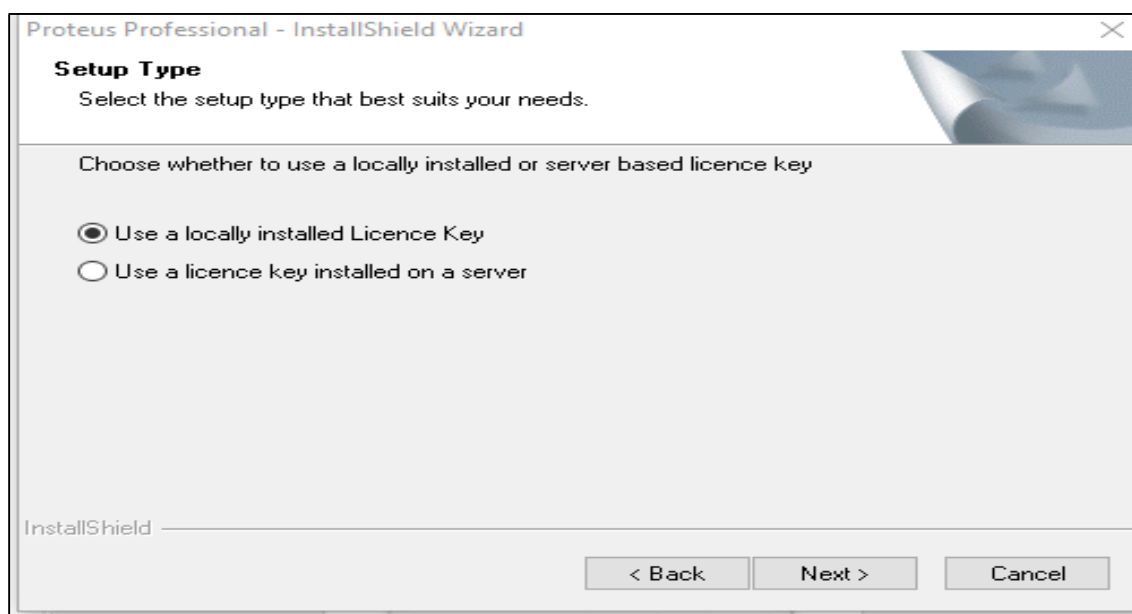


Figura 70. Configuración de licencia

Cuando se muestra esta pantalla “license Agreement “das click en next como se muestra en la figura 71.

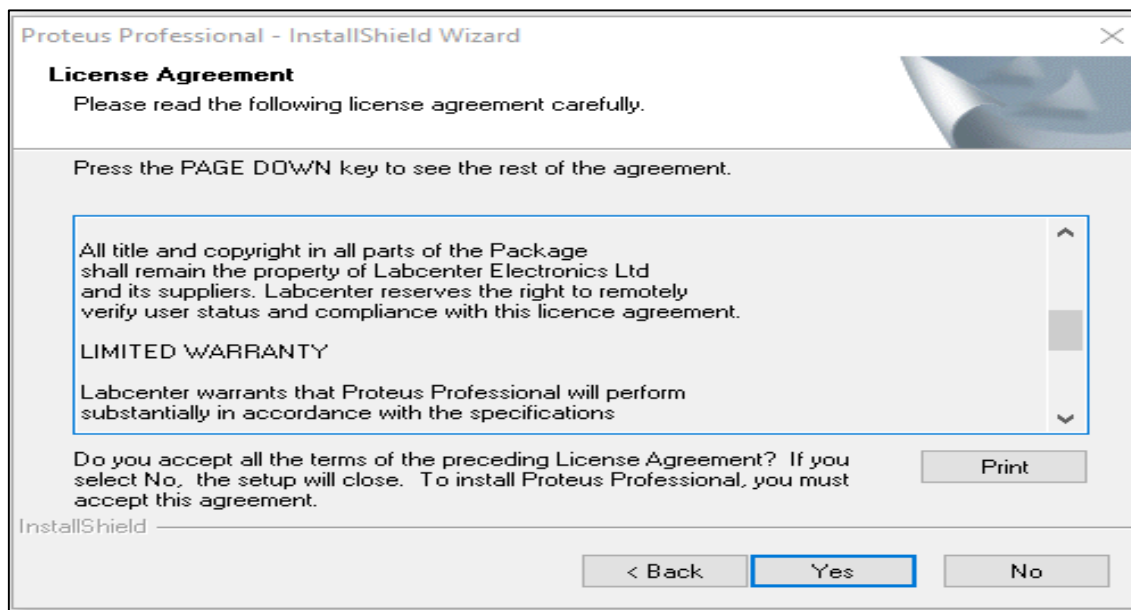


Figura 71. Términos de licencia

En la pantalla muestra que la llave está instalado das click en next como se muestra en la figura 72.

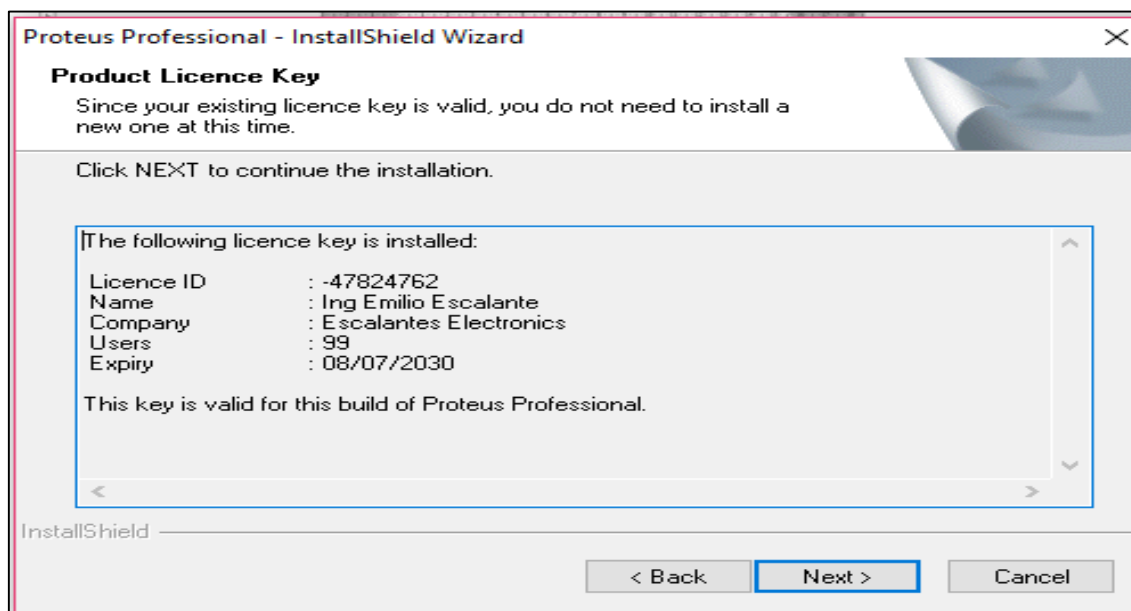


Figura 72. La licencia esta instalado

Selecciona el lugar donde deseas guardar el programa, das click en next como se muestra en la figura 73.

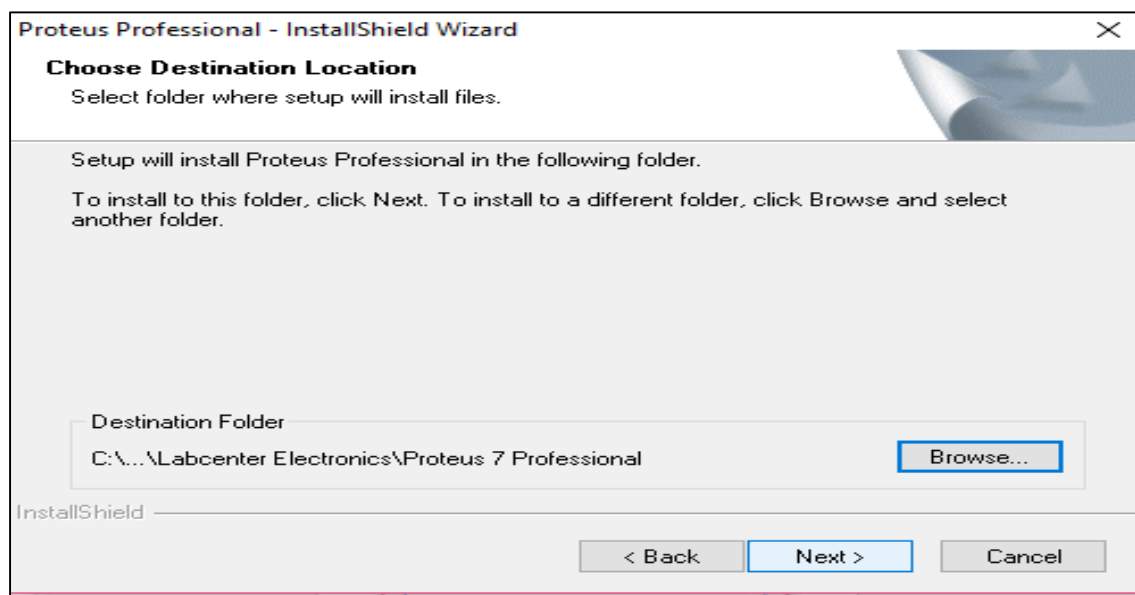


Figura 73. Selección del directorio de instalación

Ahora en seleccionando las funciones es “opcional” le das click en next como se muestra en la figura 74.

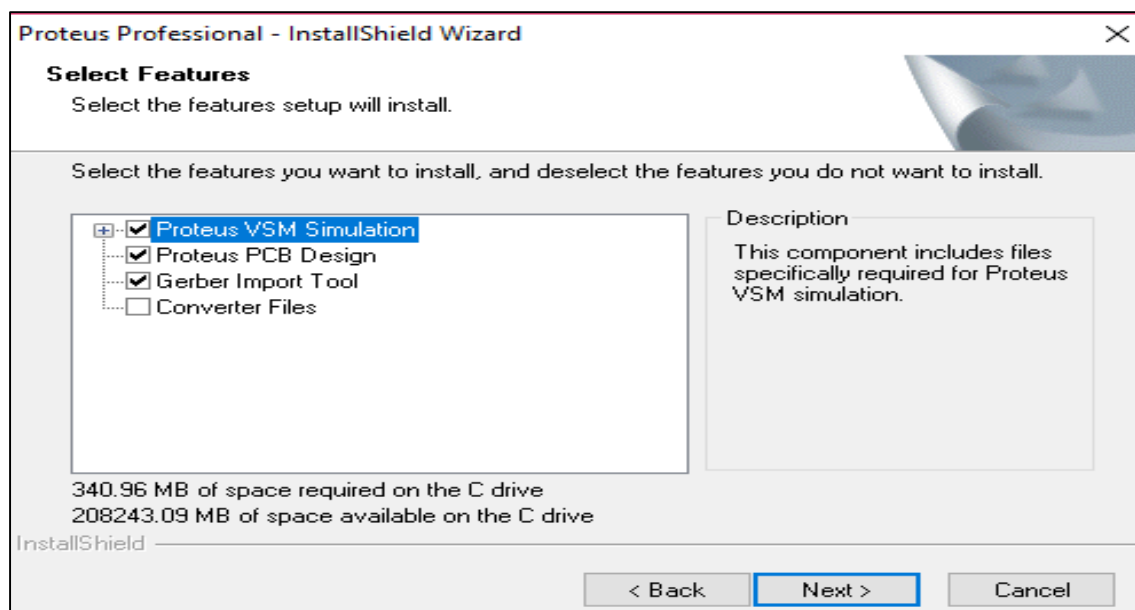


Figura 74. Selección de componentes a instalar

Selecciona el folder donde usaran el programa es “opcional” le das click en next tal como se muestra en la figura 75.

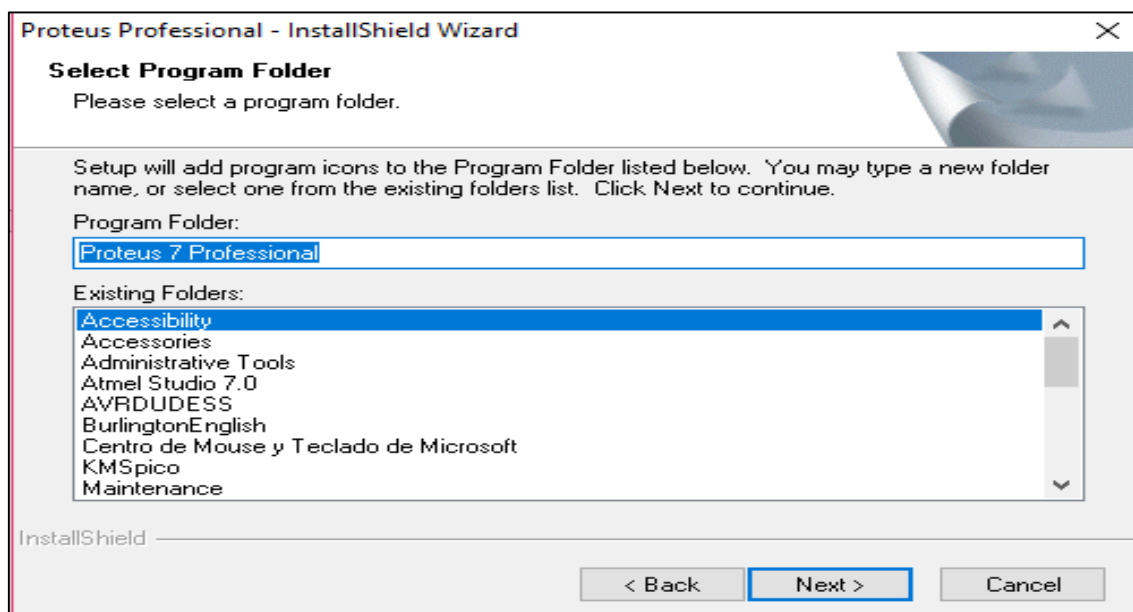


Figura 75. Selección del archivo donde se utilizara el programa

Se inicia el proceso de instalación como se muestra en la figura 76.

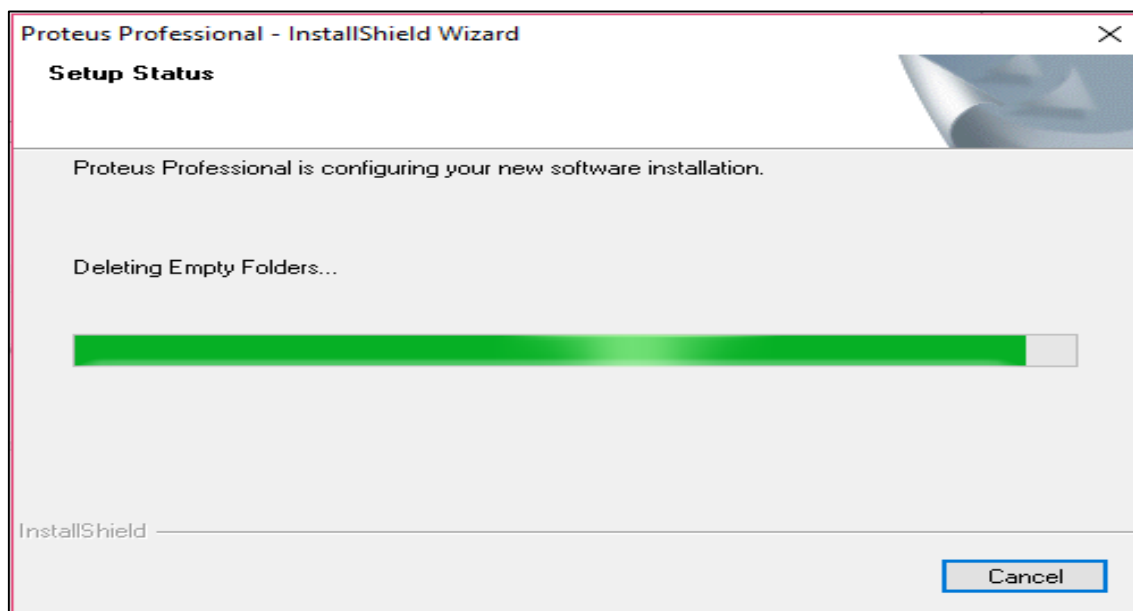


Figura 76. Proceso de instalación

Para finalizar le das en finish como se muestra en la figura 77.

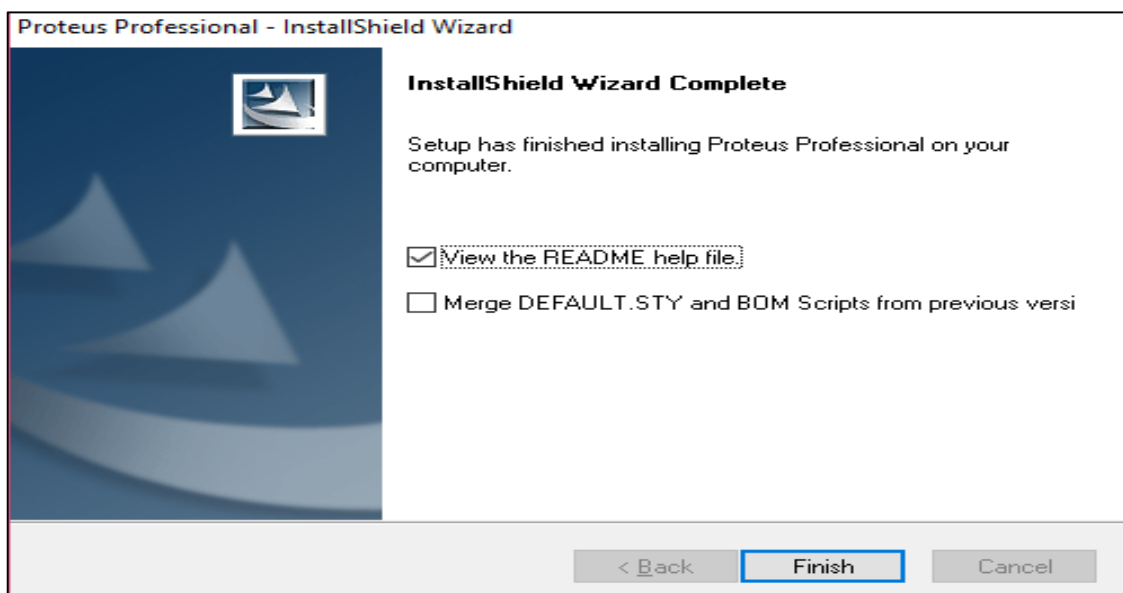


Figura 77. La instalación ha terminado

Abres la carpeta crack y dentro de ella elige la opción “LXK Proteus 7.8 SP2 RUS v1.0.0” como se muestra en la siguiente figura 78.

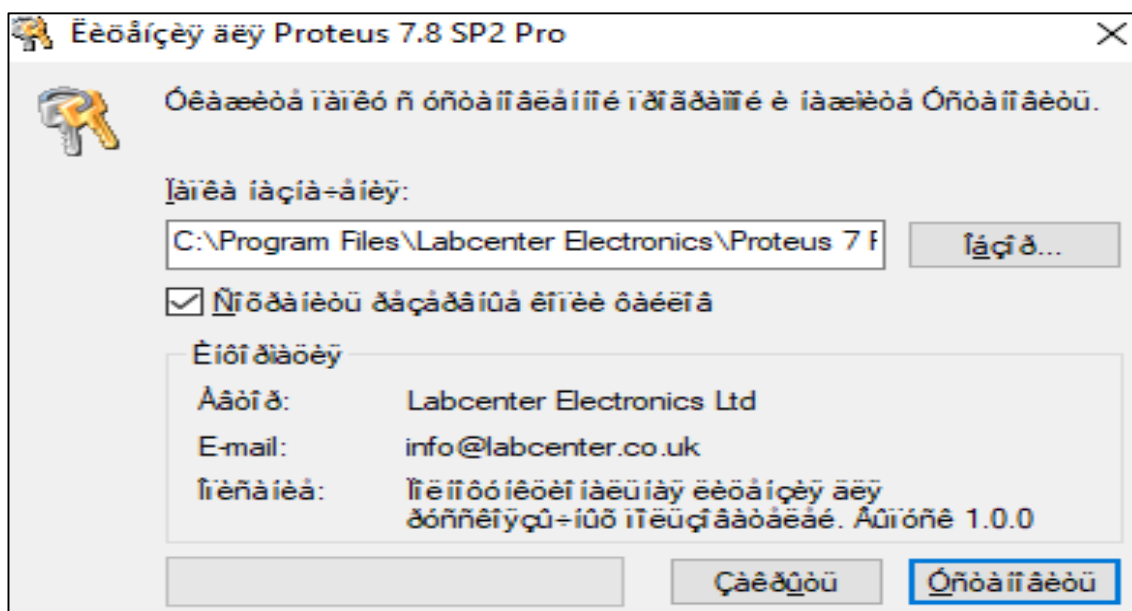


Figura 78. Selección de la carpeta “LXK Proteus 7.8 SP2 RUS v1.0.0”

Se terminó el proceso de instalación das click en “aceptar” como se muestra en la figura 79.

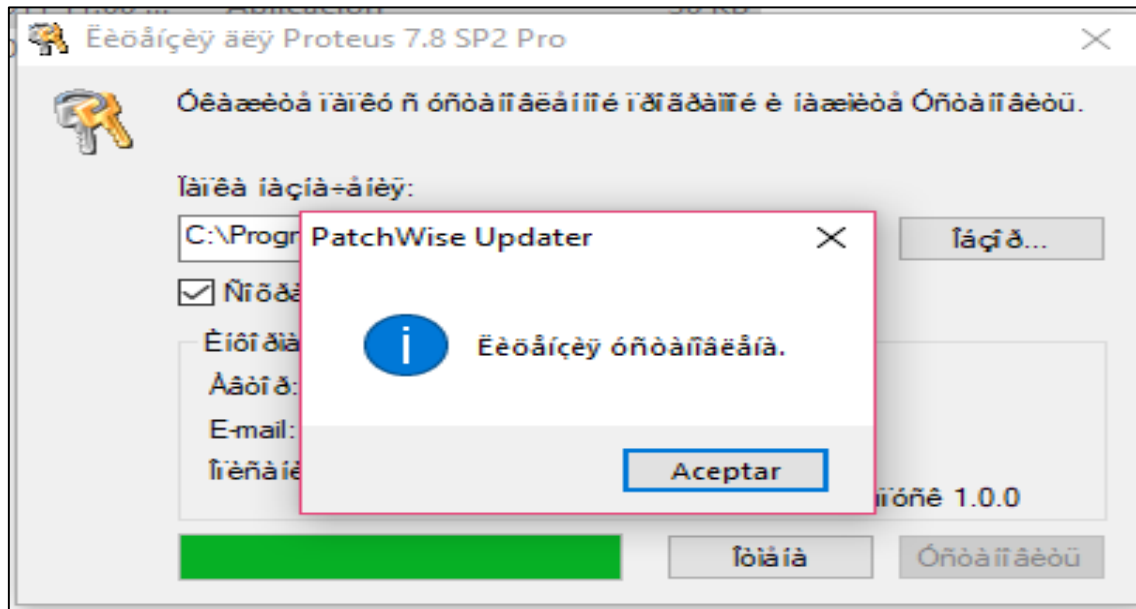


Figura 79. Proceso de instalación terminado

4.3 INSTALACIÓN DE LA MÁQUINA VIRTUAL

Se eligió utilizar virtual box para instalar el software controlador para dicho módulo ya que el software no cuenta con las compatibilidades del actual sistema operativo Windows 10. Este es el sitio web <https://www.virtualbox.org/> donde se descargará el archivo de instalación Virtual Box, haz click en windows hosts y comenzara la descarga.



Figura 80. Descarga del programa Virtual Box

Abres el archivo VirtualBox EXE. Dirígite a la ubicación donde se descargó el archivo .EXE y haz doble click en él y se abrirá la instalación de VirtualBox como se muestra en la siguiente figura 81.

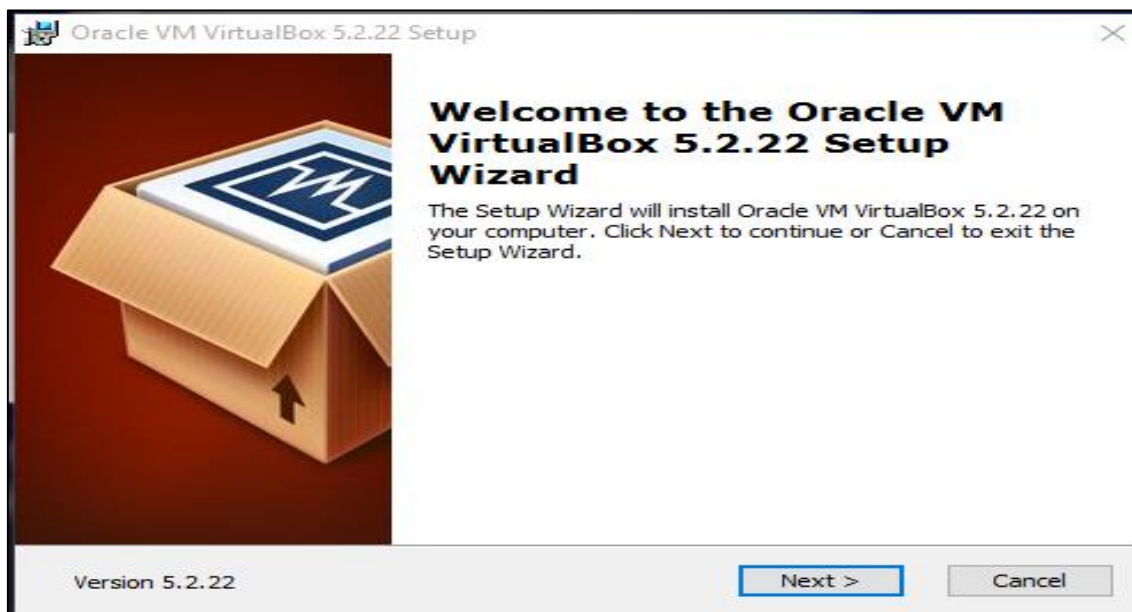


Figura 81. Mensaje de bienvenida Virtual Box

En la siguiente figura 82 das click en “Install”.

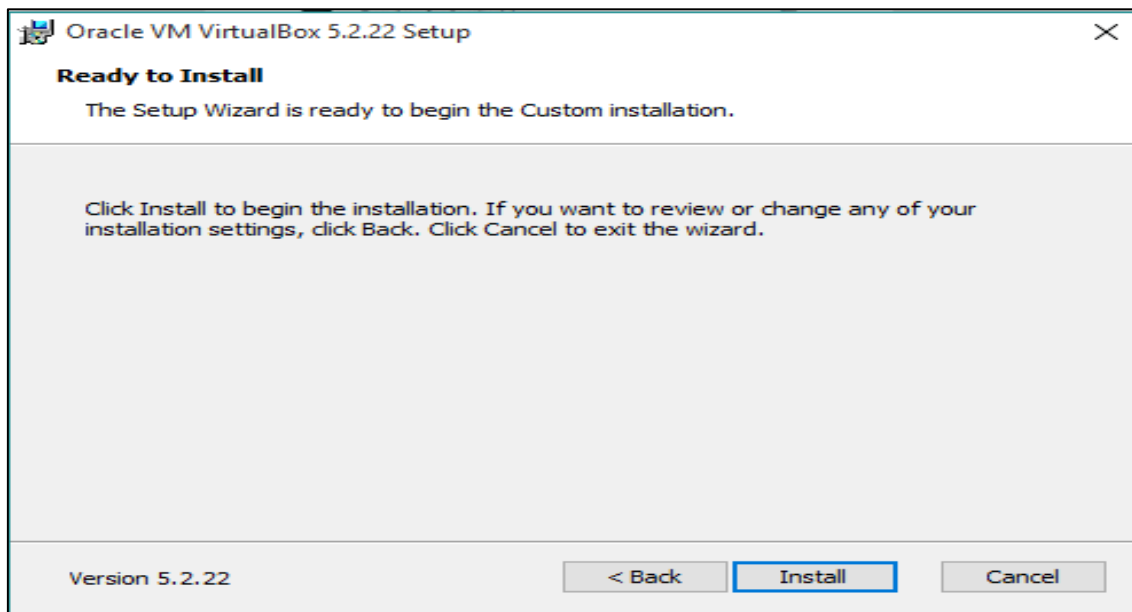


Figura 82. Instalación de la máquina virtual

Selecciona la forma que desea que se instalen las características de instalación como en la siguiente figura 83.

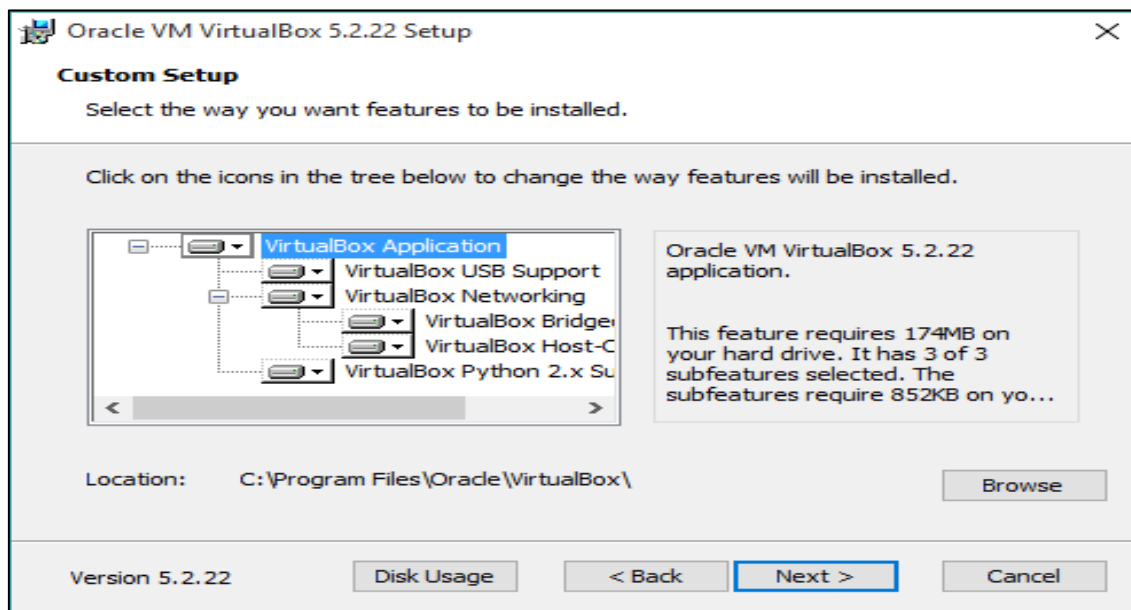


Figura 83. Selección de funciones con las cuales se instalara el programa

Selecciona las siguientes casillas y oprima next como se muestra en la siguiente figura 84.

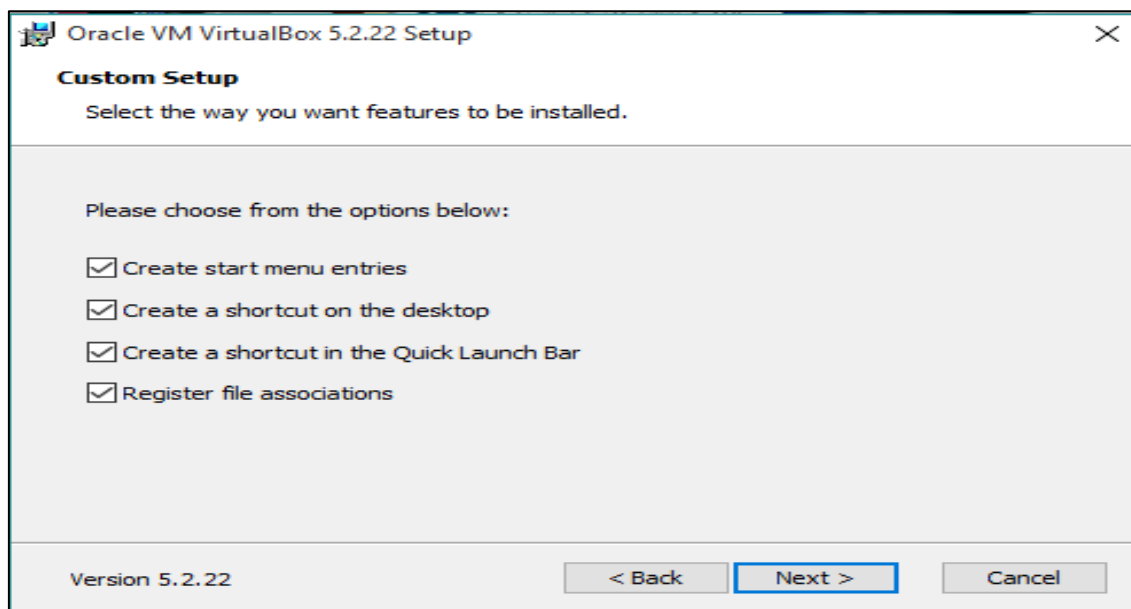


Figura 84. Selección de las características de instalación

La instalación de la función red Oracle se desconectará de la red temporalmente le das click en “yes” como se muestra en la figura 85.

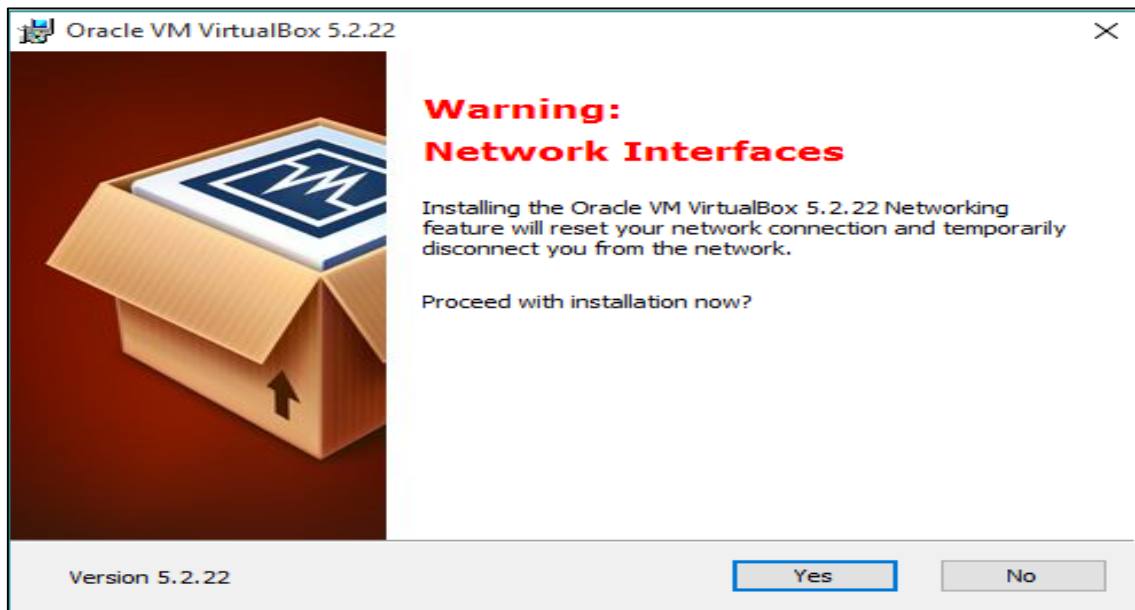


Figura 85. Procede en la instalación

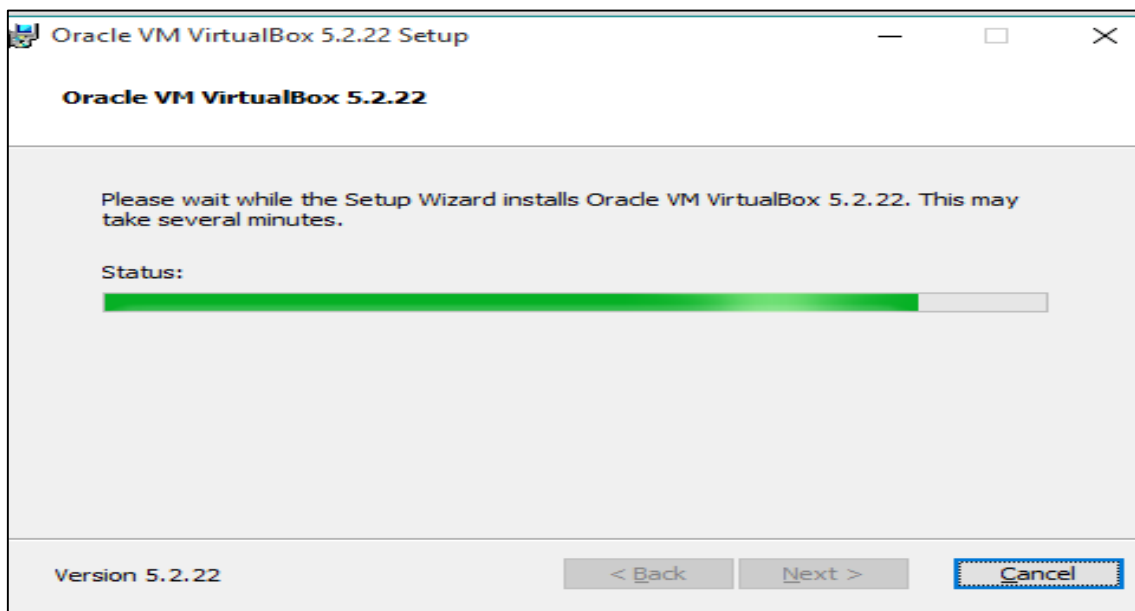


Figura 86. Instalación en progreso

LISTA DE MATERIALES UTILIZADOS

- 1 programador MASTER PROG
- 1 Push boton
- 1 Resistencia de 330 Ω
- 1 display LCD 16x2 LM016L
- 2 Potenciómetros 10 k Ω y 50 k Ω
- 1 driver L293D
- 1 motor CD
- 1 motor paso a paso bipolar
- 1 servo motor sg90
- 3 transistores BD135
- 6 resistencias 10k Ω
- 2 bombas de agua chisgueteros
- 1 módulo de 4 relevadores
- 4 Metros de cable de proboard
- 1 Sensor CNY70
- 1 Fotorresistencia
- 1 Resistencia 220 Ω
- 1 Resistencia de 18 k Ω
- 1 Integrado LM358
- 1 sensor ultrasónico SFR04

CAPÍTULO 5.

RESULTADOS

5.1 PRÁCTICA 1 CONFIGURACIÓN DE ENTRADAS Y SALIDAS

Las siguientes prácticas fueron adaptadas del Libro Microcontrolador PIC16F84 (Palacios, Remiro, López 2009)

A continuación, se muestra el programa donde el puerto A se configura como entrada y el puerto B se configura como salida.

En la entrada:

- Un interruptor cerrado representa un “0” lógico.
- Un interruptor abierto representa un “1” lógico.

En la salida:

- Un LED apagado representa un “0” lógico
- Un LED encendido representa un “1” lógico

PROGRAMACIÓN LENGUAJE ENSAMBLADOR

```
.*****Práctica 1 configuración de entradas y salidas *****
;
; CONFIGURACIÓN *****
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST    P=16F84A                ; Procesador.
INCLUDE<P16F84A.INC>            ; Definición de los operandos utilizados.
; PROGRAMA *****
                ORG      0        ; Comienza en la dirección 0 de memoria de programa.
Inicio  bsf      STATUS, RP0      ; Pone a 1 el bit 5 del STATUS. Acceso al Banco 1.
                clrf     TRISB     ; Las líneas del Puerto B se configuran como salidas.
                movlw    b'11111111'
                movwf    TRISA     ; Las líneas del Puerto A se configuran como entradas.
                bcf      STATUS, RP0 ; Pone a 0 el bit 5 de STATUS. Acceso al Banco 0.

Principal
                movf     PORTA, W   ; Lee el Puerto A.
                movwf    PORTB      ; El contenido de W se visualiza por el Puerto B.
                goto     Principal  ; Crea un bucle cerrado.
                END              ; Fin del programa.
```

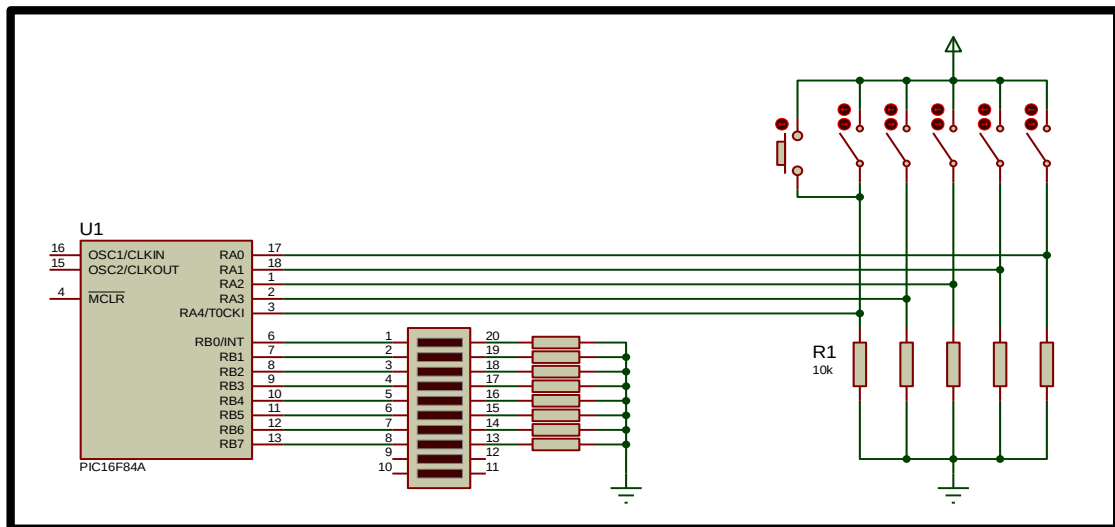
```
#include <ConfiguracionIO.h>

#BYTE TRISB = 0x86 //Se define los registros TRISA y TRISB con las direcciones de memoria
#BYTE TRISA = 0x85

void main()
{
    int a;

    TRISA=0x1F; //las 5 líneas del puerto A se configuran como entradas
    TRISB= 0x00; //las líneas del puerto B se configuran como salidas

    while(TRUE)
    {
        a=Input_A(); // lee el puerto A y el dato lo guarda en la variable
        Output_B(a); //El contenido del Puerto A se visualiza en el Puerto B
    }
}
```



140



Figura 88. Implementación de la práctica configuración de entradas y salidas

5.2 PRÁCTICA 2 DISPLAY DE 7 SEGMENTOS

El display de 7 segmentos es un dispositivo opto electrónico en este caso el display es ánodo común y se activa con ceros es decir el microcontrolador le asigna a cada segmento un cero lógico. Mediante el uso de tablas es posible gobernar la información que aparece en un display de 7 segmentos.

En un display de 7 segmentos conectado al Puerto B se visualiza la cantidad leída por el Puerto A.

PROGRAMACIÓN LENGUAJE ENSAMBLADOR

```
*****Práctica 2 Display de 7 Segmentos *****
; CONFIGURACIÓN*****
LIST          P=16F84A
INCLUDE       <P16F84A.INC>
__CONFIG     _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
; PROGRAMA*****
          ORG    0                      ; El programa comienza en la dirección 0.
Inicio bsf          STATUS,RP0          ; Acceso al Banco 1.
          clrf    PORTB                 ; Las líneas del Puerto B se configuran como salida.
          movlw   b'00011111'           ; Las 5 líneas del Puerto A se configuran como entrada.
          movwf   PORTA
          bcf     STATUS,RP0            ; Acceso al Banco 0.
Principal
          movf    PORTA,W                ; Lee la entrada
          andlw   b'00001111'           ; Máscara para quedarse con el valor de las
                                          ; entradas correspondientes al nibble bajo.
          call    Binario_a_7Segmentos  ; Convierte código binario a 7 segmentos del display.
          movwf   PORTB                 ; Resultado se visualiza por el puerto de salida.
          goto    Principal
; Subrutina "Binario_7Segmentos" -----
;
Binario_a_7Segmentos                      ; Tabla para display de 7 segmentos.
          addwf   PCL,F
Tabla   retlw    b'11000000'             ; El código 7 segmentos para el "0".
          retlw   b'11111001'           ; El código 7 segmentos para el "1".
```

retlw	b'10100100'	; El código 7 segmentos para el "2".
retlw	b'10110000'	; El código 7 segmentos para el "3".
retlw	b'10011001'	; El código 7 segmentos para el "4".
retlw	b'10010010'	; El código 7 segmentos para el "5".
retlw	b'10000010'	; El código 7 segmentos para el "6".
retlw	b'11111000'	; El código 7 segmentos para el "7".
retlw	b'10000000'	; El código 7 segmentos para el "8".
retlw	b'10011000'	; El código 7 segmentos para el "9".
retlw	b'10001000'	; El código 7 segmentos para la "A".
retlw	b'10000011'	; El código 7 segmentos para la "B".
retlw	b'10100111'	; El código 7 segmentos para la "C".
retlw	b'10100001'	; El código 7 segmentos para la "D".
retlw	b'10000110'	; El código 7 segmentos para la "E".
retlw	b'10001110'	; El código 7 segmentos para la "F".
END		; Fin del programa.

PROGRAMACIÓN EN LENGUAJE C

```
#include <Display7seg.h>
#BYTE PORTB = 0x06
#BYTE TRISB = 0x86
// Tabla con los códigos en Hexadecimal del Display 7 segmentos

Byte                                                    const
display[16]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xB8,0x80,0x98,0x88,0x83,0xC6,0xA1,0x86,0x8E
};
void main()
{
    int a;
    TRISB= 0x00; //las líneas del puerto B se configuran como salidas
    set_tris_A(0x1F); //las 5 líneas del puerto A se configuran como entradas

    while(TRUE)
    {
        a=Input_A()&(0x0F); // lee el puerto A y aplica una mascara para quedarse con el valor de las
        entradas del nibble mas bajo
        switch (a)
        {
```

```

case 0:Output_B(display[a]); //El contenido del Puerto A se visualiza en el Puerto B
    break;

                                case 1:Output_B(display[a]);

    break;
case 2:Output_B(display[a]);
    break;
case 3:Output_B(display[a]);
    break;
case 4:Output_B(display[a]);
    break;
case 5:Output_B(display[a]);
    break;
case 6:Output_B(display[a]);
    break;
case 7:Output_B(display[a]);
    break;
case 8:Output_B(display[a]);
    break;
case 9:Output_B(display[a]);
    break;
case 10:Output_B(display[a]);
    break;
case 11:Output_B(display[a]);
    break;
case 12:Output_B(display[a]);
    break;
case 13:Output_B(display[a]);
    break;
case 14:Output_B(display[a]);
    break;
case 15:Output_B(display[a]);
    break;
case 16:Output_B(display[a]);
    break;
default:Output_B(0x00);
}
}
}

```

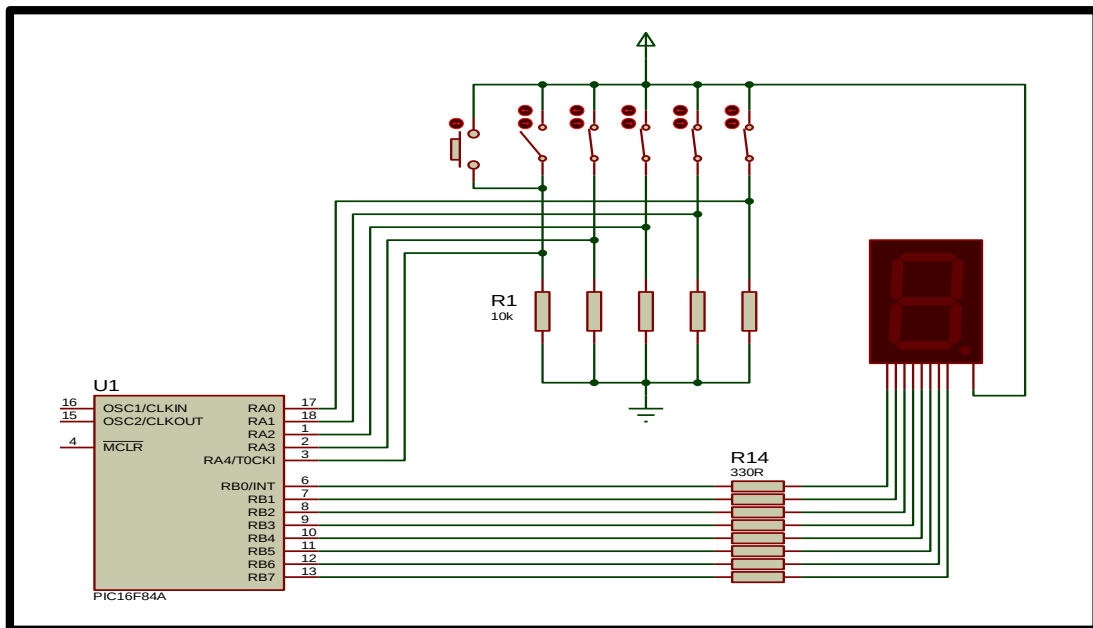


Figura 89. Simulación de la práctica display de 7 segmentos

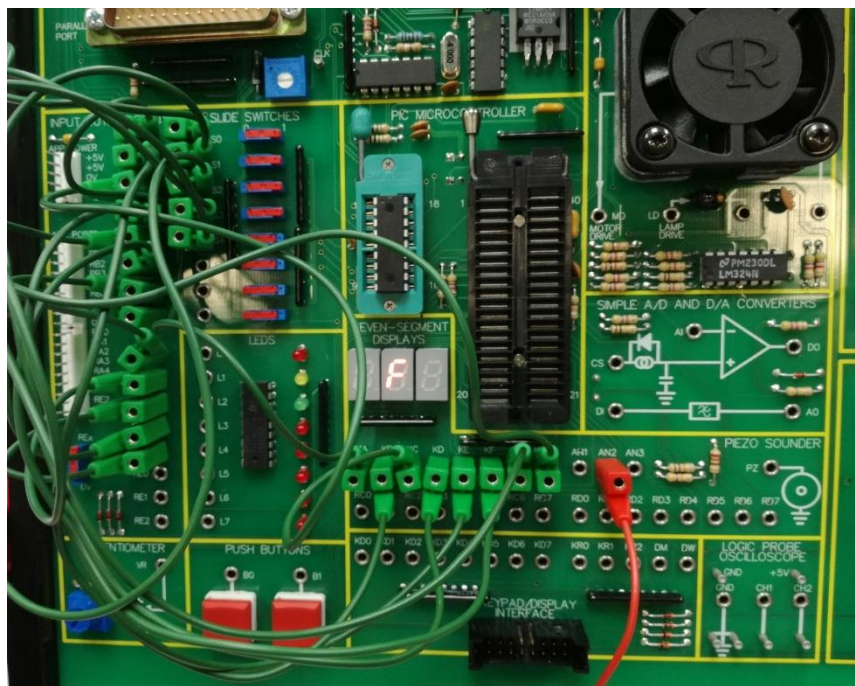


Figura 90. Implementación de la práctica display de 7 segmentos

5.3 PRÁCTICA 3 MENSAJE

En el siguiente programa el módulo LCD visualiza un mensaje largo (más de 16 caracteres) que se desplaza a lo largo de la pantalla. Se utiliza la subrutina LCD_MensajeMovimiento de la librería LCD _MENS.INC.

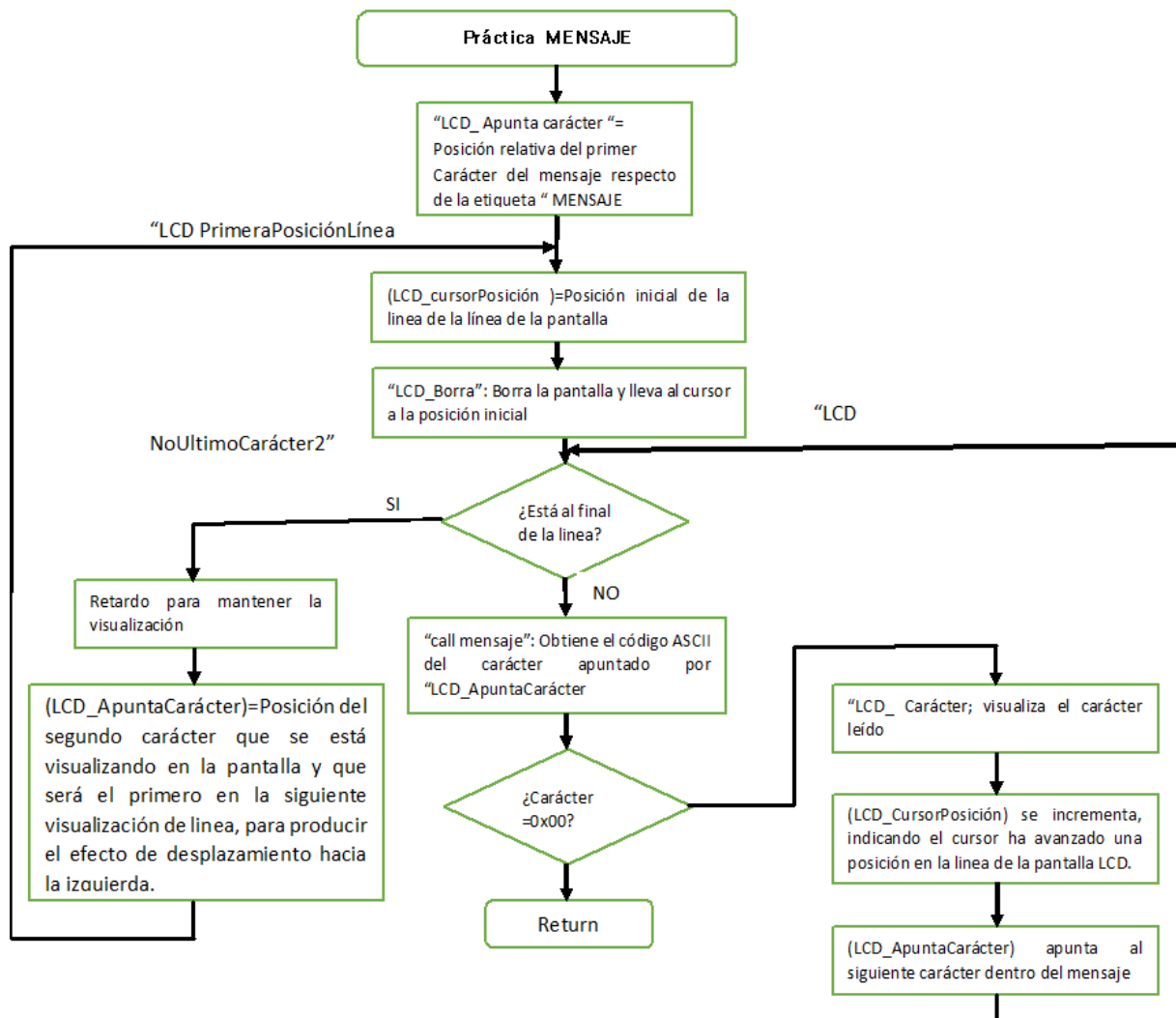


Figura 91. Subrutina de visualización mensajes en movimiento

PROGRAMACIÓN LENGUAJE ENSAMBLADOR

*****Práctica 3 Mensaje*****

; CONFIGURACIÓN *****

```
LIST          P=16F84A

INCLUDE        <P16F84A.INC>

__CONFIG      _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

CBLOCK 0x0C

ENDC
```

; PROGRAMA *****

```
ORG 0
```

Inicio

```
call LCD_Inicializa          ; Prepara la pantalla.
```

Principal

```
movlw Mensaje0              ; Apunta al mensaje.
```

```
call LCD_MensajeMovimiento
```

```
goto Principal              ; Repite la visualización.
```

; "Mensajes" -----

;

Mensajes

```
addwf PCL,F
```

Mensaje0 ; Posición inicial del mensaje.

```
DT " " ; Espacios en blanco al principio para mejor
```

```
DT "Estudia ELECTRONICA " ; visualización.
```

```
DT "en el ITSTa."
```

```
DT " ", 0x0 ; Espacios en blanco al final.
```

;

```
INCLUDE <LCD_MENS.INC> ;Subrutina LCD_MensajeMovimiento.
```

```
INCLUDE <LCD_4BIT.INC> ; Subrutinas de control del LCD.
```

```
INCLUDE <RETARDOS.INC> ; Subrutinas de retardos.
```

```
END
```

PROGRAMACIÓN EN LENGUAJE C

```
#include <lcdtextoR.h>
```

```
#use delay(clock=4000000)
```

```
//Conexiones del modulo LCD
```

```
#define LCD_ENABLE_PIN PIN_A2
```

```
#define LCD_RS_PIN PIN_A0
```

```
#define LCD_RW_PIN PIN_A1
```

```
#define LCD_DATA4 PIN_B4
```

```
#define LCD_DATA5 PIN_B5
```

```
#define LCD_DATA6 PIN_B6
```

```
#define LCD_DATA7 PIN_B7
```

```
//Fin de las conexiones del modulo LCD
```

```
#include <lcd.c>
```

```
void main()
```

```
{
```

```
    lcd_init();          // Inicializa el modulo LCD
```

```
    printf(lcd_putc, "\f"); // limpia el LCD
```

```
    while(TRUE)
```

```
    {
```

```
        for(int i=16;i>=1;i--) // Escribe el texto en LCD como marquesina
```

```
        {          // se recorre de derecha izquierda
```

```
            lcd_gotoxy(i,1); // Escribe "Curso formativo" en el modulo LCD en la columna la fila 1
```

```
            printf(lcd_putc, "Curso Formativo ");
```

```

    delay_ms(100);

}

printf(lcd_putc, "%f"); // limpia el LCD

}

}

```

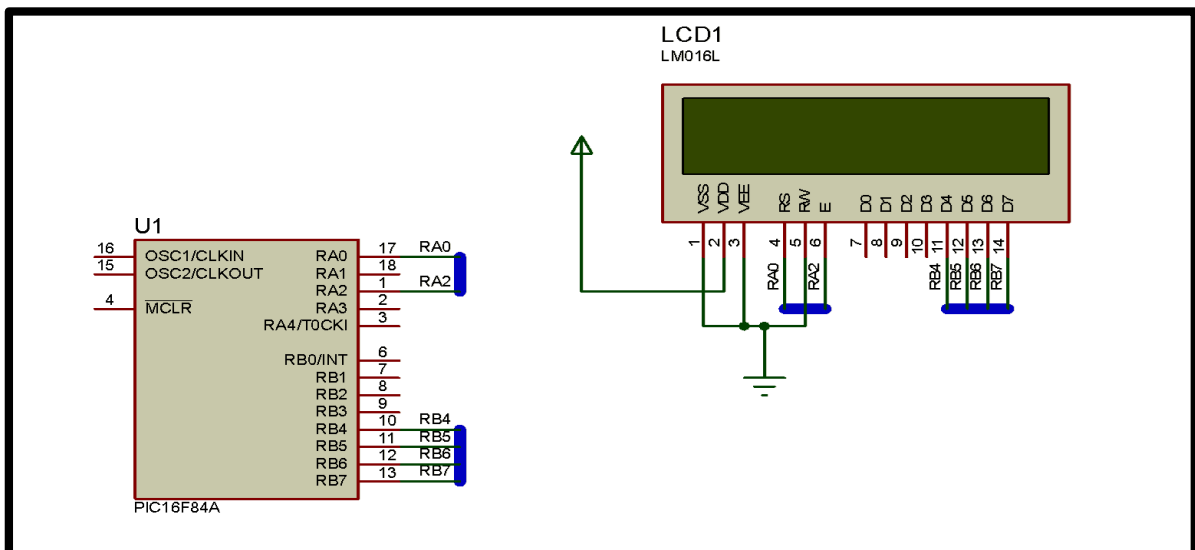


Figura 92. Simulación de la práctica Mensaje

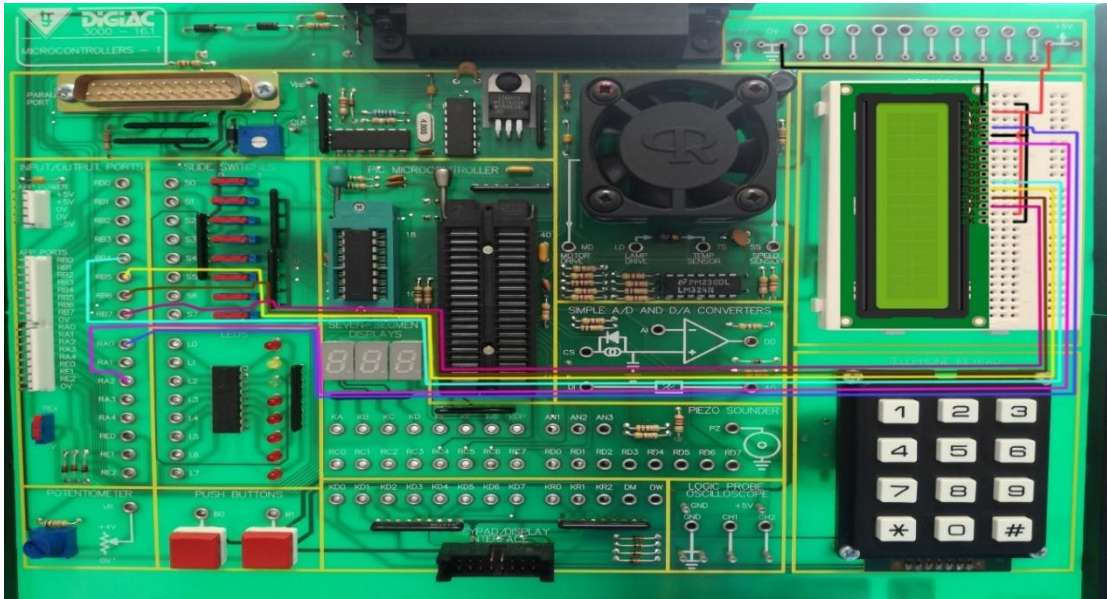


Figura 93. Esquema de la práctica Mensaje

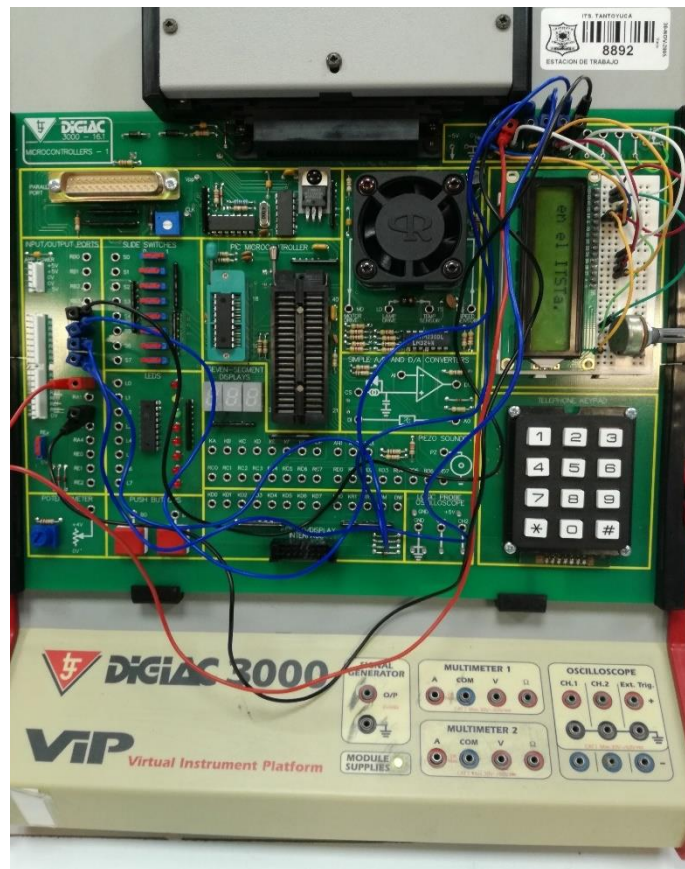


Figura 94. Implementación de la práctica Mensaje

5.4 PRÁCTICA 4 MOTOR DC

La línea RA0 controla la puesta en marcha del motor, Si RA0 es 0 el motor se pone en marcha y si RA0 es 1 se detiene. La posición del interruptor conectado a RA4 controla el sentido de giro que por supuesto depende de la polaridad de conexión del motor, si se intercambian los terminales del motor los giros serán en sentido contrario el organigrama de este programa se muestra en la figura 96.

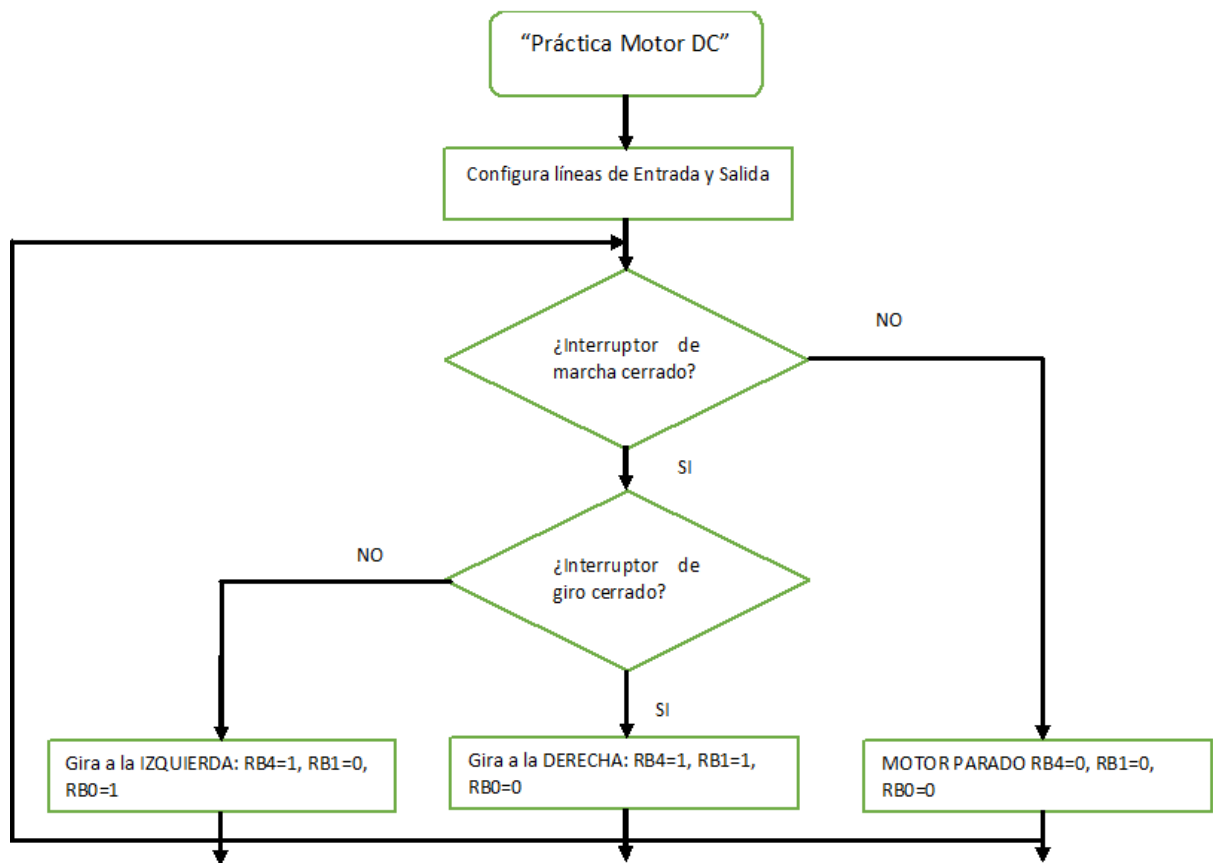


Figura 95. Organigrama del programa Motor DC

Programa de control para un motor de corriente continua en funcionamiento y sentido de giro. Con RA0=0, el motor se pone en marcha y su sentido de giro dependerá del valor que tenga RA4.

PROGRAMACIÓN LENGUAJE ENSAMBLADOR

```

;*****Práctica 4 Motor DC*****
; CONFIGURACIÓN*****

        LIST          P=16F84A
        INCLUDE        <P16F84A.INC>
        __CONFIG      _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
#define EntradaMarcha    PORTA,0      ; Interruptor de puesta en marcha.
#define EntradaSentido    PORTA,4      ; Interruptor de sentido de giro.
;
PROGRAMA *****
ORG     0

Inicio
    bsf     STATUS,RP0
    bsf     EntradaMarcha          ; Configura las líneas de entrada.
    bsf     EntradaSentido
    clrf    PORTB                  ; Las líneas del Puerto B configuradas como salida.
    bcf     STATUS,RP0

Principal
    clrw                    ; Con esta combinación se detiene el motor.
    btfsc   EntradaMarcha     ; Comprueba el estado del interruptor.
    goto    ActivaSalida
    movlw   b'00010010'        ; Gira en un sentido.
    btfsc   EntradaSentido     ; Comprueba el sentido de giro deseado.
    movlw   b'00010001'        ; Gira en el otro sentido.

ActivaSalida
    movwf   PORTB
    goto    Principal

END

```

PROGRAMACIÓN EN LENGUAJE C

```

#include <motorcd.h>

#define PORTB = 0x06 //Se define los registros PORTB, TRISB y TRISA con las direcciones de

```

```

memoria
#BYTE TRISB = 0x86
#BYTE TRISA = 0x85

void main()
{
    int EntradaMarcha, EntradaSentido; // Se definen dos variables para arranque y para
                                        // el cambio de giro para del motor de CD
    TRISB=0x00;                        // las líneas del puerto B se configuran como salidas
    TRISA=0x03;                        // RA0 y RA1 se configuran como entradas
    while(TRUE)
    {
        EntradaMarcha=input(PIN_A0); // lee el PIN A0 y el dato lo guarda en la variable
        EntradaSentido=input(PIN_A1);

        if (EntradaMarcha==1)
        {
            Output_B(0x01);           // Escribe en puerto B arranque del motor de CD
        }
        else
        {
            Output_B(0x00);           // Escribe en puerto B paro del motor de CD
        }

        if(EntradaSentido==1)
        {
            Output_B(0x01);           // Escribe en puerto B cambio de sentido de giro del motor de CD
        }
        else
        {
            Output_B(0x02);
        }
    }
}

```

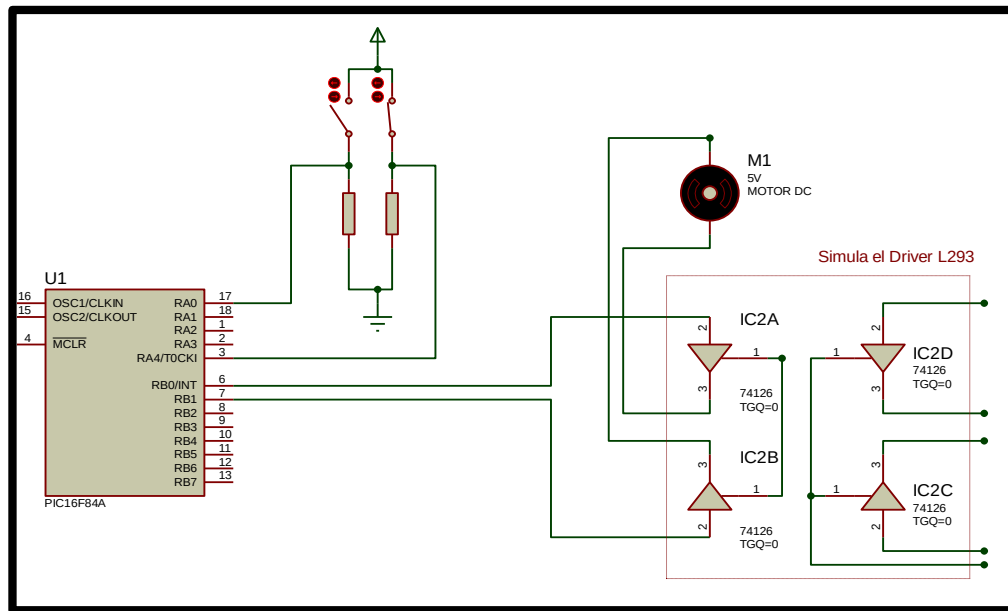


Figura 96. Simulación de la práctica Motor DC

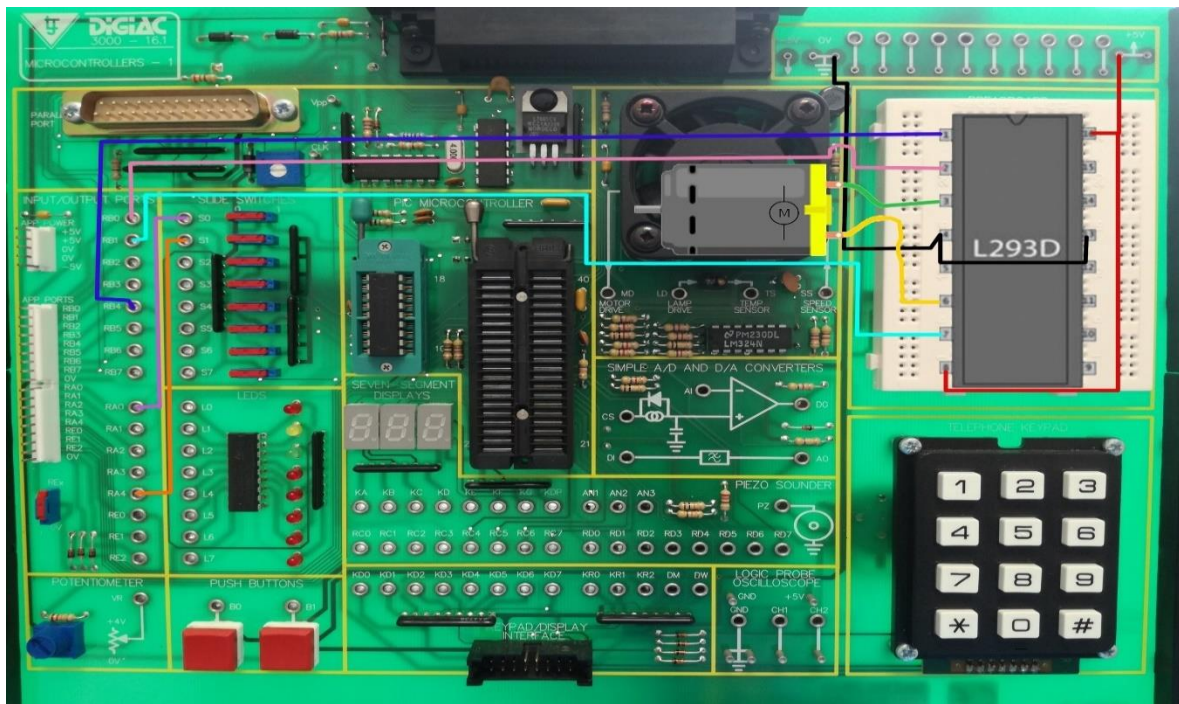


Figura 97. Esquema de la práctica motor DC

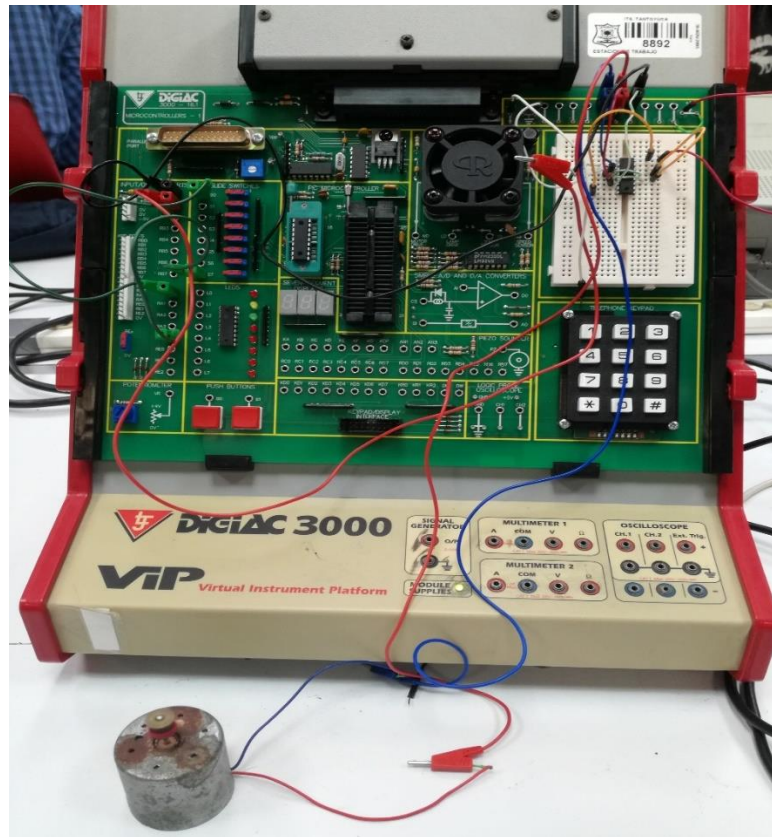


Figura 98. Implementación de la práctica Motor DC

5.5 PRÁCTICA 5 MOTOR PASO A PASO

Programa de control para un motor paso a paso (PAP) en funcionamiento y sentido de giro. Con RA0 a "0" el motor se pone en marcha y su sentido de giro dependerá del valor que tenga RA4 y su organigrama se muestra en la figura 100.

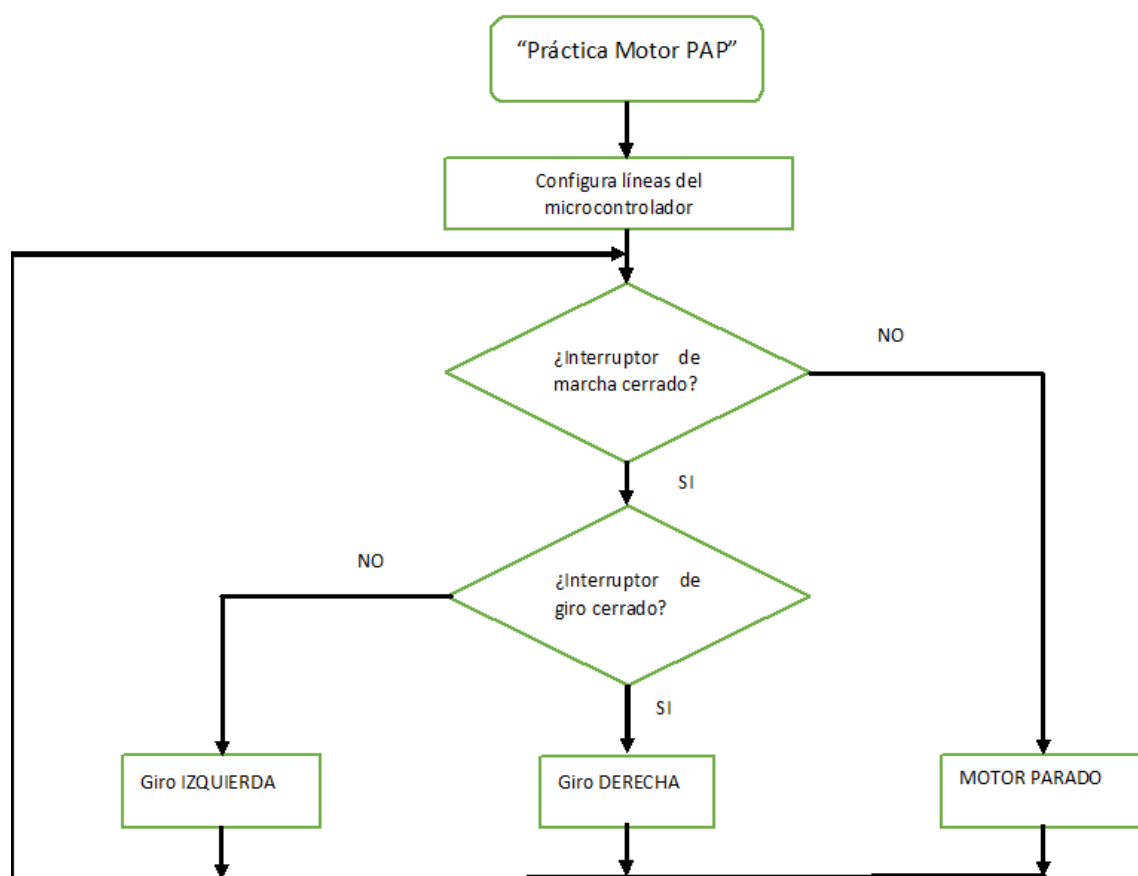


Figura 99. Organigrama Motor PAP

PROGRAMACIÓN LENGUAJE ENSAMBADOR

*****Práctica 5 Motor Paso a Paso *****

; CONFIGURACIÓN*****

```
LIST          P=16F84A
INCLUDE       <P16F84A.INC>
__CONFIG     _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
CBLOCK       0x0C
ENDC
```

```
#DEFINE EntradaMarcha    PORTA,0      ; Interruptor de puesta en marcha.
```

```
#DEFINE EntradaSentido   PORTA,4      ; Interruptor de sentido de giro.
```

;PROGRAMA *****

```
ORG 0
```

Inicio

```
bsf    STATUS,RP0
bsf    EntradaMarcha      ;Estas líneas se configuran como entrada.
bsf    EntradaSentido
clrf    PORTB              ;El Puerto B configuradas como salidas.
bcf    STATUS,RP0
```

Principal

```
btfss   EntradaMarcha      ; ¿Puesta en marcha?
goto    Gira                ; Sí.
clrf    PORTB              ; No, para el motor, poniendo a cero la línea
goto    Fin                 ; de habilitación.
```

Gira

```
btfsc   EntradaSentido     ; Comprueba el sentido de giro deseado.
goto    A_Izquierda
call    GiroDerecha        ; Gira en un sentido.
goto    Fin
```

A_Izquierda

```
call    Girolzquierda      ; Gira en sentido contrario.
```

```
Fin     goto    Principal
```

; Subrutina "Girolzquierda" -----

Girolzquierda

```
Movlw b'00110101'         ; Primer paso.
call    ActivaSalida       ; Envía a la salida donde está conectado el motor
```

PAP.

```
movlw   b'00110110'       ; Segundo paso.
```

```

        call    ActivaSalida
        movlw   b'00111010'           ; Tercer paso.
        call    ActivaSalida
        movlw   b'00111001'           ; Cuarto y último paso
        call    ActivaSalida
        return

; Subrutina "GiroDerecha" -----
GiroDerecha
        movlw   b'00111001'           ; Primer paso.
        call    ActivaSalida
        movlw   b'00111010'           ; Segundo paso.
        call    ActivaSalida
        movlw   b'00110110'           ; Tercer paso.
        call    ActivaSalida
        movlw   b'00110101'           ; Último paso.
        call    ActivaSalida
        return

; Subrutina "ActivaSalida" -----
ActivaSalida
        movwf   PORTB
        call    Retardo_10ms          ; Temporización antes del siguiente paso.
        return
INCLUDE <RETARDOS.INC>
END

```

PROGRAMACION EN LENGUAJE C

```

#include <motorpasos.h>
#define delay(clock = 4000000)
#define PORTB = 0x06 //Se define los registros PORTB, TRISB y TRISA con las direcciones de
memoria
#define TRISB = 0x86
#define TRISA = 0x85

void main()
{
    int EntradaMarcha, EntradaSentido; // Se definen dos variables para arranque y para

```

```

// el cambio de giro para del motor a pasos (unipolar)
TRISB=0x00;           //las líneas del puerto B se configuran como salidas
TRISA=0x03;           // RA0 y RA1 se configuran como entradas
while(TRUE)
{
    EntradaMarcha=input(PIN_A0); // lee el PIN A0 y el dato lo guarda en la variable
    EntradaSentido=input(PIN_A1);

    if (EntradaMarcha==1)
    {
        if(EntradaSentido==1)    // Escribe en puerto B la secuencia para cada paso del motor
        {
            Output_B(0x38);
            delay_ms(100);
            Output_B(0x34);
            delay_ms(100);
            Output_B(0x31);
            delay_ms(100);
            Output_B(0x32);
            delay_ms(100);
        }
        else
        {
            Output_B(0x32);
            delay_ms(100);
            Output_B(0x31);
            delay_ms(100);
            Output_B(0x34);
            delay_ms(100);
            Output_B(0x38);
            delay_ms(100);
        }
    }
    else
        Output_B(0x00);
}
}

```

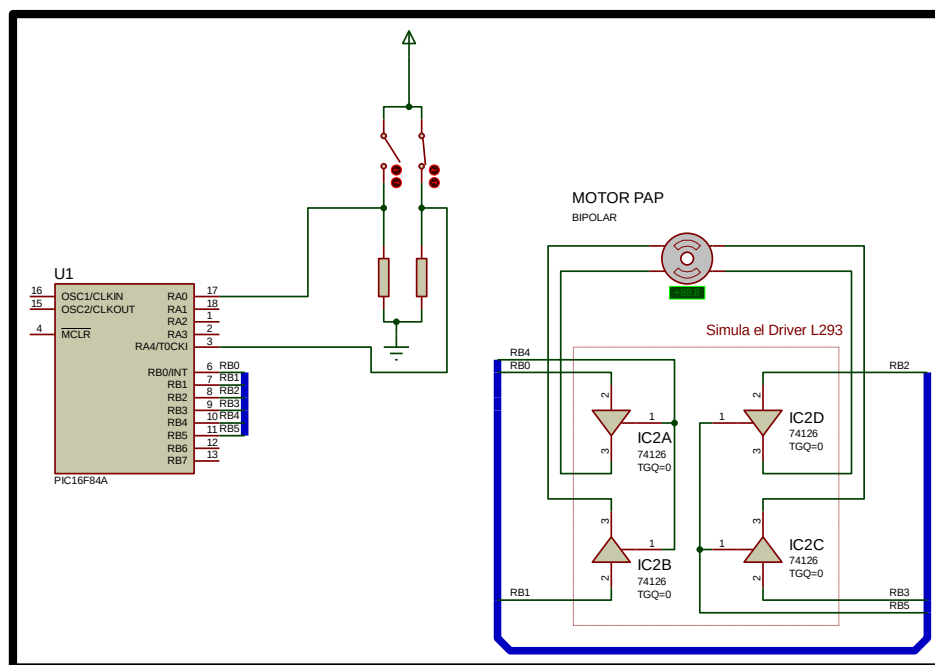


Figura 100. Simulación de la práctica Motor Paso a Paso

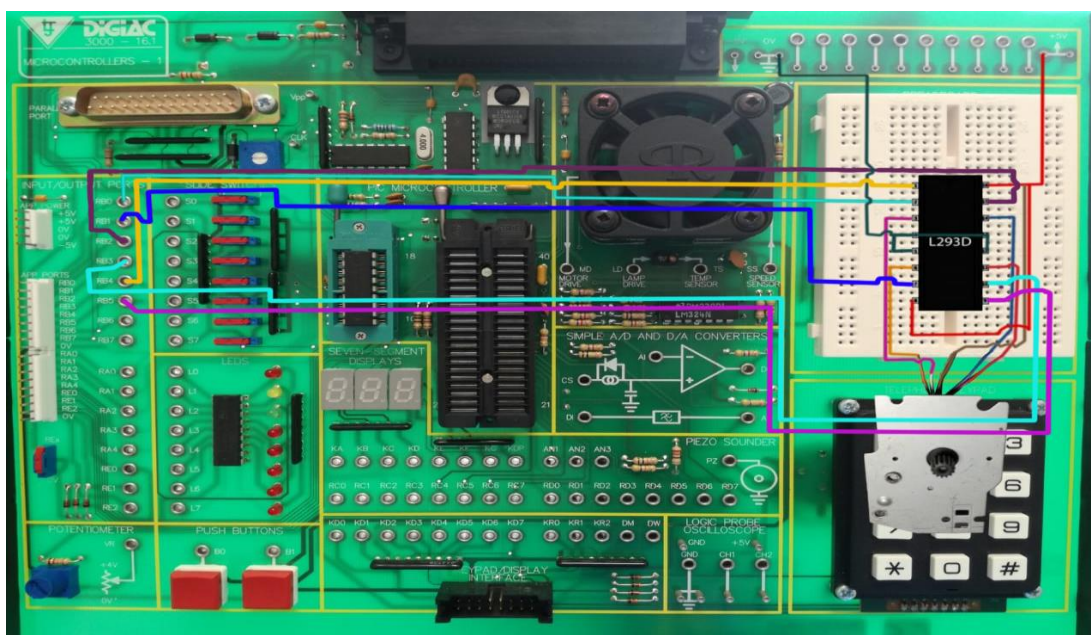


Figura 101. Esquema de la práctica motor Paso a Paso

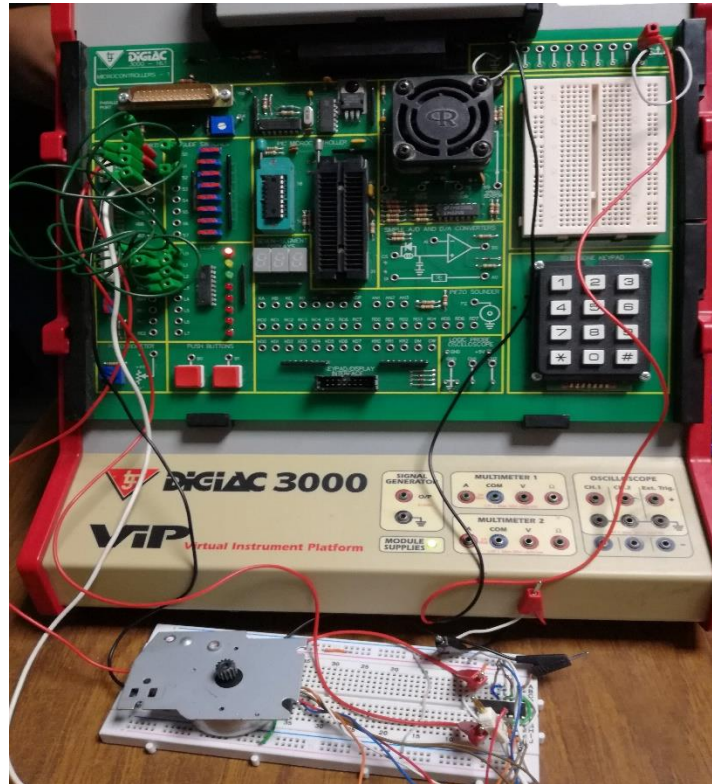


Figura 102. Implementación de la práctica Motor Paso a Paso

5.6 PRÁCTICA 6 SENSOR CNY70

En pantalla LCD se visualiza el Objeto "Detectado" o "No Detectado" que está detectando el sensor CNY70, según la configuración del esquema correspondiente.

Si:

Objeto Detectado transistor saturado entrada al inversor "0" --> RA3 = "1".

Objeto No Detectado transistor en corte entrada al inversor "1" --> RA3 = "0".

PROGRAMACIÓN LENGUAJE ENSAMBLADOR

```
*****Práctica 6 Sensor CNY70 *****
; CONFIGURACIÓN *****
    LIST          P=16F84A
    INCLUDE        <P16F84A.INC>
    __CONFIG      _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

    CBLOCK 0x0C
    ENDC

#DEFINE Sensor PORTA,3                ; Líneas donde se conecta el sensor.

; PROGRAMA *****

    ORG 0
Inicio
    call LCD_Inicializa
    movlw MensajeObjeto
    call LCD_Mensaje
    bsf     STATUS,RP0
    bsf     Sensor          ; Línea del sensor se configura como entrada.
    bcf     STATUS,RP0

Principal
    call LCD_Linea2
    movlw MensajeNoDetectado    ; En principio considera que es No Detectado.
    btfsc  Sensor              ; Lee el sensor.
    movlw MensajeDetectado      ; No, Detectado.
    call  LCD_Mensaje           ; Visualiza el resultado.
    goto  Principal
```

```

; "Mensajes" -----
;
Mensajes
    addwf PCL,F
MensajeObjeto
    DT " OBJETO", 0x00
MensajeDetectado
    DT "DETECTADO", 0x00
MensajeNoDetectado
    DT "NO DETECTADO", 0x00

    INCLUDE <RETARDOS.INC>
    INCLUDE <LCD_4BIT.INC>
    INCLUDE <LCD_MENS.INC>
    END

```

PROGRAMACIÓN EN LENGUAJE C

```

//Conexiones del módulo LCD
#define LCD_ENABLE_PIN PIN_A2
#define LCD_RS_PIN PIN_A0
#define LCD_RW_PIN PIN_A1
#define LCD_DATA4 PIN_B4
#define LCD_DATA5 PIN_B5
#define LCD_DATA6 PIN_B6
#define LCD_DATA7 PIN_B7
//Fin de las conexiones del modulo LCD

#include <lcd.c>

#define PORTA = 0x05
#define TRISB = 0x86
#define TRISA = 0x85

void main()
{
    int EntradaSensor;
    TRISB = 0x00; //las líneas del puerto B se configuran como salidas
    TRISA = 0x08; // Configura RA4 como entrada

```

```

lcd_init(); // Inicializa el modulo LCD
printf(lcd_putc,"\f"); // limpia el LCD
while(TRUE)
{
    EntradaSensor=bit_test(PORTA,3); // lee el PIN A3 del puerto A
    if(EntradaSensor==1)
    {
        lcd_gotoxy(1,1);          // Manda el texto "Color BLANCO" a la columna 1 fila 1
        printf(lcd_putc,"Color BLANCO");
        delay_ms(50);
    }
    else
    {
        lcd_gotoxy(1,1);          // Manda el texto "Color NEGRO" a la columna 1 fila 1
        printf(lcd_putc,"Color NEGRO ");
        delay_ms(50);
    }
}
}

```

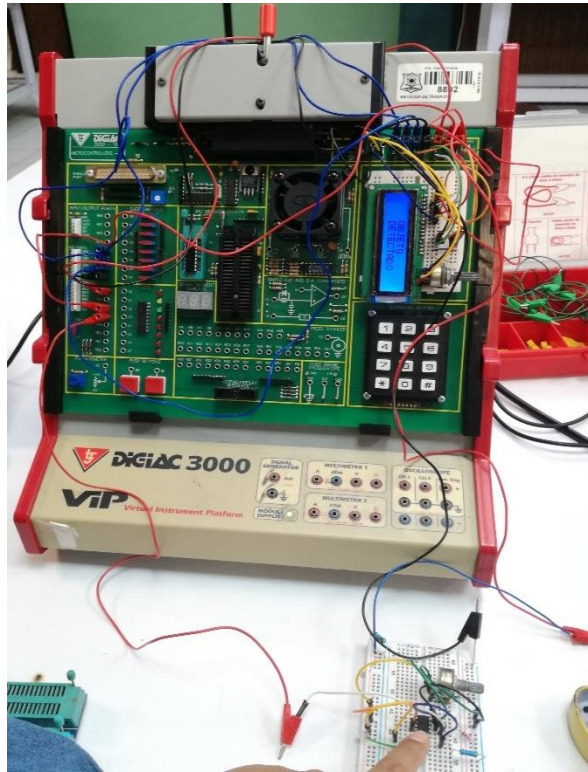



Figura 105. Implementación de la práctica CNY70

5.7 PRÁCTICA 7 SENSOR LDR

Una LDR se conecta a la entrada Trigger Schmitt RA4/T0CKI aplicando impulsos al Timer 0 cada vez que se oscurece al interponerse un objeto entre la fuente de luz y la LDR. En la pantalla del módulo LCD se visualiza el número de veces que se interrumpe el haz de luz en dos dígitos (hasta 99 máximos).

PROGRAMACIÓN LENGUAJE ENSAMBLADOR

```
*****Práctica 7 Sensor LDR*****
; CONFIGURACIÓN*****
LIST          P=16F84A
INCLUDE       <P16F84A.INC>
__CONFIG     _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
CBLOCK       0x0C
ENDC

; PROGRAMA *****
ORG 0

Inicio
    call  LCD_Inicializa          ; Se pone al principio de la línea 1.
    bsf   STATUS,RP0 ; Acceso al Banco 1.
    movlw b'00101000'            ; TMR0 como contador por flanco ascendente de
    movwf OPTION_REG ; RA4/T0CKI. Prescaler asignado al Watchdog.
    bcf   STATUS,RP0 ; Acceso al Banco 0.
    clrf  TMR0 ; Inicializa contador.

Principal
    call  LCD_Linea1              ; Se pone al principio de la línea 1.
    movf  TMR0,W ; Lee el Timer 0.
    call  BIN_a_BCD ; Se debe visualizar en BCD.
    call  LCD_Byte ; Visualiza, apagando las decenas en caso de que sean 0.
    goto  Principal
INCLUDE <RETARDOS.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
END
```

PROGRAMACIÓN EN LENGUAJE C

```
#include <sensorLDR.h>
#use delay(clock=4000000)
//Conexiones del modulo LCD
#define LCD_ENABLE_PIN PIN_A2
#define LCD_RS_PIN PIN_A0
#define LCD_RW_PIN PIN_A1
#define LCD_DATA4 PIN_B4
#define LCD_DATA5 PIN_B5
#define LCD_DATA6 PIN_B6
#define LCD_DATA7 PIN_B7
//Fin de las conexiones del módulo LCD
#include <lcd.c>

#BYTE PORTB = 0x06
#BYTE TRISB = 0x86
#BYTE TRISA = 0x85

int x;
#int_TIMER0

void TIMER0_isr(void)
{
    set_timer0(0);
}

void main()
{

    TRISB = 0x00; // Configura el puerto B como salidas
    TRISA = 0x1A; // Configura RA0, RA2 como entradas y RA1, RA3, RA4 como salidas
    lcd_init(); // Inicializa el modulo LCD

    setup_timer_0(RTCC_EXT_L_TO_H|RTCC_DIV_1); // Configura el Timer0
    set_timer0(0); // Carga el TIMER0
```

```

enable_interrupts(int_TIMER0);          // Habilita la interrupción del TIMER0
enable_interrupts(GLOBAL);              // Habilita las interrupciones globales
output_b(0x00);                          // Limpia el puerto B
printf(lcd_putc,"f"); // limpia el LCD
while(TRUE)
{
    x=get_timer0();      // Lee el valor actual del registro TIMER0

    lcd_gotoxy(1,1);
    printf(lcd_putc," %d",x); // Escribe el valor de x en el modulo LCD en la columna 1 fila 1
    delay_ms(50);          // Espera 50 ms para la siguiente escritura en el LCD
}
}

```

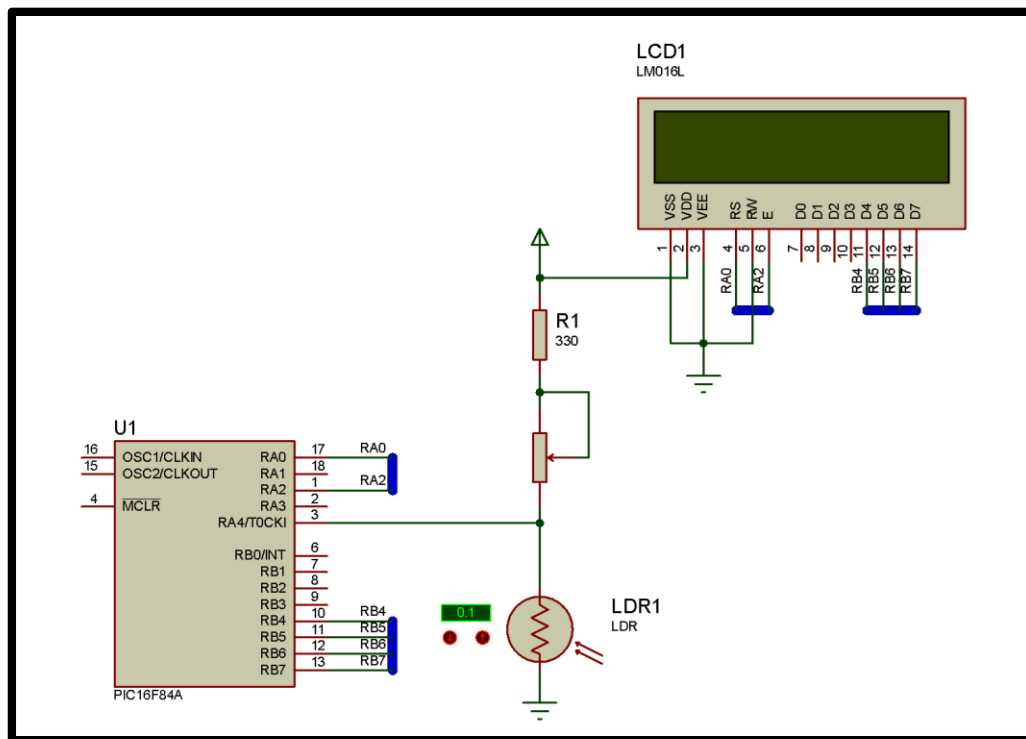


Figura 106. Simulación de la práctica sensor LDR

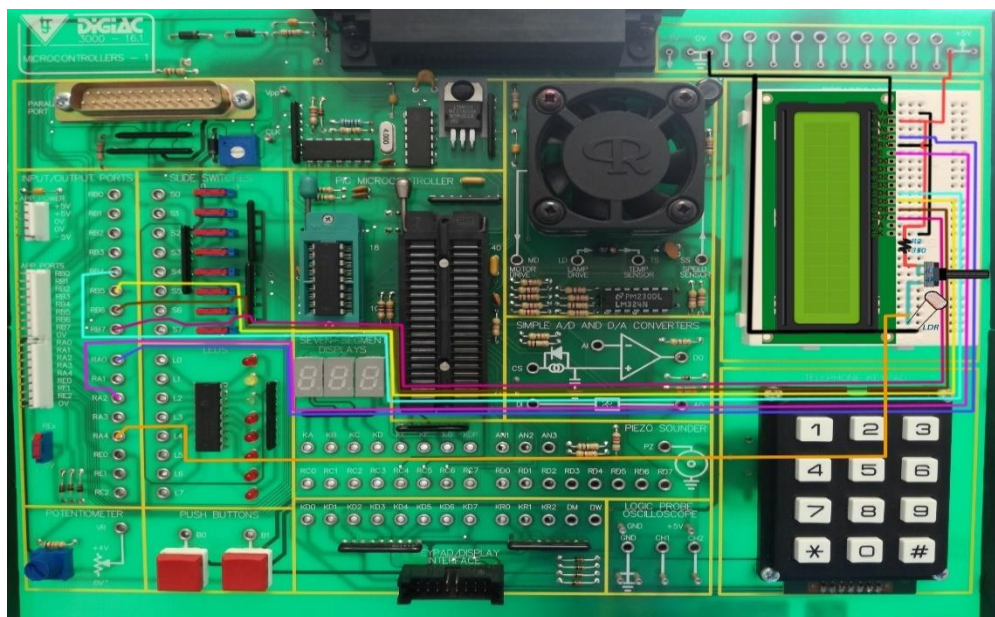


Figura 107. Esquema de la práctica sensor LDR

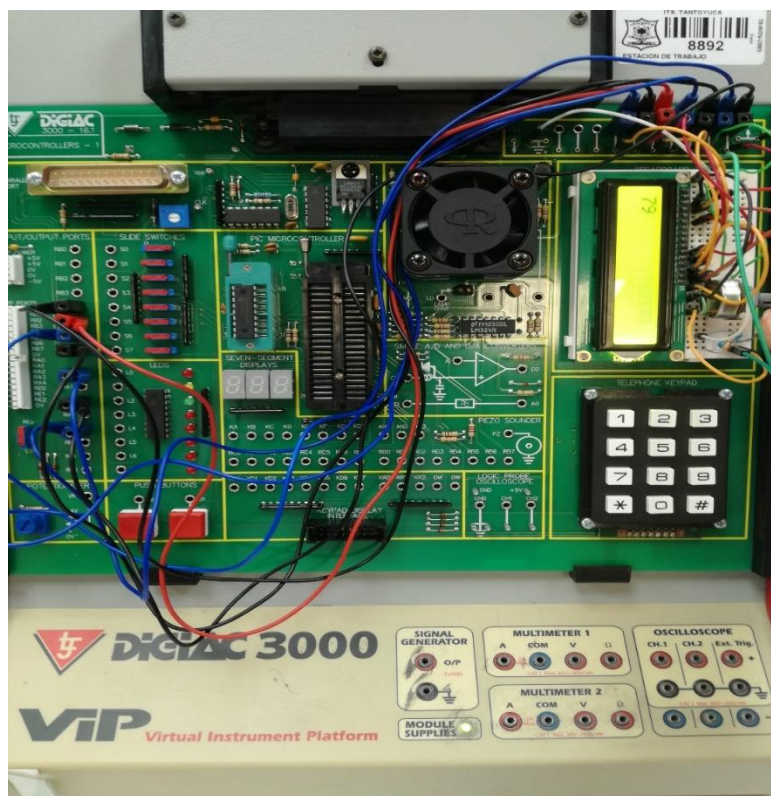


Figura 108. Implementación de la práctica sensor LDR

5.8 PRÁCTICA 8 SENSOR ULTRASÓNICO

Programa para un medidor de distancias hasta un objeto utilizando sensor por ultrasonido SRF04. Para el control del sensor en primer lugar se genera un pulso de 10 μ s a nivel alto por la línea RA3 que se conecta a la entrada de disparo del sensor. Seguidamente se espera a que en el sensor se ponga un nivel alto en la salida ECO que se conecta a la línea RA4 y se utilizan las interrupciones por desbordamiento del Timer 0 para medir el tiempo que está en alto el pulso. Seguidamente se visualiza en el módulo LCD el valor de la distancia hasta el objeto expresada en centímetros. Por cada centímetro de distancia al objeto el SRF04 aumenta 60 μ s la anchura del pulso.

En este programa la distancia mínima que se puede medir es 3 cm y la máxima 250 cm.

PROGRAMACIÓN LENGUAJE ENSAMBLADOR

```
***** Practica 8 Sensor Ultrasónico*****
;
; CONFIGURACIÓN*****
LIST          P=16F84A
INCLUDE       <P16F84A.INC>
__CONFIG     _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
CBLOCK 0x0C
Distancia
; Se expresará en centímetros.
ENDC
#DEFINE Disparo PORTA,3 ; Disparo para iniciar la medida.
#DEFINE Eco PORTA,4 ; Pulso cuya anchura hay que medir.
MIN_DISTANCIA EQU .3
MAX_DISTANCIA EQU .250
TMR0_Carga60micros EQU d'256'-d'27'; Valor obtenido experimentalmente con la
; ventana Stopwatch para
una interrupción del
; Timer 0 cada 60  $\mu$ s. Si no mide correctamente por las tolerancias de los componentes habrá
; que hacer un ajuste fino de este valor, comprobándolo sobre las condiciones reales.
;
; PROGRAMA *****
ORG 0
```

```

    goto    Inicio
    ORG     4
    goto    ServicioInterrupcion

Mensajes
    addwf   PCL,F
MensajeDistancia
    DT " Distancia: ", 0x00
MensajeCentimetro
    DT " cm", 0x00
MensajeDistanciaMenor
    DT "Dist. MENOR de:", 0x00
MensajeDistanciaMayor
    DT "Dist. MAYOR de:", 0x00

Inicio
    call    LCD_Inicializa
    bsf     STATUS,RP0
    bcf     Disparo
    bsf     Eco
    movlw   b'00000000'           ; Prescaler de 2 para el TMR0.
    movwf   OPTION_REG
    bcf     STATUS,RP0
    bcf     Disparo               ; Inicializa línea de disparo en bajo.

Principal
    clrf    Distancia             ; Inicializa el registro.
    bsf     Disparo               ; Comienza el pulso de disparo.
    call    Retardo_20micros      ; Duración del pulso.
    bcf     Disparo               ; Final del pulso de disparo.

Espera_Eco_1
    btfss   Eco                   ; Si ECO=0, espera el flanco de subida de la señal
    goto    Espera_Eco_1         ; de salida del sensor.
    movlw   TMR0_Carga60micros   ; Ya se ha producido el flanco de subida.
    movwf   TMR0                  ; Carga el Timer 0.
    movlw   b'10100000'          ; Autoriza interrupción del TMR0 (TOIE).
    movwf   INTCON

Espera_Eco_0
    btfsc   Eco                   ; Espera flanco de bajada de la señal de la
salida

```

```

        goto    Espera_Eco_0                ; del SRF04.
        clrf    INTCON                      ; Se ha producido el flanco de bajada. Prohíbe interrup.
        call    Visualiza                    ; Visualiza la distancia.
        call    Retardo_2s                  ; Espera un tiempo hasta la próxima medida.
Fin      goto    Principal

```

```

; Subrutina "ServicioInterrupcion" -----

```

```

;
; Se ejecuta debido a la petición de interrupción del Timer 0 cada 60 µs que es el incremento
; de la anchura de pulso por centímetro de distancia medido. La variable "Distancia" contiene el
; valor de la distancia expresada en centímetros.

```

```

;

```

```

ServicioInterrupcion

```

```

        movlw   TMR0_Carga60micros          ; Carga el Timer 0.
        movwf   TMR0
        movlw   .1                          ; Se utiliza instrucción "addwf", en lugar de "incf"
        addwf   Distancia,F                  ; para posicionar flag de Carry.
        movlw   MAX_DISTANCIA                ; En caso de desbordamiento carga su
máximo valor.
        btfsc   STATUS,C
        movwf   Distancia
        bcf     INTCON,T0IF
        retfie

```

```

; Subrutina "Visualiza" -----

```

```

;
; Visualiza la distancia expresada en centímetros. Se hace de manera que cuando haya que
; visualizar un número mayor de 99 las decenas siempre se visualicen aunque sean cero.
; Y cuando sea menor de 99 las decenas no se visualicen si es cero.

```

```

;

```

```

; Si la distancia es menor de 3 cm o mayor de 250 cm aparece un mensaje de error.

```

```

;

```

```

Visualiza

```

```

        call    LCD_Borra                    ; Borra la pantalla anterior.
        movlw   MIN_DISTANCIA                ; Va a comprobar si es menor del mínimo
admisible.
        subwf   Distancia,W                  ; (W) = (Distancia) - MIN_DISTANCIA
        btfss   STATUS,C                     ; ¿C=1?, ¿(W) positivo?, ¿(Distancia) >=

```

MIN_DISTANCIA?

 goto DistanciaMenor ; No ha resultado menor, y salta al mensaje de error.
 movf Distancia,W ; Va a comprobar si es mayor del máximo
admisible.

 sublw MAX_DISTANCIA ; (W)=MAX_DISTANCIA-(Distancia)
 btfsc STATUS,C ; ¿C=0?, ¿(W) negativo?, ¿MAX_DISTANCIA <
(Distancia)?

 goto DistanciaFiable ; No, la medida de la distancia entra dentro del rango.

;

DistanciaMayor

 movlw MAX_DISTANCIA ; La distancia es mayor que el máximo.

 movwf Distancia

 movlw MensajeDistanciaMayor

 goto VisualizaDistancia

;

DistanciaMenor

 movlw MIN_DISTANCIA ; La distancia es menor del mínimo fiable.

 movwf Distancia

 movlw MensajeDistanciaMenor

 goto VisualizaDistancia

DistanciaFiable

 movlw MensajeDistancia

VisualizaDistancia

 call LCD_Mensaje

 movlw .5 ; Centra la medida de la distancia en la segunda línea

 call LCD_PosicionLinea2 ; de la pantalla.

 movf Distancia,W

 call BIN_a_BCD ; Lo pasa a BCD.

 movf BCD_Centenas,W ; Primero las centenas.

 btfss STATUS,Z ; Si son cero no visualiza las centenas.

 goto VisualizaCentenas

 movf Distancia,W ; Vuelve a recuperar este valor.

 call BIN_a_BCD ; Lo pasa a BCD.

 call LCD_Byte ; Visualiza las decenas y unidades.

 goto Visualiza_cm

VisualizaCentenas

 call LCD_Nibble ; Visualiza las centenas.

 movf Distancia,W ; Vuelve a recuperar este valor.

```

        call    BIN_a_BCD                ; Lo pasa a BCD.
        call    LCD_ByteCompleto        ; Visualiza las decenas (aunque sea cero) y
Visualiza_cm                             ; unidades.
        movlw   MensajeCentimetro
        call    LCD_Mensaje
        return
INCLUDE <RETARDOS.INC>
INCLUDE <LCD_4BIT.INC>
INCLUDE <LCD_MENS.INC>
INCLUDE <BIN_BCD.INC>
END

```

PROGRAMACÓN EN LENGUAJE C

```

#include <ultrasonicov1.h>
#fuses HS, NOWDT, PUT, NOPROTECT
#use delay(clock = 4000000)
#use fast_io(A)

//Conexiones del modulo LCD
#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
//Fin de las conexiones del modulo LCD

#include <lcd.c>
unsigned int8 count;
unsigned int16 i, distancia;
#INT_TIMER0
void timer0_isr(){
    count++;
    clear_interrupt(INT_TIMER0);
}
int1 Sensor_Esperando(){

```

```

i = 0;
set_timer0(0);
count = 0;           // Reset Timer0
while(!input(PIN_A1) && (i < 1000))
    i = count * 256 + get_timer0();
if(i > 990)
    return 0;
else
    return 1;
}

unsigned int16 obtener_distancia(){
    i = 0;
    set_timer0(0);
    count = 0;
    while(input(PIN_A1) && (i < 25000))
        i = count * 256 + get_timer0();
    return i;
}

void main(){
    output_a(0);
    set_tris_a(2);           // Configura RA1 como entrada
    lcd_init();              // Inicializa el modulo LCD
    lcd_putc("\f");          // limpia el LCD
    clear_interrupt(INT_TIMER0);
    enable_interrupts(GLOBAL);
    enable_interrupts(INT_TIMER0);
    setup_timer_0 (T0_INTERNAL | T0_DIV_2);      // Configura el Timer0
    lcd_gotoxy(4, 1);        // Manda el texto "Distancia" a la columna 4 fila 1
    lcd_putc("Distancia.");
    while(TRUE){
        // Manda el pulso de 10us al HC-SR04 Trigger pin
        output_high(PIN_A0);
        delay_us(10);
        output_low(PIN_A0);
        // Lee el pulso que envia el HC-SR04 por Echo pin
        if(Sensor_Esperando()) {
            distancia = obtener_distancia();
            if(distancia > 24990) {

```

```

    lcd_gotoxy(3, 2);                // Manda el texto "Fuera de Rango" a la columna 3 fila 2
    lcd_putc("Fuera de Rango");
}
else {
    distancia = i/58;                // Calcula la distancia
    lcd_gotoxy(3, 2);
    lcd_putc("    cm ");
    lcd_gotoxy(6, 2);
    printf(lcd_putc,"%3Lu",distancia*2);    // Escribe la distancia calculada en la columna 3 fila 2
}
}
else {
    lcd_gotoxy(3, 2);
    lcd_putc(" Error ");
}
delay_ms(100);
}
}

```

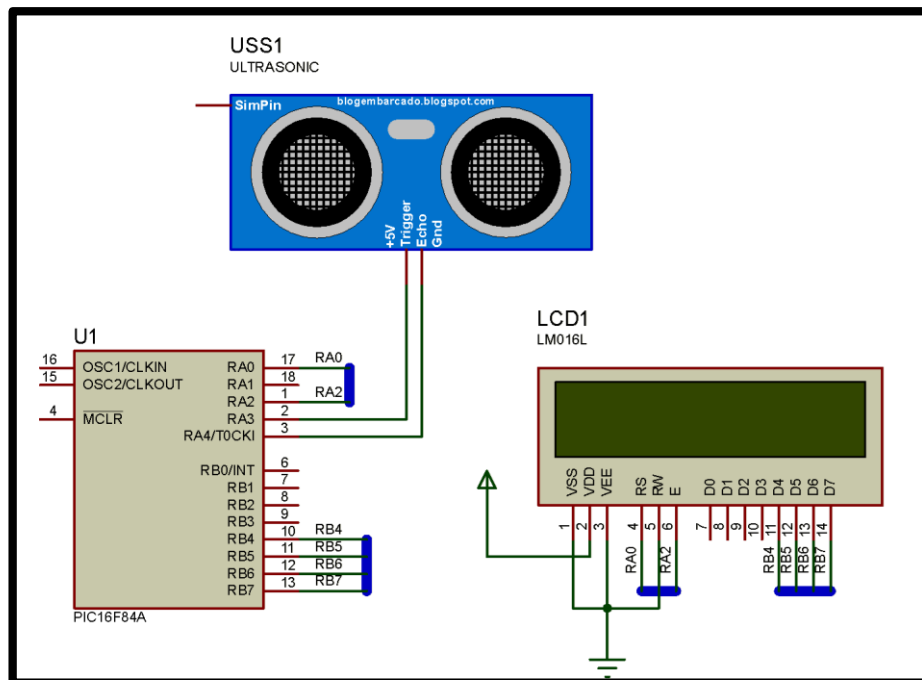


Figura 109. Simulación de la práctica sensor ultrasónico

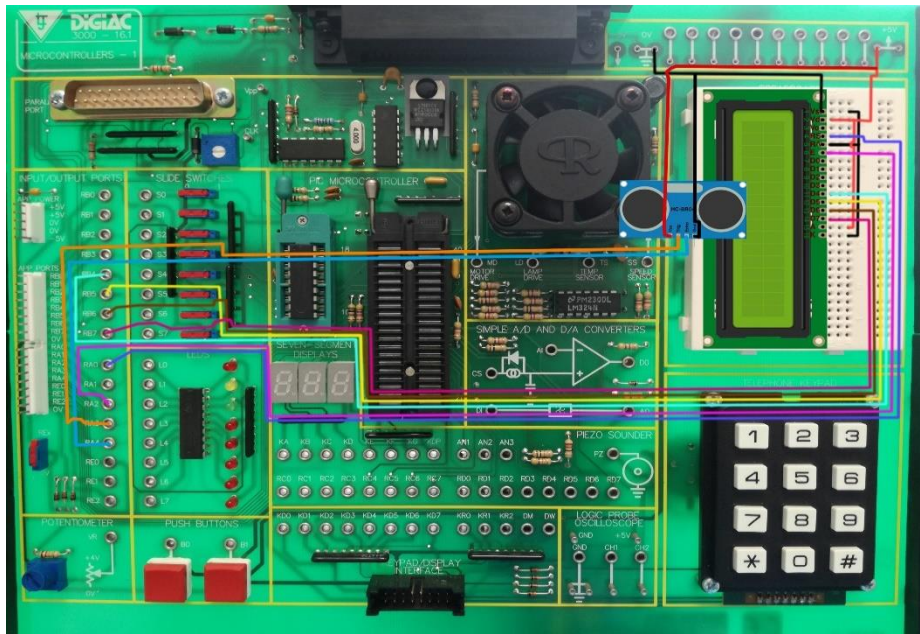


Figura 110. Esquema de la práctica sensor Ultrasónico

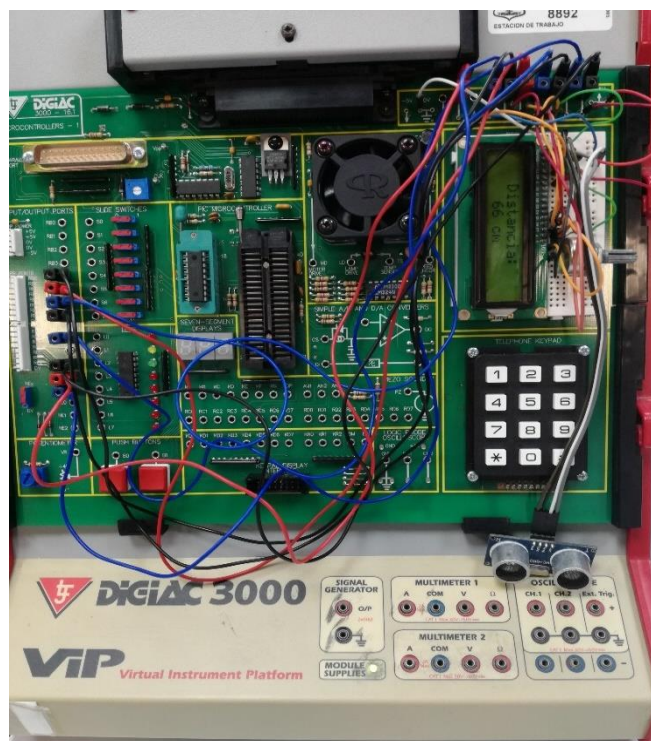


Figura 111. Implementación de la práctica sensor ultrasónico

5.9 PRÁCTICA 9 SERVO MOTOR

Programa de control de posicionamiento del servomotor sg90. Controla el ángulo mediante una señal cuadrada PWM de 20 ms de periodo que se aplica a su línea de control. El ángulo es gobernado por el tiempo en alto de la señal cuadrada desde 0° (para 0.3 ms de tiempo en alto), hasta 180° (para un tiempo en alto de 2.1 ms).

En este programa las líneas del Puerto A controlan el ángulo de posicionamiento con una resolución de 10° según los valores que se indican en la siguiente tabla, tomando como tiempo patrón 100 µs (0,1 ms) conseguidos mediante interrupciones por desbordamiento del Timer 0.

Tabla 27. Ángulo de posicionamiento del servo motor

Entrada RA4:RA0	Factor Alto (3+Entrada)	Tiempo Alto 0.1xFactorAlto	Tiempo Bajo 0.1 (200- Factor Alto)	Ángulo grados
0	3	0.3 ms	19.7 ms	0°
1	4	0.4 ms	19.6 ms	10°
2	5	0.5 ms	19.5 ms	20°
3	6	0.6 ms	19.4 ms	30°
17	20	2.0 ms	18.0 ms	170°
18	21	2.1 ms	17.9 ms	180°

PROGRAMACIÓN LENGUAJE ENSAMBLADOR

***** Práctica 9 Servo Motor *****

; CONFIGURACIÓN *****

LIST P=16F84A

INCLUDE <P16F84A.INC>

__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

CBLOCK 0x0C

FactorAlto ; Factor por el que se va a multiplicar el tiempo

ENDC ; patrón de 100 µs para obtener el tiempo en alto.

TMR0_CARGA EQU d'256'-d'90' ; Valor obtenido experimentalmente con la ventana

; Stopwatch para un tiempo de 100 µs.

; La próxima constante hay que variarla según el tipo de Servomotor utilizado.

```

ALTO_CERO_GRADOS EQU    d'300'          ; Tiempo en alto para 0°. Para el Futaba
S3003, 300 µs.
TIEMPO_PATRON EQU    d'100'          ; 100 µs conseguido mediante interrupciones.
FACTOR_MINIMO EQU    ALTO_CERO_GRADOS/TIEMPO_PATRON
#DEFINE Salida PORTB,0          ; Línea del Puerto B donde se conecta el
servomotor.
; PROGRAMA *****
                                ORG      0
                                gotoInicio
                                ORG      .4
                                gotoTimer0_Interrupcion

Inicio

                                bsf       STATUS,RP0
                                bcf       Salida          ; Esta línea se configura como salida.
                                movlw    b'00011111'    ; Puerto A configurado como entrada.
                                movwf    PORTA
                                movlw    b'00001000'    ; TMR0 sin prescaler.
                                movwf    OPTION_REG
                                bcf       STATUS,RP0
                                movlw    TMR0_CARGA
                                movwf    TMR0            ; Carga el Timer 0.
                                movlw    b'10100000'
                                movwf    INTCON          ; Autoriza interrupción T0I y la general (GIE).

Principal

                                movf     PORTA,W         ; Lee el puerto de entrada
                                andlw    b'00011111'    ; Se queda con los bits válidos.
                                addlw    FACTOR_MINIMO    ; Para conseguir el tiempo mínimo
correspondiente a 0°.

                                movwf    FactorAlto      ; Valor entregado a la subrutina de
                                gotoPrincipal          ; atención a la interrupción.

; Subrutina "Timer0_Interrupcion" -----
;
; Mantiene la salida en alto un tiempo igual a 100µs x (FactorAlto) y en bajo un tiempo igual
; a 100µs x (200-FactorAlto). El periodo de la señal cuadrada lo mantiene en 20 ms.
;

                                CBLOCK
                                Guarda_W

```

```

Guarda_STATUS
Timer0_ContadorA    ; Contador auxiliar.
ENDC

Timer0_Interrupcion

STATUS en el
    movwf    Guarda_W    ; Guarda los valores de tenían W y
STATUS en el
    swapf    STATUS,W    ; programa principal.
    movwf    Guarda_STATUS
    bcf      STATUS,RP0   ; Garantiza que trabaja en el Banco 0.
    movlw    TMR0_CARGA
    movwf    TMR0
    decfsz   Timer0_ContadorA,F    ; Decrementa el contador.
    goto     Fin_Timer0_Interrupcion
    btfsc    Salida        ; Testea el anterior estado de la salida.
    goto     EstabaAlto

EstabaBajo
    bsf      Salida        ; Estaba bajo y lo pasa a alto.
    movf     FactorAlto,W    ; Repone el contador nuevamente con el
tiempo en
    movwf    Timer0_ContadorA    ; alto.
    goto     Fin_Timer0_Interrupcion

EstabaAlto
    bcf      Salida        ; Estaba alto y lo pasa a bajo.
    movf     FactorAlto,W    ; Repone el contador nuevamente con
el tiempo
    sublw    .200          ; en bajo.
    movwf    Timer0_ContadorA    ; El periodo será de
100µs·200=20000µs=20ms.
Fin_Timer0_Interrupcion
    swapf    Guarda_STATUS,W ; Restaura registros W y STATUS.
    movwf    STATUS
    swapf    Guarda_W,F
    swapf    Guarda_W,W
    bcf      INTCON,RBIF
    bcf      INTCON,T0IF
    retfie
END

```

PROGRAMACIÓN EN LENGUAJE C

```
#include <ServoMotor.h>
#use delay(clock=4000000)
#BYTE TRISB = 0x86
#BYTE TRISA = 0x85
#define FactorMinimo 4
int FactorAlto;      //Variables globales
int TiempoBajo = 250;
int i=0;
#int_TIMER0
void TIMER0_isr(void)
{
    set_timer0(225);      // Carga el TIMER0
    FactorAlto=FactorMinimo + input_A(); // Lee el puerto B para determinar los grados
                                   // para mover el ServoMotor

    i++;
    if(i<=FactorAlto){
        output_b(0x01);
    }
    else{
        output_b(0x00);
    }
    if(i==TiempoBajo){
        i=0;}
}
void main()
{
    TRISB = 0x00; // Configura el puerto B como salidas
    TRISA = 0xFF; // Configura el puerto A como entradas
    setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1); // Configura el Timer0
    set_timer0(225);      // Carga el TIMER0 para un desbordamiento a los 100 us
    enable_interrupts(int_TIMER0);      // Halilita la interrupción del TIMER0
    enable_interrupts(GLOBAL);          // Habilita las interrupciones globales
    output_b(0x00);      // Limpia el puerto B
    while(TRUE) // Se mantiene oscioso
    {
    }
```

}

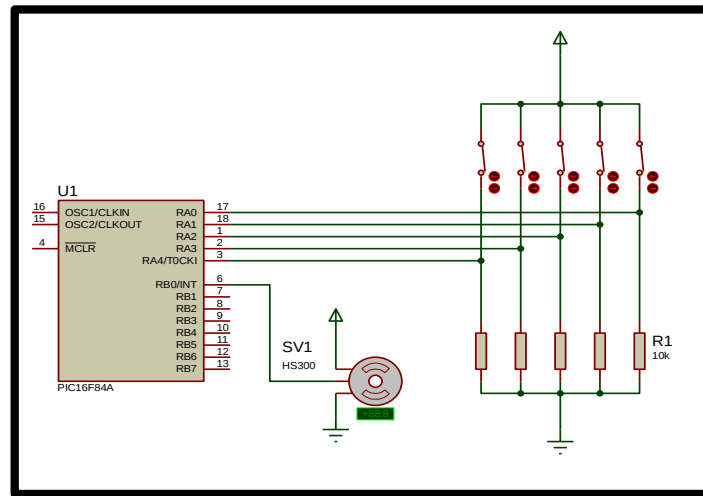


Figura 112. Simulación de la práctica servo motor

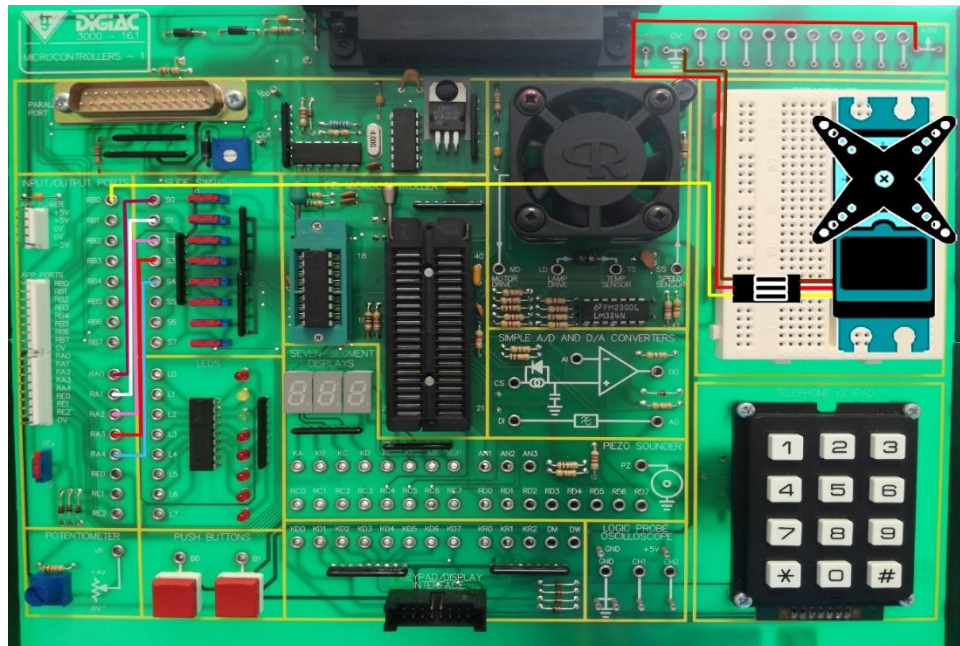


Figura 113. Esquema de la práctica servo motor

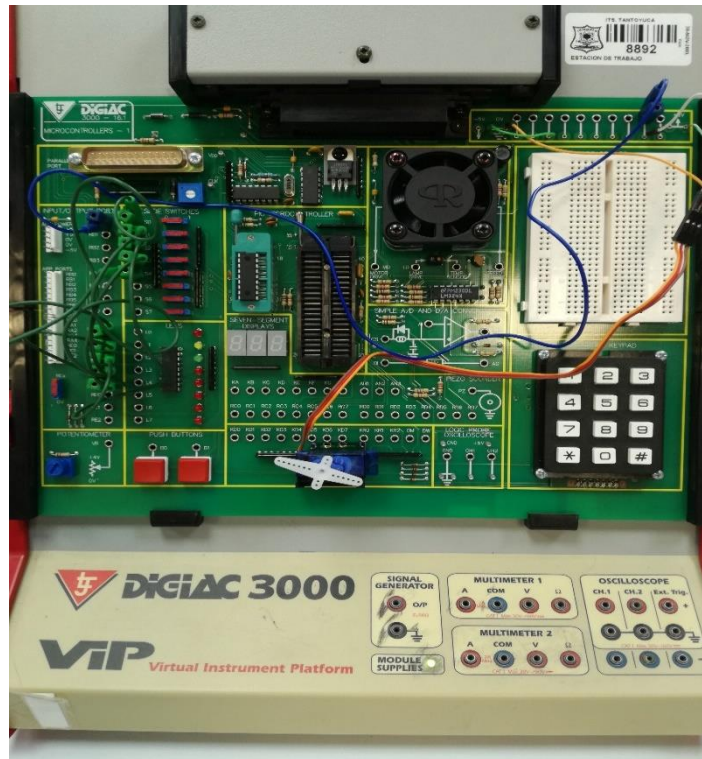


Figura 114. Implementación de la práctica servo motor

5.10 PRÁCTICA 10 TECLADO PHONE

Un teclado matricial está constituido por una matriz de pulsadores dispuestos en filas y columnas. Cada tecla se conecta a una fila y a una columna, las 12 teclas necesitan solo 7 líneas para conectarse. Cuando una tecla es pulsada queda en contacto una fila con una columna.

- Las filas del teclado se conectan en el puerto B (RB0 a RB3) configuradas como salida.
- Las columnas del teclado se conectan del puerto B (RB4 a RB7) configuradas como entrada.

El módulo LCD visualiza el orden de la tecla pulsada expresado en decimal. Utiliza técnica Polling o lectura constante del puerto al que está conectado el teclado.

El orden del teclado matricial del módulo DIGIAC 3000 es la siguiente:

RB0-----KD4

RB1-----KD6

RB2-----KD5

RB3-----KD7

RB4-----KD0

RB5-----KD1

RB6-----KD2

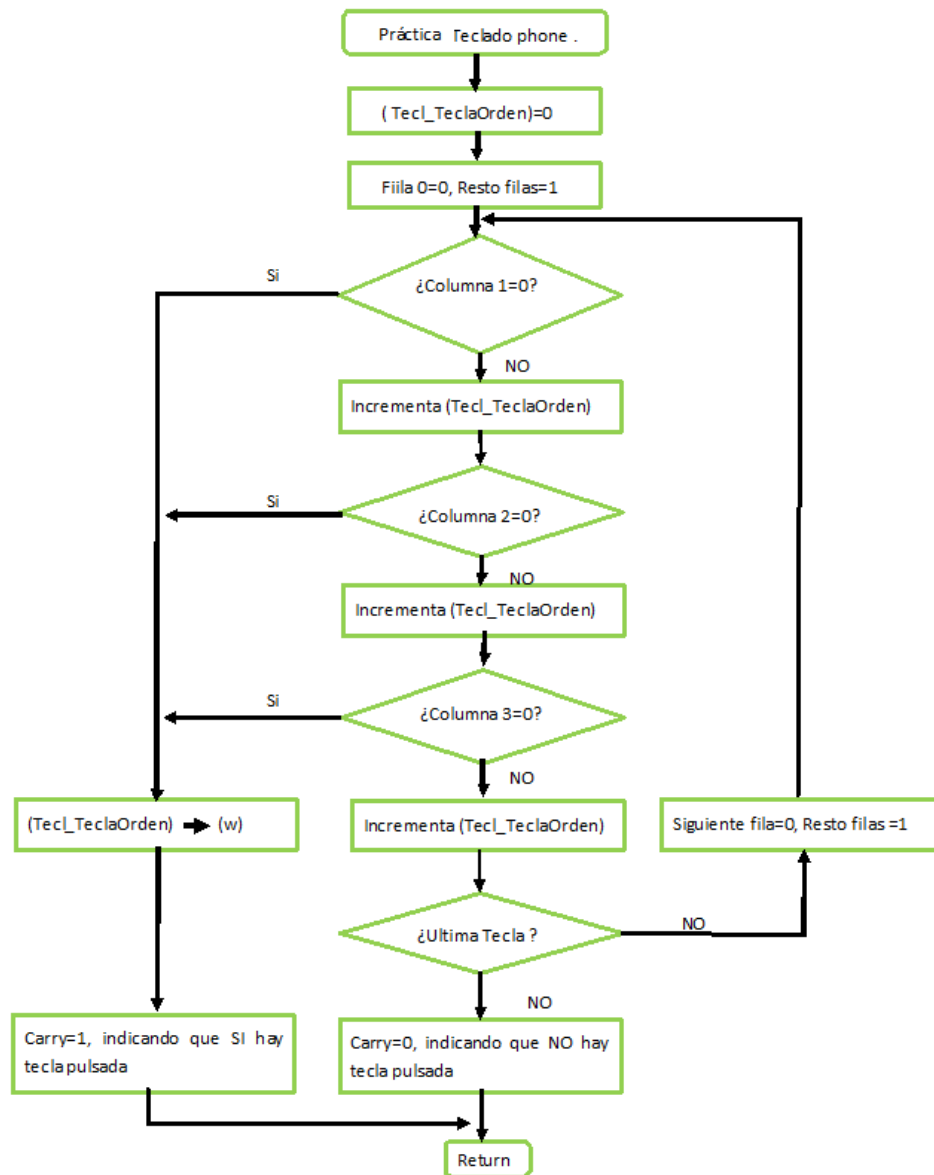


Figura 115. Organigrama de la práctica teclado phone

PROGRAMACIÓN LENGUAJE ENSAMBLADOR

```
*****práctica 11 Teclado phone *****
;
; CONFIGURACIÓN *****
    LIST          P=16F84A
    INCLUDE        <P16F84A.INC>
    __CONFIG      _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
    CBLOCK 0x0C
    ENDC
; PROGRAMA *****
    ORG 0
Inicio
    call LCD_Inicializa
    call Teclado_Inicializa          ; Configura las líneas del teclado.
Principal
    call Teclado_LeeOrdenTecla      ; Lee el teclado.
    btfss STATUS,C                  ; ¿Pulsa alguna tecla?, ¿C=1?
    goto Fin                        ; No, por tanto sale.
    call BIN_a_BCD                   ; Lo pasa a BCD.
    call LCD_ByteCompleto           ; Visualiza en pantalla.
    call Teclado_EsperaDejePulsar   ; No sale hasta que levante el dedo.
Fin
    goto Principal

    INCLUDE <TECLADO.INC>            ; Subrutinas de control del teclado.
    INCLUDE <BIN_BCD.INC>
    INCLUDE <LCD_4BIT.INC>
    INCLUDE <RETARDOS.INC>
    END
```

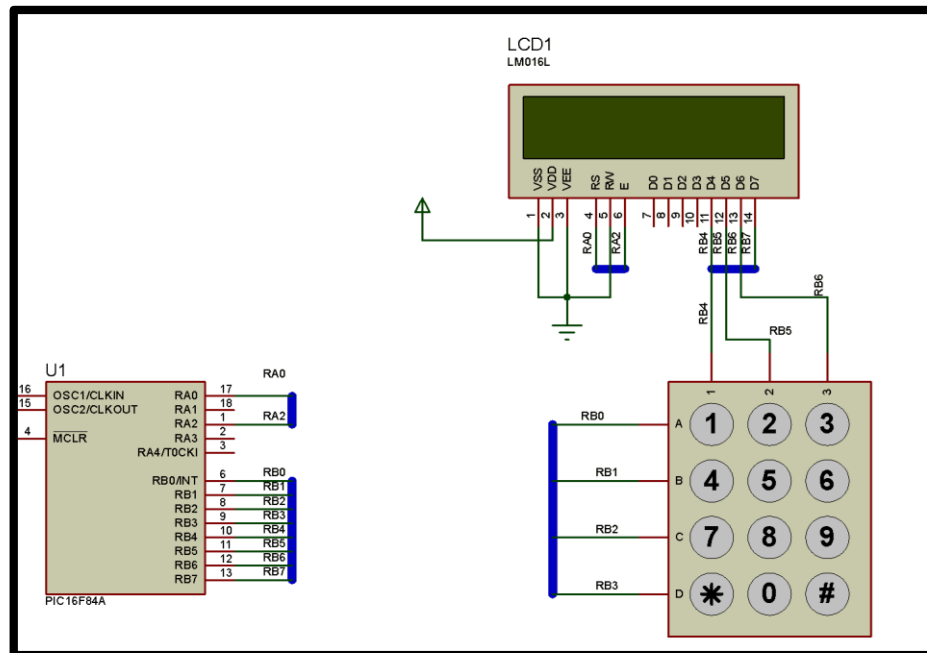


Figura 116. Simulación de la práctica teclado phone

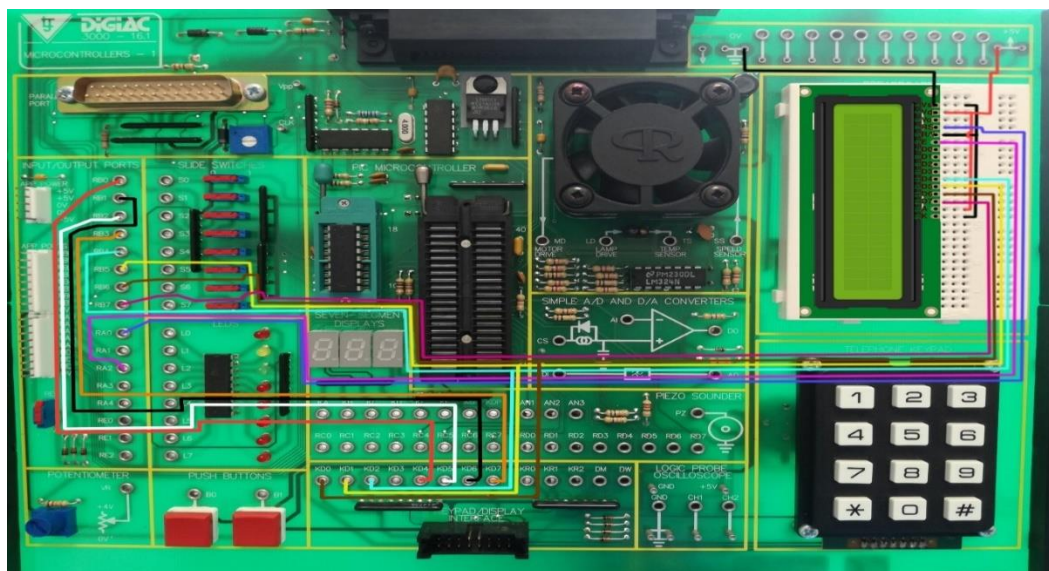


Figura 117. Esquema de la práctica teclado phone

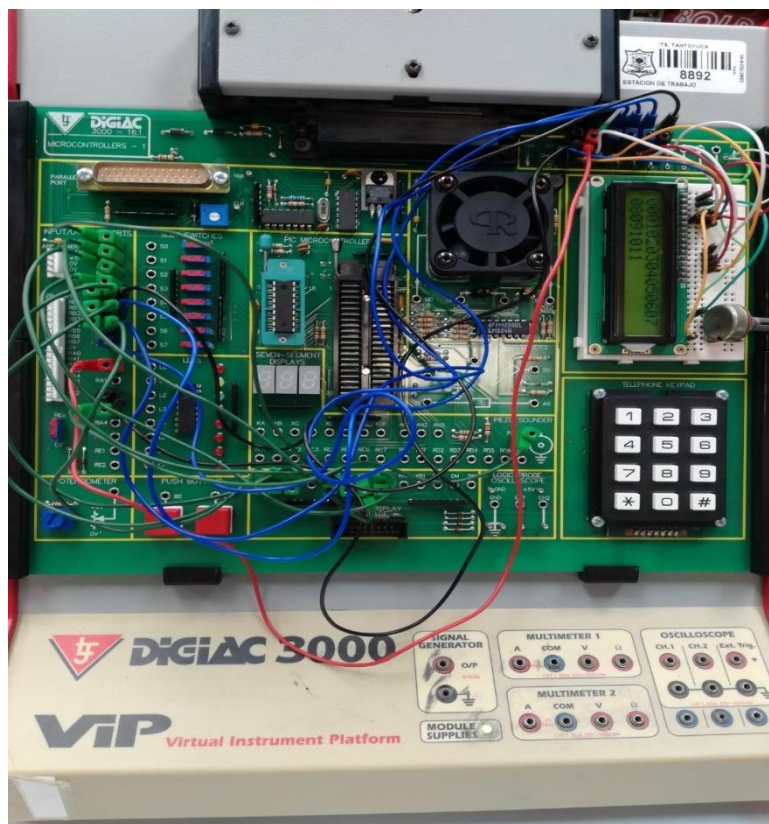


Figura 118. Implementación de la práctica teclado phone

5.11 PRÁCTICA 11 TMR0 COMO CONTADOR

Este programa comprueba el funcionamiento del timer 0 como contador de los impulsos aplicados a la línea RA4/ T0CKI, donde se ha conectado un pulsador, cada vez que se presiona el pulsador se incrementa un contador visualizando en el display LCD. Como es un incremento por cada impulso aplicado al pin T0CKI, no es necesario asignarle divisor de frecuencia al TMR0, por tanto, el Prescaler se asigna al Watchdog.

PROGRAMACIÓN LENGUAJE ENSAMBLADOR

```
;***** Práctica 11 Timer 0 como contador*****
; CONFIGURACIÓN*****
LIST          P=16F84A
INCLUDE        <P16F84A.INC>
__CONFIG      _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
CBLOCK        0x0C
ENDC

; PROGRAMA *****
ORG 0

Inicio
call LCD_Inicializa
bsf STATUS,RP0 ; Acceso al Banco 1.
movlw b'00111000' ; TMR0 como contador por flanco descendente de
movwf OPTION_REG ; RA4/T0CKI. Prescaler asignado al Watchdog.
bcf STATUS,RP0 ; Acceso al Banco 0.
clrf TMR0 ; Inicializa el contador.

; La sección "Principal" es de mantenimiento. Sólo se dedica a visualizar el Timer 0.
Principal
call LCD_Lineal ; Se pone al principio de la línea 1.
movf TMR0,W ; Lee el Timer 0.
call BIN_a_BCD ; Se debe visualizar en BCD.
call LCD_Byte ; Visualiza apagando las decenas en caso de que sean 0.
goto Principal
INCLUDE <RETARDOS.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
END
```

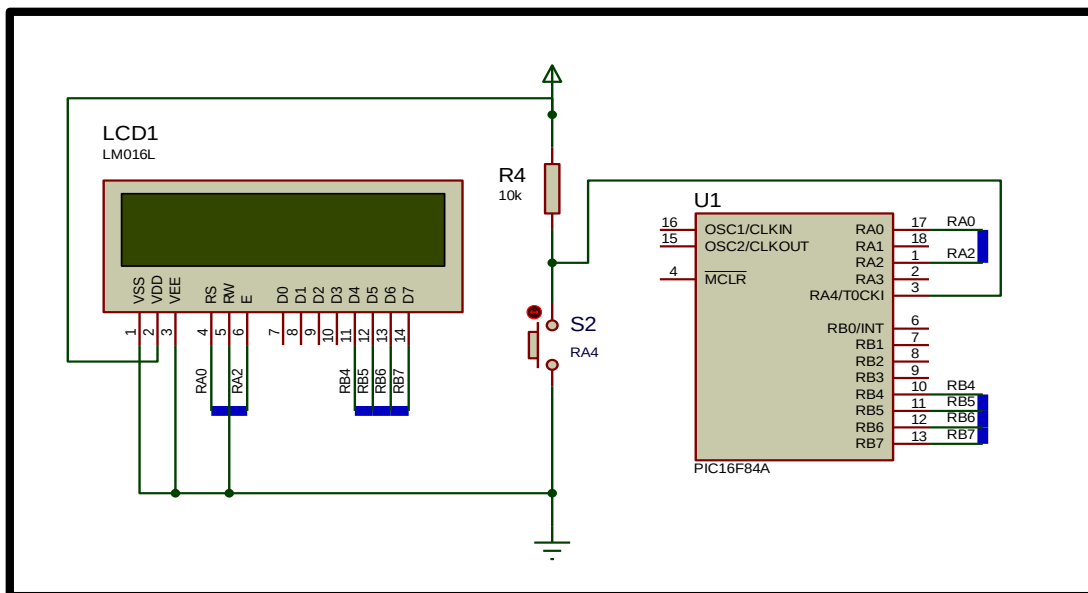


Figura 119. Simulación de la práctica TIMER0 como contador

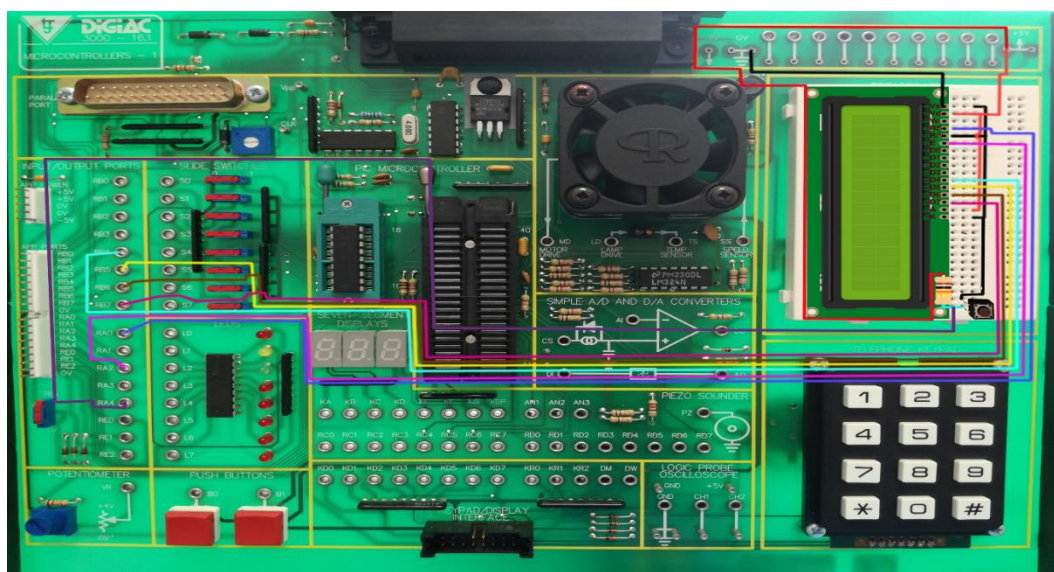


Figura 120. Esquema de la práctica TIMER0 como contador

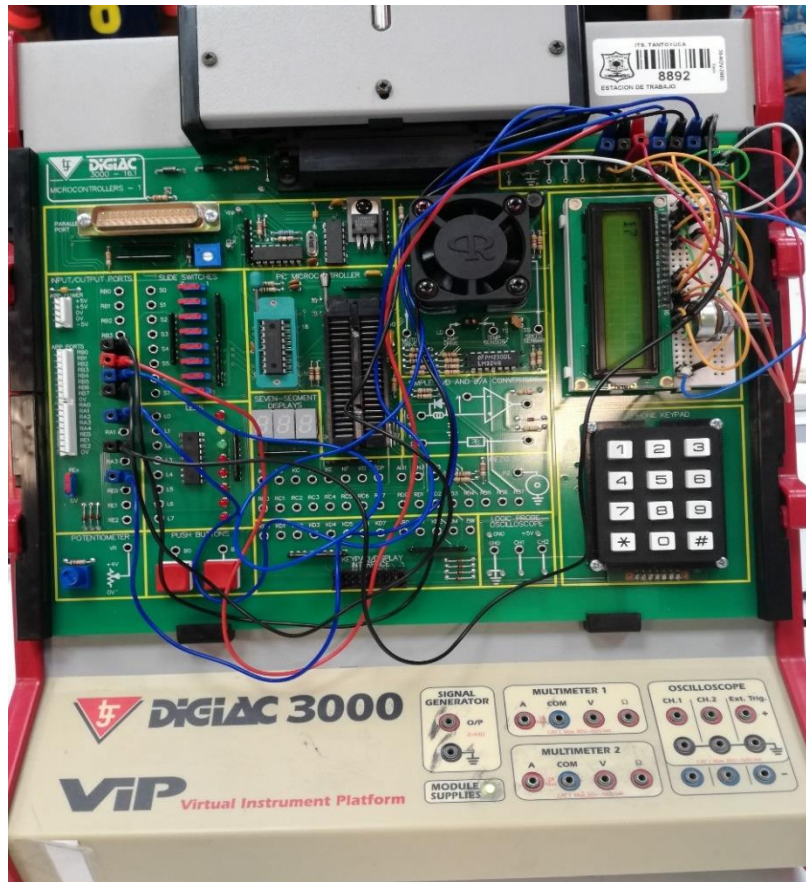


Figura 121. Implementación de la práctica TMRO como contador

5.12 PRÁCTICA 12 AUTOMATIZACIÓN DE UNA CISTERNA

Controla el nivel de un depósito de líquido. Utiliza (entre paréntesis las líneas del microcontrolador a la que se han conectado):

- Tres sondas detectoras: SV, Sonda de Vacío (RA0); SLL, Sonda de Llenado (RA1); SR, Sonda de Rebose (RA2).
- Dos bombas de agua: B1 (RB5), B2 (RB6).
- Cinco indicadores: Vacío (RB0), Llenándose (RB1), Lleno (RB2), Rebose (RB3) y Alarma (RB4).

FUNCIONAMIENTO

Cuando ninguna de las sondas está mojada se entiende que el depósito está vacío y se accionarán las dos bombas. El indicador "Vacío" se iluminará. Cuando el nivel del líquido toque la sonda de vacío "SV" seguirá llenándose el depósito con las dos bombas. El indicador "Llenándose" se ilumina. Cuando el nivel del líquido toca la sonda de llenado "SLL", para la bomba B2, quedando B1 activada en modo mantenimiento. El indicador "Lleno" se ilumina. Si el nivel del líquido moja la sonda de rebose "SR" se apaga también la bomba B1, quedando las dos bombas fuera de servicio. El indicador "Rebose" se enciende. Cuando se produce un fallo o mal funcionamiento en las sondas de entrada (por ejemplo que se active la sonda de rebose y no active la de vacío) se paran las dos bombas. El indicador "Alarma" se ilumina.

Según el enunciado del problema, teniendo en cuenta las conexiones citadas y poniendo la salida no utilizada (RB7) siempre a cero, la tabla de verdad como se muestra en la siguiente tabla 28.

Tabla 28. Tabla de verdad del funcionamiento de la automatización de una cisterna

RA2	RA0	RB7									
0	0	0	0	1	1	0	0	0	0	1	Configuración 0. Estado "Vacío".
0	0	1	0	1	1	0	0	0	1	0	Configuración 1. Estado "Llenándose".
0	1	0	0	0	0	1	0	0	0	0	Configuración 2. Estado "Alarma".
0	1	1	0	0	1	0	0	1	0	0	Configuración 3. Estado "Lleno".
1	0	0	0	0	0	1	0	0	0	0	Configuración 4. Estado "Alarma".
1	0	1	0	0	0	1	0	0	0	0	Configuración 5. Estado "Alarma".
1	1	0	0	0	0	1	0	0	0	0	Configuración 6. Estado "Alarma".
1	1	1	0	0	0	0	1	0	0	0	Configuración 0. Estado "Rebose".

PROGRAMACIÓN LENGUAJE ENSAMBLADOR

*****Práctica 12 Automatización de una cisterna*****

; CONFIGURACIÓN *****

LIST P=16F84A

INCLUDE <P16F84A.INC>

__CONFIG __CP_OFF & __WDT_OFF & __PWRTE_ON & __XT_OSC

; PROGRAMA *****

ORG 0 ; El programa comienza en la dirección 0.

Inicio

bsf STATUS,RP0 ; Acceso al Banco 1.

clrf TRISB ; Las líneas del Puerto B se configuran como salida.

movlw b'00011111' ; Las 5 líneas del Puerto A se configuran como entrada.

movwf TRISA

bcf STATUS,RP0 ; Acceso al Banco 0.

Principal

movf PORTA,W ; Lee el valor de las variables de entrada.

andlw b'00000111' ; Se queda con los tres bits de entrada.

addwf PCL,F ; Salta a la configuración adecuada.

Tabla

goto Configuracion0

goto Configuracion1

goto Configuracion2

goto Configuracion3

goto Configuracion4

goto Configuracion5

```

        goto    Configuracion6
        goto    Configuracion7
Configuracion0
        movlw   b'00001010'           ; (Configuración 0).
        goto    ActivaSalida
Configuracion1
        movlw   b'00001001'           ; (Configuración 1).
        goto    ActivaSalida
Configuracion2
        movlw   b'00100011'           ; (Configuración 2).
        goto    ActivaSalida
Configuracion3
        movlw   b'00001111'           ; (Configuración 3).
        goto    ActivaSalida
Configuracion4
        movlw   b'00100000'           ; (Configuración 4).
        goto    ActivaSalida
Configuracion5
        movlw   b'00000111'           ; (Configuración 5).
        goto    ActivaSalida
Configuracion6
        movlw   b'00010111'           ; (Configuración 6).
        goto    ActivaSalida
Configuracion7
        movlw   b'00111111'           ; (Configuración 7).
ActivaSalida
        movwf   PORTB                 ; Visualiza por el puerto de salida.
        goto    Principal
END

```

PROGRAMACIÓN EN LENGUAJE C

```

#include <ControlNivel.h>
#BYTE TRISB = 0x86
#BYTE TRISA = 0x85
#BYTE PORTB = 0x06
//ENTRADAS
#define SR PIN_A2

```

```

#define SLL PIN_A1
#define SV PIN_A0

//SALIDAS
#define Vacio PIN_B0
#define Llenandose PIN_B1
#define Lleno PIN_B2
#define Rebose PIN_B3
#define Alarma PIN_B4
#define Bomba1 PIN_B5
#define Bomba2 PIN_B6

void main()
{

    TRISB=0x00;
    TRISA=0x07;
    output_b(0x00);
    while(TRUE)
    {
        if(input(SR)==0 & input(SLL)==0 & input(SV)==0)
        {
            output_high(Vacio);
            output_low(Llenandose);
            output_low(Lleno);
            output_low(Rebose);
            output_low(Bomba1);
            output_low(Bomba2);
            output_low(Alarma);
        }
        else if(input(SR)==0 & input(SLL)==0 & input(SV)==1)
        {
            output_low(Vacio);
            output_high(Llenandose);
            output_low(Lleno);
            output_low(Rebose);
            output_low(Bomba1);
            output_low(Bomba2);
        }
    }
}

```

```

    output_low(Alarma);
}
else if(input(SR)==0 & input(SLL)==1 & input(SV)==1)
{
    output_low(Vacio);
    output_low(Llenandose);
    output_high(Lleno);
    output_low(Rebose);
    output_low(Bomba1);
    output_high(Bomba2);
    output_low(Alarma);
}
else if(input(SR)==1 & input(SLL)==1 & input(SV)==1)
{
    output_low(Vacio);
    output_low(Llenandose);
    output_low(Lleno);
    output_high(Rebose);
    output_high(Bomba1);
    output_high(Bomba2);
    output_low(Alarma);
}
else
{
    output_low(Vacio);
    output_low(Llenandose);
    output_low(Lleno);
    output_low(Rebose);
    output_high(Alarma);
    output_high(Bomba1);
    output_high(Bomba2);
}
}
}

```

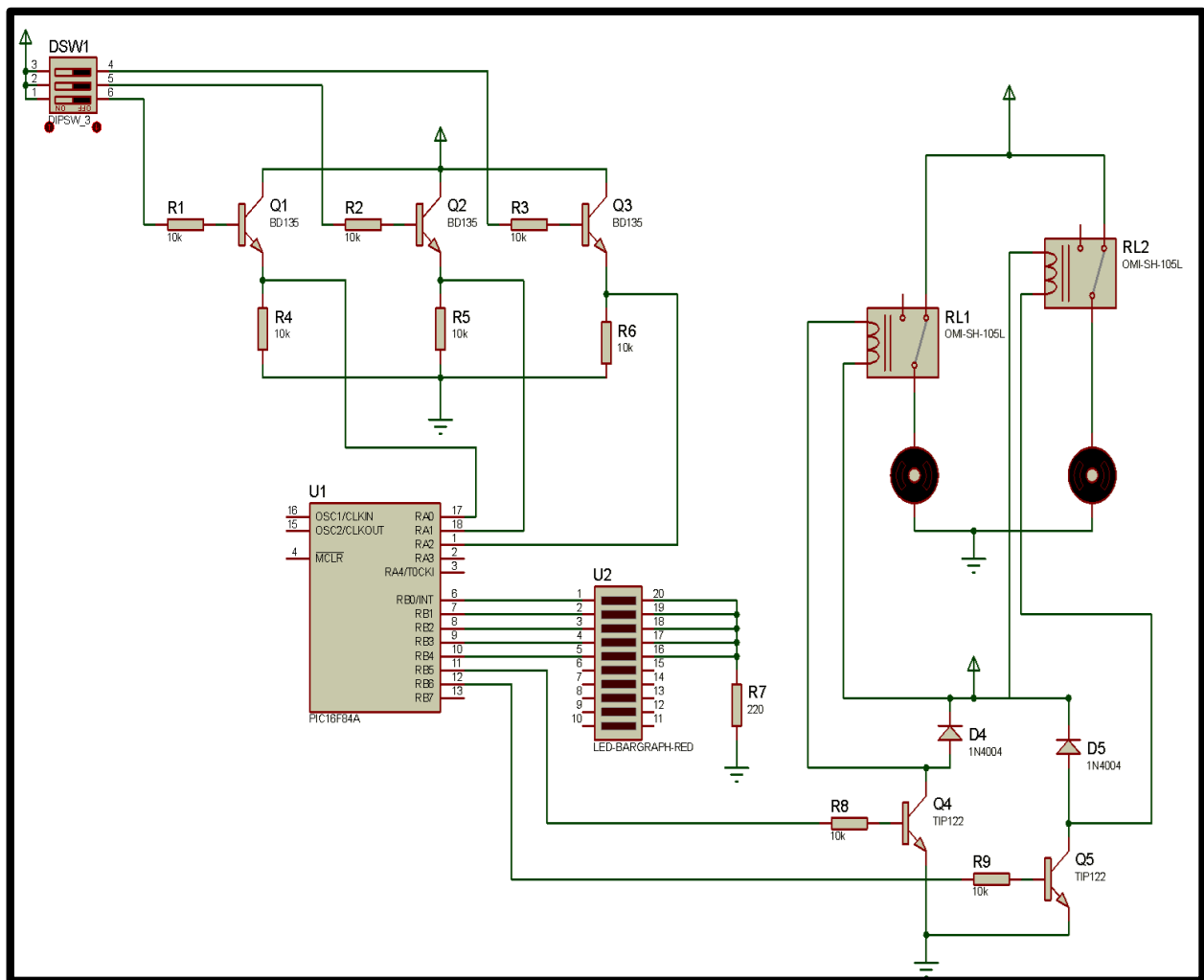


Figura 122. Simulación de la práctica automatización de una cisterna

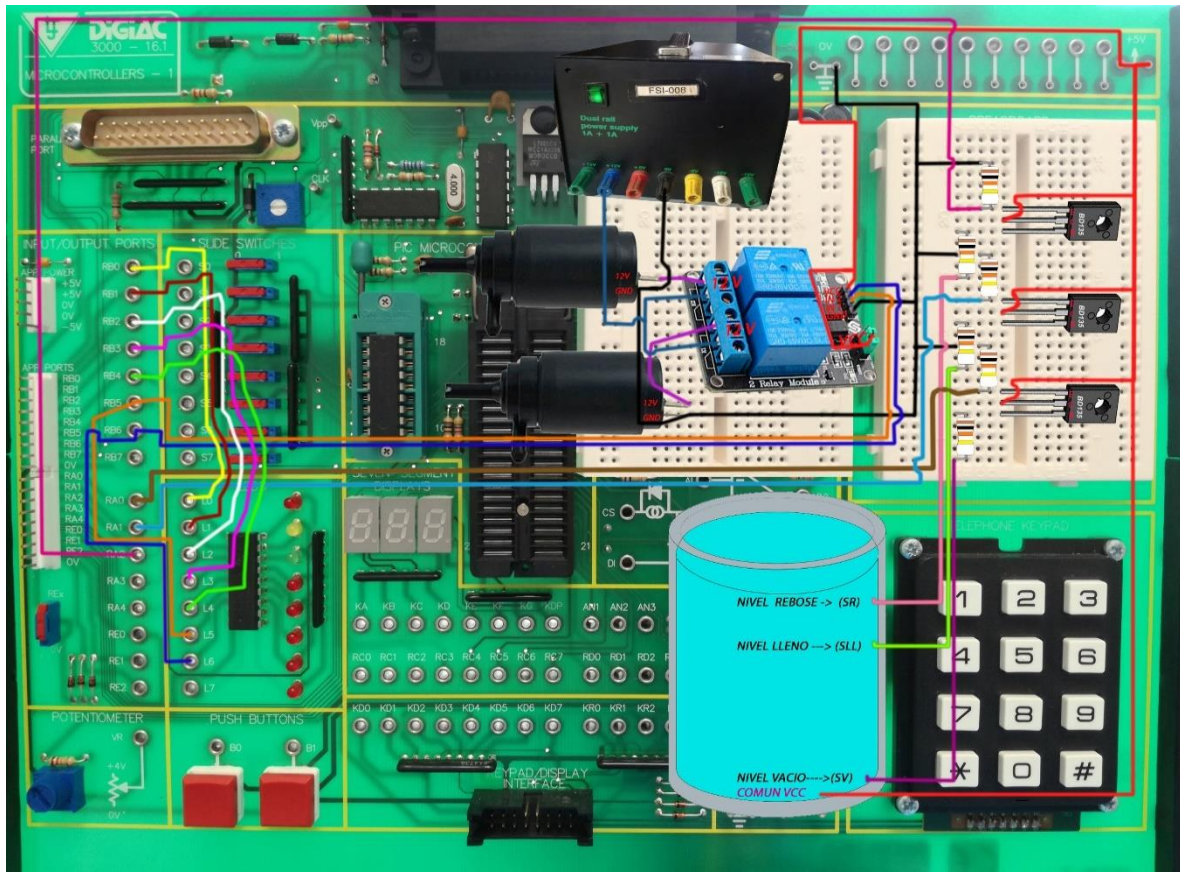


Figura 123. Esquema de la práctica automatización de una cisterna

CAPÍTULO 6.

CONCLUSIONES Y RECOMENDACIONES

6.1 CONCLUSIONES

Durante el desarrollo del proyecto se presentó una situación poco peculiar, ya que, en contraste con otros temas, acerca de los microcontroladores PIC16F84A, existe una gran cantidad de fuentes bibliográficas, por lo que fue necesario analizar minuciosamente cada fuente de información y posteriormente seleccionar solo el mejor material.

En lo que se refiere a la explicación de las características, arquitectura y operación de dicho microcontrolador, fue necesario hacer una transcripción de los términos que se encuentran en las diversas fuentes bibliográficas, ya que la mayoría de las obras relacionadas con este tema fueron desarrolladas en otros países con modismos y terminología diferente.

Los recursos del PIC16F84A son muy variados, por lo que requirieron de un fuerte análisis para poder ser explicados.

El manejo de los programas MPLAB, PROTEUS no se especifica correctamente en ninguna fuente de información, por lo que el desarrollo de estas secciones fue basado en la experimentación, la práctica y los resultados al operar dichas herramientas de desarrollo.

La parte más importante del proyecto son las prácticas propuestas, la cuales se desarrollaron en circuitos reales y se garantiza su correcto funcionamiento. Durante el desarrollo de este trabajo se adquirió una gran habilidad para implementar soluciones a problemas comunes y específicos mediante los microprocesadores PIC.

6.2 RECOMENDACIONES

Es muy recomendable que el usuario de los microcontroladores conozca muy bien las especificaciones de operación del dispositivo, las cuales están disponibles en el sitio web del fabricante; esto con la finalidad de darle un buen uso, de no sobrecargarlo o de operarlo en condiciones inadecuadas.

El PIC16F84A de Microchip está construido mediante tecnología CMOS, por lo que se recomienda manipularlo de forma cuidadosa y siempre con una pulsera antiestática u otra herramienta que ayude a la conservación de este dispositivo.

Cuando se pretende cargar un programa en un dispositivo microcontrolador, es necesario un circuito grabador, el cual se debe elegir de acuerdo a las necesidades del usuario. Los programadores de uso libre son una excelente opción para todos aquellos usuarios que no requieren de un gran sistema para realizar prácticas y prototipos didácticos.

Por lo general, todos los circuitos grabadores operan bajo ciertos niveles de señal procedentes de algún puerto de la computadora, por lo que de preferencia se debe trabajar con una computadora de escritorio, debido a que comúnmente se tiene el problema de que las computadoras portátiles no ofrecen el mismo voltaje a la salida de sus puertos.

El simulador del programa ISIS PROTEUS es una herramienta indispensable que debe ser usada para comprobar el correcto funcionamiento del programa diseñado, todo esto, con la finalidad de que no se grabe el PIC con un programa que no cumpla con las especificaciones requeridas.

El tema de los microcontroladores es muy extenso, y aun cuando en este proyecto se ha pretendido analizar y explicar todo lo relacionado con el dispositivo PIC16F84A, es necesario que el usuario investigue y se actualice constantemente sobre la evolución de esta línea de productos PIC Micro de la empresa Microchip.

REFERENCIAS

- Angulo, J. Ma. y Angulo, I., (1997). *Microcontroladores PIC, Diseño práctico de aplicaciones*, Editorial McGraw Hill.
- Angulo, J. Ma., Cuenca, E. M. y Angulo, J., (1997). *Microcontroladores PIC, la solución en un chip*, Editorial Paraninfo.
- Angulo, J. Ma., Cuenca, E. M. y Angulo, J., (1998). *Aplicaciones de los microcontroladores PIC de Microchip*, Editorial McGraw Hill.
- Bolton, William.(2001). *Ingeniería de control*, Editorial Alfaomega
- Microchip Technology Inc, (1997). *Microchip PIC16F84 Microcontroller Data Book*.
- Digikey 2013. *LM358 Digikey* Disponible en: <https://www.digikey.com.mx/es/product-highlight/t/texas-instruments/lm358-dual-operational-amplifiers>. Consultada en 06/01/2019
- Duque, Edison C., (1998). *Curso Avanzado de Microcontroladores PIC*, CEKIT.
- EcuRed 2015. *Transistor BD135 EcuRed* Disponible en: [https://www.ecured.cu/Transistor BD135](https://www.ecured.cu/Transistor_BD135). Consultada en 06/01/2019
- Joyanes, Luis A., (1996). *Problemas de metodología de la programación*, Editorial McGraw Hill.
- *Master prog* disponible en: <https://jonathanmelgoza.com/blog/master-prog-manual-y-programa/>. Consultada en 06/01/2019
- *MPLAB IDE* disponible en: <https://pic-microcontroller.com/mplab-ide-v8-92-free-download/> Consultada en 06/01/2019
- Palacios, E., Remiro, F. y López, L. (2009). *Microcontrolador PIC16F84. Tercera Edición*. Ed. Alfaomega.
- profetolocka 2015. *Módulo de relevadores de profetolocka* Sitio web: <http://www.profetolocka.com.ar/2015/05/09/modulo-de-4-reles-para-arduino/>. Consultada en 06/01/2019
- Tecnosefarad 2014. *Sensor CNY70 de Tecnosefarad* Disponible en: <http://www.tecnosefarad.com/2014/03/sensor-de-infrarrojos-cny70-como-entrada-digital/>. Consultada en 06/01/2019

- Ti (1986- 2016). *L293D Ti* Disponible en: <http://www.ti.com/lit/ds/symlink/l293.pdf>. Consultada en 06/01/2019
- *Virtual box* Disponibles en: <https://www.virtualbox.org/>. Consultada en 06/01/2019

ANEXOS

MÓDULO DIGIAC 3000

Este módulo se ha diseñado específicamente para operar con el programa de electrónica modular DIGIAC; incluye una placa de circuitos, un dispositivo PIC reprogramable, software de desarrollo de programas en lenguaje ensamblador. También incluye un cable de conexión para el puerto paralelo de la PC.

Áreas incluidas:

- Microprocesadores, Microcomputadoras y Microcontroladores
- Generalidades de los Microcontroladores PIC
- Sistemas Numéricos *f* Software de PIC
- Interfaces
- Desarrollo de Programas
- Entrada/Salida Básica
- Sistemas Lógicos
- Subrutinas y Ciclos
- Interrupciones
- Temporizadores
- Escanear el Teclado
- Generación de Sonido
- Programación de EEPROM
- Conversión “Analógica a Digital”
- Conversión “Digital a Analógica”
- Control Simple de Ciclo Cerrado

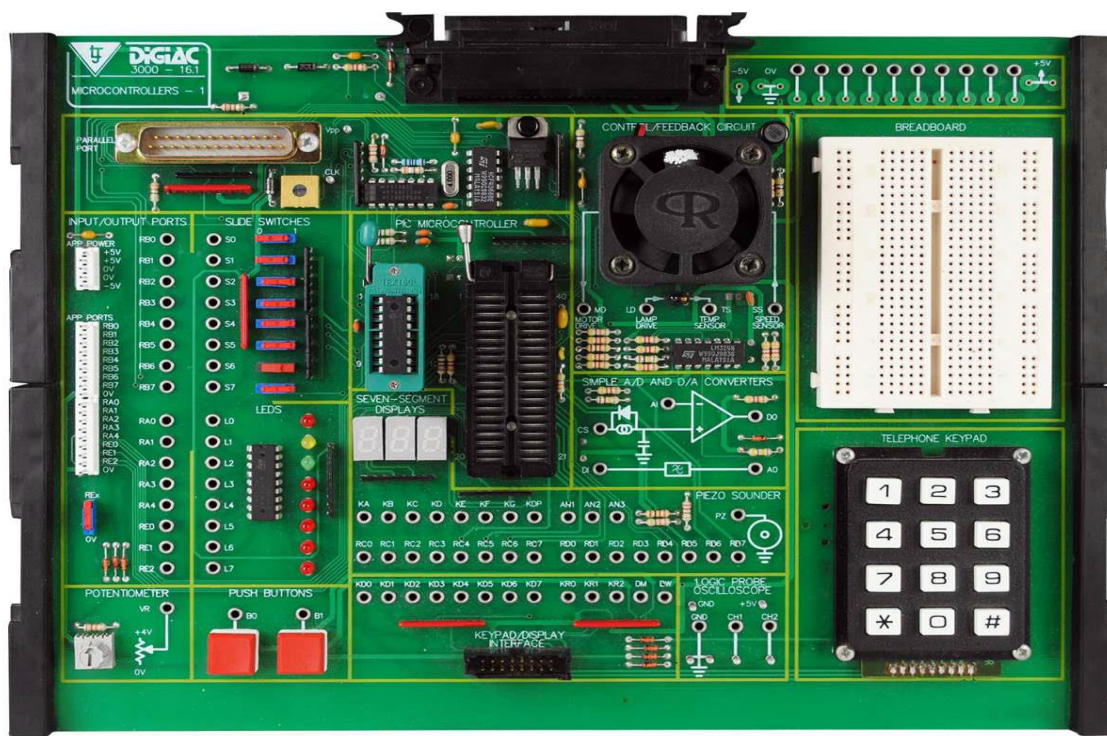


Figura 124. Placa de circuitos DIGIAC 3000

MASTER PROG

El programador Master Prog fue diseñado para brindar un óptimo desempeño y versatilidad de uso con el puerto USB, su controlador interno emplea tecnología CMOS y por tanto debe tomarse ciertas precauciones.

Instalación en 32 y 64 bits windows.

Dispositivos soportados:

Familia Baseline (Series 10Fxxx, 12F5xx, 16F50x): Selecciona la familia de PIC's con núcleo de 12 bits, entre los cuales están el 12F508, 16F505, 10F200, etc. (éstos deben seleccionarse de forma manual).

Familia Midrange (Serie 16Fxxx, 12F5xx, 16F50x): Selecciona la familia de PIC's con núcleo de 14 bits, entre los cuales están el 16F628A, 16F873A, 16F877A, etc. (El número de PIC es seleccionado automáticamente al realizar cualquier operación, por

ejemplo lectura, checar programador, etc.)

Familia PIC18F y PIC18F K: Selecciona la familia de PIC's de alto desempeño (llamados también High performance), entre los cuales están el 18F452A, 18F4539A, etc. (El número de PIC es seleccionado automáticamente al realizar cualquier operación, por ejemplo lectura, verificación, etc.)

Familia PIC18F J: Selecciona la familia de PIC's de alto desempeño y bajo voltaje (3.3V), entre los cuales están el 18F24J10, 18F65J10, etc. (Nota: Debido a que estos PIC se alimentan con 3.3V, requieren alimentación externa y deben conectarse al programador por medio del puerto ICSP, pero sin usar VDD.)

Familia PIC24: Selecciona la familia de PIC's con arquitectura interna de 16 bits, entre los cuales están el PIC24FJ64GA008, 18F65J10, PIC24FJ128GA010, etc. (Nota: Debido a que estos PIC se alimentan con 3.3V, requieren alimentación externa.

Familia dsPIC30: Selecciona la familia de dsPIC's con arquitectura interna de 16 bits (Nota: Estos dsPIC's deben conectarse al programador por medio del puerto ICSP, usando las 5 terminales).

Familia dsPIC33 y dsPIC30 SMPS: Selecciona la familia de dsPIC's con arquitectura interna de 16 bits (Nota: Debido a que estos dsPIC's se alimentan con 3.3V, requieren alimentación externa y deben conectarse al programador por medio del puerto ICSP+, pero sin usar VDD.)

Familia KEELOQ HCS: Selecciona la familia de dispositivos de seguridad electrónicos "KEELOQ", los cuales funcionan como "cerraduras electrónicas" en aplicaciones como alarmas, sistemas de seguridad, control de acceso, etc. Se deben programar por medio del puerto ICSP+.

Familia EEPROMS: Selecciona la familia de memorias EEPROM seriales de tipo I2C y Microwire de las series 24xxx, 25xxx, 93xxxA (8 bits), 93xxxB (16 bits) y 93xxxC (8/16 bits). Las familias 24xxx y 93xxx son soportadas por la base ZIF de 18 pines y el puerto ICSP+. La familia 25xxx se debe programar por medio del puerto ICSP+. El "JUMPER" del programador permite seleccionar el formato de la memoria 93xxxC entre 8 ó 16 bits. Si la memoria es de tipo I2C (serie 24xxx) el "JUMPER" se usa para seleccionar el tamaño en bits, entre 1Mb como la 24C1025 y menor de 1Mb (512, 256, 128, etc.)

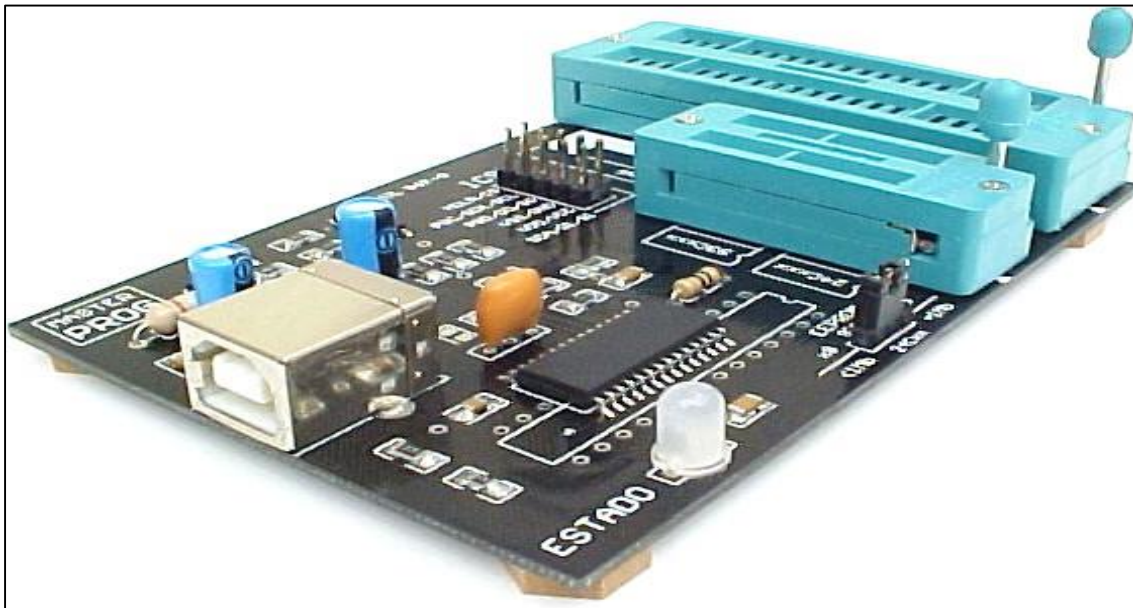


Figura 125. Mater Prog



INSTITUTO TECNOLÓGICO SUPERIOR DE TANTOYUCA

FORMATO DE REGISTRO DE PROYECTO

Departamento DE INGENIERÍA EN ELECTRÓNICA

Lugar: TANTOYUCA VERACRUZ

Fecha: 25 DE JUNIO DE 2019

Nombre del Proyecto	PUESTA EN MARCHA Y ELABORACIÓN DE UN MANUAL DE PRÁCTICAS CON EL MÓDULO PIC3000
Nombre del Asesor	MII PABLO IVAN ROMERO DE LA ROSA
Nombre del Revisor (1)	M.C. ORLANDO MEZA ZALETÁ
Nombre del Revisor (2)	M.C. RAUL HERNANDEZ RIVERA
Nombre del Suplente	M.C. VICENTE CRUZ FRANCO
Numero de estudiantes	1

Datos del / de los estudiantes:

Nombre:	MARISOL MARTINEZ MARTINEZ
No. De Control:	14350011
Carrera:	INGENIERÍA EN ELECTRÓNICA
Nombre:	_____
No. De Control:	_____
Carrera:	_____
Nombre:	_____
No. De Control:	_____
Carrera:	_____

Observaciones: NO APLICA

C.c.p Depto de Estudios Profesionales y Titulación
Archivo

[Handwritten signature]
28/06/19



INSTITUTO TECNOLÓGICO SUPERIOR DE TANTOYUCA

FORMATO DE LIBERACION DEL PROYECTO PARA TITULACION INTEGRAL

Ciudad: Tantoyuca Estado: Veracruz Fecha: 25 de Junio de 2019

Asunto: **Liberacion De Proyecto Para La Titulacion Integral**

MC SONIA CRUZ RIVERA

Jefe De Departamento De Estudios Profesionales Y Titulación

Por este medio le informo que ha sido liberado el siguiente proyecto para la Titulación integral:

Nombre del Alumno:	MARISOL MARTINEZ MARTINEZ
Carrera	INGENIERÍA EN ELECTRÓNICA
No. De Control	143S0011
Nombre del Proyecto	PUESTA EN MARCHA Y ELABORACIÓN DE UN MANUAL DE PRÁCTICAS CON EL MÓDULO PIC3000
Producto:	TITULACIÓN INTEGRAL: TESIS

- Lo anterior lo hago de su conocimiento para los fines correspondientes a su Examen Profesional, por lo cual debiera de entregar al Departamento de Estudios Profesionales, su trabajo profesional en 5 mini CD's (debidamente rotulados) de su trabajo en archivo PDF, así como Donar un libro (nuevo) de su especialidad al centro de informacion.

Agradezco de antemano su valioso apoyo en esta importante actividad para la formacion profesional de nuestros egresados:

ATENTAMENTE

JEFE DE DIVISION DE CARRERA DE INGENIERIA EN ELECTRÓNICA

MII PABLO IVAN ROMERO DE LA ROSA	M.C. ORLANDO MEZA ZALET	M.C. RAUL HERNANDEZ RIVERA
Nombre y firma del Asesor	Nombre y Firma del Revisor	Nombre y firma del Revisor

C.c.p Depto. De Servicios Escolares.- Para carta de no inconveniencia
Depto. De Recursos Financieros.- Para el cobro de arancel correspondiente
Depto. Estudios Profesionales
Expediente

R00/05/14

F-EP-34



INSTITUTO TECNOLÓGICO SUPERIOR DE TANTOYUCA

FORMATO PARA INTEGRAR LA COMISION DE JURADOS PARA EL EXAMEN
PROFESIONAL DE LA TITULACION INTEGRAL

Tantoyuca, Veracruz; a 27 de JUNIO del 2019

Asunto: Asignacion de fecha y Comision de Jurados para el examen profesional

Por este medio le informo que ha sido integrado el jurado que llevara a cabo el Acto de examen profesional, de acuerdo a la normatividad de la Opcion: Titulación Integral :
TESIS.

Funгиendo como :

PRESIDENTE:	M.I.I. PABLO IVÁN ROMERO DE LA ROSA
SECRETARIO:	M.C. ORLANDO MEZA ZALETА
VOCAL:	M.C. RAÚL HERNÁNDEZ RIVERA
SUPLENTE:	M.C. VICENTE CRUZ FRANCO

Donde sustentará el Examen Profesional del o la ciudadano (a):

NOMBRE:	MARISOL MARTINEZ MARTINEZ
EGRESADO DE LA CARRERA:	INGENIERÍA ELECTRÓNICA
CON NUMERO DE CONTROL:	143S0011
PRODUCTO	TITULACION INTEGRAL: TESIS
CON EL TRABAJO:	"PUESTA EN MARCHA Y ELABORACIÓN DE UN MANUAL DE PRÁCTICAS CON EL MÓDULO PIC3000"

El evento se efectuará el día **JUEVES 11 de JULIO del 2019 a las 10:00 hrs** en el Sala de Juntas de la Biblioteca del ITSTA.

Sin mas por el momento, reciba un cordial saludo.

ATENTAMENTE
MC SONIA CRUZ RIVERA
JEFE DE DEPARTAMENTO DE ESTUDIOS PROFESIONALES Y TITULACIÓN

C.c.p Depto de Servicios Escolares
Sala de Juntas
Alumno
Archivo

R00/05/14

F-EP-37