



UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

FACULTAD DE EDUCACIÓN TÉCNICA PARA EL DESARROLLO

CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

TEMA:

**APLICACIONES PRÁCTICAS DE MICROCONTROLADORES A TRAVÉS DE
LA PLATAFORMA DE PROGRAMACIÓN MATLAB**

Previa la obtención del Título

INGENIERO EN TELECOMUNICACIONES

ELABORADO POR:

Alex Josué González Linch

Guayaquil, 20 de Septiembre del 2013



UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

CERTIFICACIÓN

Certifico que el presente trabajo fue realizado en su totalidad por el Sr.
Alex Josué González Linch como requerimiento parcial para la obtención del
título de INGENIERO EN TELECOMUNICACIONES.

Guayaquil, 20 de Septiembre del 2013

DIRECTOR

MsC. Edwin Palacios Meléndez

REVISADO POR

Ing. Marcos Montenegro Tamayo.
Revisor Metodológico

MsC. Luis Pinzón Barriga.
Revisor de Contenido



UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

INGENIERÍA EN TELECOMUNICACIONES

DECLARACIÓN DE RESPONSABILIDAD

ALEX JOSUÉ GONZÁLEZ LINCH

DECLARAMOS QUE:

El proyecto de tesis denominado “Aplicaciones prácticas de Microcontroladores a través de la Plataforma de Programación MatLab” ha sido desarrollado con base a una investigación exhaustiva, respetando derechos intelectuales de terceros conforme las citas que constan al pie de las páginas correspondientes, cuyas fuentes se incorporan en la bibliografía.

Consecuentemente este trabajo es de mi autoría.

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance científico del proyecto de grado en mención.

Guayaquil, 20 de Septiembre del 2013

EL AUTOR

ALEX JOSUÉ GONZÁLEZ LINCH



UNIVERSIDAD CATÓLICA
DE SANTIAGO DE GUAYAQUIL

INGENIERÍA EN TELECOMUNICACIONES

AUTORIZACIÓN

Yo, ALEX JOSUÉ GONZÁLEZ LINCH

Autorizó a la Universidad Católica de Santiago de Guayaquil, la publicación, en la biblioteca de la institución del proyecto titulado: “Aplicaciones prácticas de Microcontroladores a través de la Plataforma de Programación MatLab”, cuyo contenido, ideas y criterios es de mi exclusiva responsabilidad y autoría.

Guayaquil, 20 de Septiembre del 2013

EL AUTOR

ALEX JOSUÉ GONZÁLEZ LINCH

DEDICATORIA

Dedico este proyecto de tesis a Dios sobre todas las cosas, y a mis padres.

A Dios porque ha estado conmigo en todo momento, cuidándome y dándome fortaleza para continuar, a mis padres, quienes a lo largo de mi vida han estado pendientes de mi bienestar y educación. Depositando su entera confianza en cada reto que se me presentaba sin dudar ni un solo momento en mi inteligencia y capacidad.

.

EL AUTOR

ALEX JOSUÉ GONZÁLEZ LINCH

AGRADECIMIENTO

Agradezco este proyecto a mi director de tesis, MsC. Edwin Palacios Meléndez , quien a lo largo de este tiempo ha puesto a prueba sus capacidades y conocimientos en el desarrollo de este nuevo plan para poderme guiar a concluir esta Tesis. A mis padres quienes a lo largo de toda mi vida han apoyado y motivado mi formación académica. A mis profesores a quienes les debo gran parte de mis conocimientos, gracias a su paciencia y por ultimo agradezco a la Universidad Católica Santiago de Guayaquil por prepararme para un futuro competitivo y formarme como persona de bien.

EL AUTOR

ALEX JOSUÉ GONZÁLEZ LINCH

Índice General

Índice de Figuras	10
Resumen	14
CAPÍTULO 1: GENERALIDADES DEL TRABAJO DE TITULACIÓN.....	15
1.1. Introducción.....	15
1.2. Antecedentes.....	15
1.3. Justificación del Problema.	16
1.4. Definición del Problema.....	16
1.5. Objetivos de la Investigación.	17
1.5.1. Objetivo General.	17
1.5.2. Objetivos Específicos.....	17
1.6. Idea a Defender.....	17
1.7. Metodología de Investigación.....	17
CAPÍTULO 2: Estado del Arte de Sistemas Microcontroladores.	18
2.1. Introducción a los Sistemas de Microcontroladores.	18
2.2. Los sistemas de microcontroladores	19
2.2.1. Memoria RAM.....	24
2.2.2. Memoria ROM.....	25
2.2.3. Memoria EPROM.	25
2.2.4. Memoria EEPROM.....	25
2.2.5. Memoria EEPROM Flash	26
2.3. Los recursos de los microcontroladores	26
2.3.1. Tensión de alimentación	26
2.3.2. Oscilador de Reloj.	28
2.3.3. Temporizadores.....	30
2.3.4. Perro guardián (Watchdog).....	31
2.3.5. Circuito Reset	32
2.3.6. Interrupciones.....	32
2.4. Arquitecturas de los Microcontroladores.....	33
2.4.1. Arquitectura Von Neumann.....	33
2.4.2. Arquitectura Harvard.	34

2.5.	Repertorio de Instrucciones RISC y CISC	34
2.6.	La Familia de Microcontroladores PIC.....	35
CAPÍTULO 3: PROGRAMACIÓN VIRTUAL MATLAB.		39
3.1.	Introducción a MATLAB.....	39
3.2.	Iniciando con Matlab	39
3.3.	Command Window.	42
3.4.	Launch Pad.....	43
3.4.1.	Command History Browser.....	43
3.4.2.	Current Directory Browser.	43
3.4.3.	Workspace Browser y Array Editor.....	45
3.5.	El Editor/Debugger	47
3.6.	Espacio de trabajo.....	50
3.7.	Variables.	51
3.8.	Formato de números.....	53
3.9.	Programas	54
3.10.	Funciones	55
3.10.1.	Reglas de construcción de funciones.....	57
3.10.2.	Funciones en línea.....	58
3.11.	Números Complejos.....	59
3.12.	Manejo de vectores y matrices.....	60
3.13.	Tipos de Gráficos.	61
3.14.	Entornografico MatLab (GUIDE).....	62
3.14.1.	Inicio.....	62
3.14.2.	Propiedad de los componentes.....	65
3.14.3.	Funcionamiento de una aplicación GUI.....	66
3.14.4.	Manejo de datos entre los elementos de la aplicación y el archivo .m	66
3.14.5.	Sentencias Get y Set	67
CAPÍTULO 4: DESARROLLO EXPERIMENTAL.....		68
4.1.	Hardware.....	68
4.1.1.	Módulo de entrenamiento.....	68
4.1.2.	Programador P.PICI&T04	70
4.1.3.	Driver: P.H. I&T 04.....	71
4.2.	Programando el Hardware	72
3.14.6.	Uso de contador y Leds.	73

3.14.7. Uso de Botonera – Contador y Leds.....	74
4.3. Software: Programación gráfica en MatLab (GUIDE).....	76
4.3.1. Ejemplo: Programación GUI – Sumador de dos Números.	76
4.3.2. Ejemplo: Graficador de Funciones.....	80
4.4. Comunicación Serial.....	83
4.4.1. Ejemplo: Envío de datos por comunicación serial	84
4.4.2. Recepción/envío de datos por comunicación serial MATLAB	86
4.4.3. Pruebas.....	90
4.5. Validación Práctica del Control de un Motor DC con PWM y monitoreo de las RPM del motor con encoder óptico desde una interfaz gráfica (Matlab-GUIDE).....	93
CAPÍTULO 5: CONCLUSIONES Y RECOMENDACIONES.....	100
5.1. Conclusiones.	100
5.2. Recomendaciones.....	100
REFERENCIAS BIBLIOGRÁFICAS.....	101

Índice de Figuras

Capítulo 2

Figura 2. 1: Microcontrolador PIC.....	21
Figura 2. 2: Sistema de Control de la Temperatura de un horno.....	22
Figura 2. 3: Sistema de Control de la Temperatura con teclado y LCD (visualizador).....	22
Figura 2. 4: Controlador de Temperatura más sofisticado.	23
Figura 2. 5: Arquitectura mínima de un Microcontrolador.....	24
Figura 2. 6: Conexión básica de un Microcontrolador PIC.	27
Figura 2. 7: Conexión a un oscilador externo.	28
Figura 2. 8: Cristal de cuarzo de 1 MHz.	28
Figura 2. 9: Oscilador externo en modo EC.	29
Figura 2. 10: Oscilador externo en modo LP, XT o HS.	29
Figura 2. 11: Oscilador externo RC (resonante – capacitivo).....	29
Figura 2. 12: Temporizador Timer0, Timer1 y Timer2.	30
Figura 2. 13: Diagrama de bloques del Watchdog.	31
Figura 2. 14: Conexión de Reset al microcontrolador.	32
Figura 2. 15: Arquitectura Von Neumann.	34
Figura 2. 16: Arquitectura Harvard.	34

Capítulo 3

Figura 3. 1: Ventanas de Matlab	40
Figura 3. 2: Workspace Browser con elementos definidos y Array Editor (Editor de Matrices).....	46
Figura 3. 3: Ventana del Editor/Debugger y Ejecución interactiva con el Editor/Debugger	47
Figura 3. 4: Comando <i>Preferences</i> del menú <i>File</i>	50
Figura 3. 5: Representación gráfica de Fun.	56
Figura 3. 6: Diferentes tipos de gráficos.....	62
Figura 3. 7: Ventana de MatLab para iniciar programación gráfica GUIDE.....	63
Figura 3. 8: Ventana de GUIDE Quick Start.	63

Figura 3. 9: Ventana del Entorno Gráfico (GUIDE) de MatLab.....	64
Figura 3. 10: Barra de Herramientas del Entorno Gráfico (GUIDE) de MatLab.....	64
Figura 3. 11: Paleta de Componentes del GUI de MatLab.....	65
Figura 3. 12: Propiedades de paleta de Componentes del GUI de MatLab.	65

Capítulo 4

Figura 4. 1: Módulo de entrenamiento MEI&T.....	68
Figura 4. 2: Puertos de Entrada y Salida (E/S) del MEI&T.....	69
Figura 4. 3: Programador P.PIC1&T04.	70
Figura 4. 4: Programador P.PIC1&T04.	71
Figura 4. 5: Pasos para la programación.	72
Figura 4. 6: Descripción de pines del PIC16F886.	73
Figura 4. 7: Diagrama de bloques para encendido indefinido de LEDs.....	73
Figura 4. 8: Programación en MikroBasic del encendido indefinido de LEDs. .	74
Figura 4. 9: Conexión entre el programador y módulo de entrenamiento.	74
Figura 4. 10: Diagrama de bloques de botonera para encendido de LEDs.....	75
Figura 4. 11: Programación en MikroBasic de la botonera para encender LEDs.	75
Figura 4. 12: Diagrama de bloques de entorno gráfico.	76
Figura 4. 13: Diagrama de bloques de entorno gráfico.	76
Figura 4. 14: Programación gráfica del sumador de dos números.....	77
Figura 4. 15: Código de objetos para el sumador de dos números.....	78
Figura 4. 16: Subrutina del sumador de dos números.....	78
Figura 4. 17: Código fuente que guarda primer número.....	79
Figura 4. 18: Código fuente que guarda segundo número.	79
Figura 4. 19: Código fuente de la suma de dos números.....	80
Figura 4. 20: Ventana GUI para la suma de dos números.	80
Figura 4. 21: Ventana GUI para graficar funciones.	81
Figura 4. 22: Asignación de String para gráficas de funciones.....	81
Figura 4. 23: Código fuente para el inicio de una función - 1.	82
Figura 4. 24: Código fuente para el inicio de una función - 2.	82
Figura 4. 25: Asignación de String para gráficas de funciones.....	83

Figura 4. 26: Código fuente para graficar funciones.....	83
Figura 4. 27: Código fuente para graficar funciones.....	84
Figura 4. 28: Código fuente para comunicación serial.	84
Figura 4. 29: Configuración del Access Port.	85
Figura 4. 30: Comunicación Hiperterminal.	85
Figura 4. 31: Configuración del Access Port.	86
Figura 4. 32: Código fuente para configuración de baudios.	87
Figura 4. 33: Código fuente para configuración y selección de los puertos COM.	87
Figura 4. 34: Código fuente para configuración de pushbutton1.....	88
Figura 4. 35: Código fuente para configuración de pushbutton2.....	88
Figura 4. 36: Código fuente para configuración de envío de datos.	88
Figura 4. 37: Código fuente para configuración pushbutton3 para enviar datos.	89
Figura 4. 38: Código fuente para configuración pushbutton4 para recepción de datos.	89
Figura 4. 39: Código fuente para configuración pushbutton8 para salir del programa.....	89
Figura 4. 40: Creación de puertos virtuales.....	90
Figura 4. 41: Configuración del puerto.	91
Figura 4. 42: Dato “ideas” enviado por el puerto virtual Access Port.....	91
Figura 4. 43: Access Port para enviar datos y lectura desde MatLab.	92
Figura 4. 44: Datos recibidos y lectura en GUI de MatLab.....	92
Figura 4. 45: Salida del programa GUI de MatLab.	93
Figura 4. 46: Conexiones para validación práctica.....	93
Figura 4. 47: Programación para control y monitoreo del motor DC.	94
Figura 4. 48: Programación del contador de RPS.....	94
Figura 4. 49: Programación del Main Principal.....	95
Figura 4. 50: Interfaz Gráfica GUI para configuración de puertos y panel de control.	96
Figura 4. 51: Programación del panel de Monitoreo y Control.	97
Figura 4. 52: Programación Pushbutton – Graficar.	98

Figura 4. 53: Programación Pushbutton – Stop.....	98
Figura 4. 54: Programación Pushbutton – Salir.....	99

Resumen

El propósito del presente trabajo de titulación, ha sido la de evaluar y aplicar los conocimientos adquiridos en la Carrera de Ingeniería en Telecomunicaciones, se han puesto en práctica el aprendizaje de los microcontroladores y de los puertos de comunicación serial, así como también la investigación de una herramienta de entorno o programación gráfica GUI de MatLab. Esta herramienta permitió crear programas con interfaz gráfica para enviar y recibir datos, y que sean compatibles con los microcontroladores, lo interesante fue la búsqueda de las tramas para la comunicación serial entre el módulo de entrenamiento, encoder óptico y computador.

CAPÍTULO 1: GENERALIDADES DEL TRABAJO DE TITULACIÓN

1.1. Introducción.

En la actualidad existe diversidad de aplicaciones de microcontroladores, es decir a nivel de hardware, también se lo puede considerar como un sistema de microcontroladores o sistemas embebidos, porque se integran diferentes dispositivos electrónicos. Adicionalmente, estos sistemas a nivel de hardware se les puede adicionar una interfaz gráfica, como lo es GUIDE de MatLab. Esta programación gráfica permite la comunicación serial a los dispositivos electrónicos que integran al sistema embebido.

Estas aplicaciones ya son analizadas, investigadas y puestas en desarrollo en Brasil, Colombia, Perú, Estados Unidos y algunos países europeos. Las Carreras de Ingeniería en Electrónica y Telecomunicaciones de las Instituciones de Educación Superior (IES) del Ecuador, se encuentran innovando el proceso de aprendizaje de aplicaciones prácticas en la Ciencias Aplicadas.

1.2. Antecedentes.

Las Instituciones de Educación Superior (IES) del Ecuador, a través de las Carreras de Ingenierías en Telecomunicaciones y Electrónica, han realizado un sin número de trabajos de graduación que involucran a los microcontroladores, estos se pueden consultar en los repositorios de las siguientes Universidades: Escuela Superior Politécnica del Litoral (ESPOL)¹, Escuela Superior Politécnica del Ejército (ESPE)², Escuela Politécnica Nacional (EPN)³, Universidad Técnica de Ambato (UTA)⁴, Universidad Tecnológica

¹ Página web del Repositorio de la ESPOL: <http://www.dspace.espol.edu.ec/>

² Página web del Repositorio de la ESPE: <http://repositorio.espe.edu.ec/>

³ Página web del Repositorio de la EPN: <http://bibdigital.epn.edu.ec/>

⁴ Página web del Repositorio de la UTA: <http://repo.uta.edu.ec/>

Equinoccial (UTE)⁵, inclusive en el de la Universidad Católica de Santiago de Guayaquil (UCSG)⁶, etc.

Inclusive hay páginas de Universidades Extranjeras, las que también tienen disponibles sus repositorios digitales de trabajos de graduación, algunos permiten leer documentos completos, mientras que otros solo de manera interna, es decir, intranet.

Mientras que en la UCSG a través de la Facultad de Educación Técnica para el Desarrollo (FETD) se han desarrollado varios trabajos de titulación con diversas aplicaciones de microcontroladores, lo que ha permitido la participación de grupos de estudiantes en concursos de robótica.

1.3. Justificación del Problema.

Hasta la presente en la FETD se han desarrollado aplicaciones de microcontroladores, pero ninguna de estos trabajos de titulación ha desarrollado entornos gráficos a través de la plataforma de programación MatLab. Esta herramienta permite controlar y manipular las variables a través de la comunicación serial.

Finalmente, se puede decir, que las aplicaciones prácticas del presente trabajo de titulación pueden trabajar con: LEDs, Pushbutton, botoneras, sensores, señales DLR, comunicaciones OnWire, tonos buzzer, y para el trabajo de titulación se controla y manipula a un motor DC con PWM en lazo abierto y las RPM del motor con encoder óptico.

1.4. Definición del Problema.

Necesidad de realizar aplicaciones prácticas de microcontroladores a través de una herramienta robusta de programación de entorno gráfico, como

⁵ Página web del Repositorio de la UTE: <http://repositorio.ute.edu.ec/>

⁶ Página web del Repositorio de la UCSG: <http://repositorio.ucsg.edu.ec/>

lo es GUIDE de MatLab, para enviar y transmitir datos, desde el dispositivo electrónico embebido hacia un computador.

1.5. Objetivos del Problema de Investigación.

1.5.1. Objetivo General.

Contribuir con el desarrollo de aplicaciones prácticas de microcontroladores, a través de un entorno de programación gráfica (GUIDE), y que permita promover la investigación formativa.

1.5.2. Objetivos Específicos.

1. Describir el Estado del Arte de los Sistemas Microcontroladores y de la programación de entorno gráfico (GUIDE – Matlab),
2. Diseñar los programa de entorno gráfico en GUIDE de MatLab.
3. Validar el trabajo de titulación mediante una práctica.

1.6. Idea a Defender.

A través de las aplicaciones prácticas de microcontroladores mediante programación virtual MatLab, permitirá controlar y manipular al sistema embebido, y a la vez, se deja una excelente herramienta de aprendizaje a la Carrera de Ingeniería en Telecomunicaciones.

1.7. Metodología de Investigación.

La metodología empleada fue propia, pero se basa principalmente de una investigación cuantitativa con enfoque empírico, cuyo método son el Descriptivo y el Exploratorio.

CAPÍTULO 2: Estado del Arte de Sistemas Microcontroladores.

2.1. Introducción a los Sistemas de Microcontroladores.

En 1969, Bob Noyce y Gordon Moore crearon la Corporación Intel para fabricar chips de memoria para la industria de las súper computadoras/ordenadores. Posteriormente, en 1971, Intel fabricó el primer chip microprocesador 4040 para un consorcio de dos compañías japonesas. Estos chips se diseñaron básicamente en una de las primeras calculadoras portátiles, denominada Busicom. Ésta era una calculadora muy simple que sólo podía sumar y restar números de 4 bits (cuartetos o nibbles) en un instante de tiempo.

El chip 4040 tuvo tanto éxito que motivó la rápida aparición del microprocesador de 8 bits, el Intel 8008. Éste fue un microprocesador sencillo con recursos limitados, con buses de datos y direcciones multiplexados, y donde se implementaron ineficientemente los mecanismos de interrupción.

El primer microprocesador potente de 8 bits apareció en 1974 y fue el chip Intel 8080. Este microprocesador tenía los buses de datos y de direcciones separados, con un espacio de direcciones 64 KBytes, muy grande para los estándares de 1975. El microprocesador 8080 fue el primero que se usó con fines domésticos, a través de un ordenador personal (PC) denominado Altair. El 8080 fue un microprocesador de mucho éxito pero rápidamente otras compañías empezaron a fabricar chips microprocesadores.

Motorola presentó el chip 6800 de 8 bits que tenía una arquitectura diferente al 8080 y que también se hizo muy popular. En 1976, la casa Zilog introdujo el microprocesador Z80, mucho más avanzado que el 8080. El juego de instrucciones del Z80 era compatible con el 8080, lo cual propició que el Z80 se convirtiera en uno de los microprocesadores de mayor éxito de su época.

El Z80 se usó en muchas aplicaciones típicas de los microprocesadores, incluyendo los ordenadores personales y las consolas de juegos. En 1976, Motorola creó el microprocesador 6801 que reemplazó al 6800 y a otros chips adicionales que eran necesarios para completar un sistema computacional. Éste fue el paso más importante en la evolución de los microcontroladores, los cuales son, básicamente, ordenadores compuestos en un único chip.

En los años siguientes, han aparecido muchos microcontroladores en el mercado, como han sido: Intel 8048, 8049, 8051, Motorola 6809, Atmel 89C51, etc. El término microordenador (o microcomputadora) se usa para describir a un sistema que incluye como mínimo a los siguientes elementos: microprocesador, memoria del programa, memoria de datos y los puertos de entrada/salida (E/S).

Algunos sistemas de microordenadores incluyen componentes adicionales como temporizadores, contadores, conversores analógicos/digitales, entre otros. De este modo, un sistema microordenador puede abarcar desde un gran ordenador con discos duros, discos flexibles, e impresoras, hasta un controlador integrado en un único chip.

En el presente trabajo de titulación sólo se estudiarán los microordenadores integrados en un solo chip de silicio. Estos sistemas de microordenadores se denominan también microcontroladores y se usan en muchos artículos domésticos como hornos microondas, mandos para TV, cocinas, equipos de alta fidelidad, reproductores de CD/DVD, ordenadores personales, frigoríficos, etc.

2.2. Los sistemas de microcontroladores

Un microcontrolador es un ordenador de en un único chip (figura xxx). La palabra micro indica que el dispositivo es pequeño, y controlador indica que el dispositivo se puede usar en aplicaciones de control. Otro término usado para

los microcontroladores es el de controlador embebido, puesto que la mayoría de los microcontroladores se integran (o se embeben) junto con los dispositivos que se encargan de controlar.

Un microprocesador se diferencia de un microcontrolador en diversos aspectos. La diferencia principal estriba en que un microprocesador necesita varios componentes para su funcionamiento, como son la memoria de programa, la memoria de datos, los dispositivos de E/S y el circuito del reloj externo.(Valdivieso Noroña, 2013)

Un microcontrolador, por el contrario, tiene todos los circuitos integrados dentro del mismo chip. Los microcontroladores funcionan con un conjunto de instrucciones (o programa de usuario) almacenados en su memoria. El microcontrolador busca (una a una) las instrucciones en su memoria de programa, las decodifica y ejecuta las operaciones requeridas.

Los microcontroladores se programan tradicionalmente empleando el lenguaje ensamblador del dispositivo en cuestión. A pesar de que un programa en lenguaje ensamblador se ejecuta de forma rápida, este tiene varias desventajas. Un programa en ensamblador está compuesto por mnemónicos y es difícil su aprendizaje y mantenimiento. Además, los microcontroladores fabricados por diversas empresas tienen diferentes lenguajes ensambladores y el usuario debe aprender un nuevo lenguaje cada vez que emplee un nuevo microcontrolador.

Los microcontroladores también pueden programarse empleando un lenguaje de alto nivel, como BASIC, PASCAL y C. Los lenguajes de alto nivel tienen la ventaja de que son mucho más fáciles de aprender que el ensamblador. Asimismo, el uso de un lenguaje de alto nivel permite desarrollar programas grandes y complejos con una mayor facilidad.

En general, el único requerimiento de un sistema microcontrolador es que se encuentre en un único chip (véase la figura. En aplicaciones prácticas, se pueden necesitar otros componentes adicionales para realizar la interfaz del microcontrolador con el entorno. De acuerdo a Melchor (200), con la aparición de la familia de microcontroladores PIC, el tiempo de desarrollo de un proyecto electrónico se ha reducido a varias horas⁷. El desarrollo de un proyecto basado en un microcontrolador PIC se reduce simplemente a sólo cinco o seis pasos.

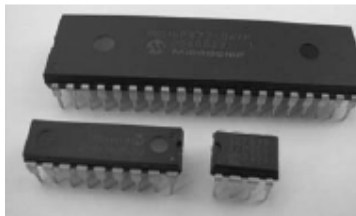


Figura 2. 1: Microcontrolador PIC.

Fuente: Dogan, Ibrahim. Programación de microcontroladores PIC.

1. Teclear el programa en un PC.
2. Ensamblar (o compilar) el programa.
3. Simular el programa en un PC (opcional).
4. Cargar el programa en la memoria de programa del PIC.
5. Diseñar y construir el hardware.
6. Probar el proyecto.

Básicamente, en un microordenador se ejecuta el programa de usuario que reside en su memoria de programa. Bajo el control de este programa, se reciben los datos desde los dispositivos externos (entradas), se manipulan y se envían a los dispositivos externos (salidas)(López Hernández & Zuñiga Castro, 2009). Por ejemplo, de acuerdo a Terven (2013): “en un sistema de control de temperatura de un horno, el microcontrolador lee el valor de temperatura empleando un sensor de temperatura. Luego, el microcontrolador actúa sobre un calefactor o un ventilador para controlar y mantener la temperatura en el

⁷ También disponible online de la página web: <http://itzamna.bnct.ipn.mx/dspace/bitstream/123456789/8682/1/205.pdf>

valor deseado. En la figura 2.2 se muestra el diagrama de bloques de este sencillo sistema de control de la temperatura del horno”.

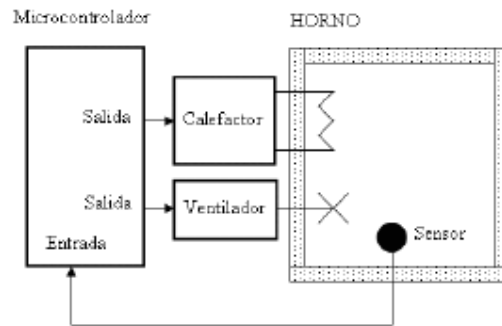


Figura 2. 2: Sistema de Control de la Temperatura de un horno.
Fuente: Dogan, Ibrahim. Programación de microcontroladores PIC.

El sistema mostrado en la figura 2.2 es un sistema de control de temperatura extremadamente simplificado. En un sistema más sofisticado, se podría disponer de un teclado pequeño para fijar la temperatura, y de un visualizador de cristal líquido (LCD) para mostrar la temperatura real. En la figura 2.3 se muestra el diagrama de bloques de este sistema más sofisticado de control de temperatura.

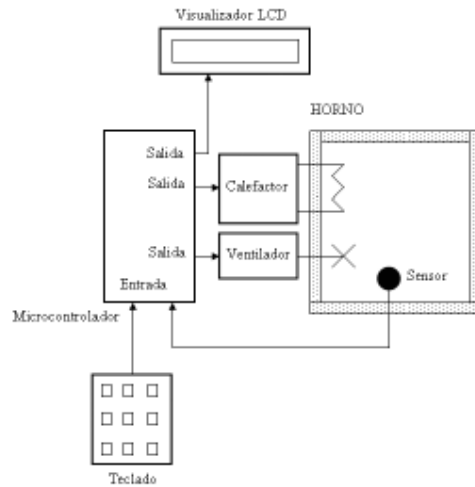


Figura 2. 3: Sistema de Control de la Temperatura con teclado y LCD (visualizador).
Fuente: Dogan, Ibrahim. Programación de microcontroladores PIC.

Se puede realizar un diseño más sofisticado (véase la figura 2.4) agregando una alarma sonora que informe si la temperatura se encuentra fuera

de los valores requeridos. Adicionalmente, se pueden enviar continuamente a un PC, los valores de temperatura leídos, para su almacenamiento y posterior procesamiento. Por ejemplo, se puede mostrar en un gráfico del PC la temperatura diaria. Como se puede observar, los microcontroladores, al ser programables, es muy fácil construir un sistema final tan simple o complicado como se quiera.

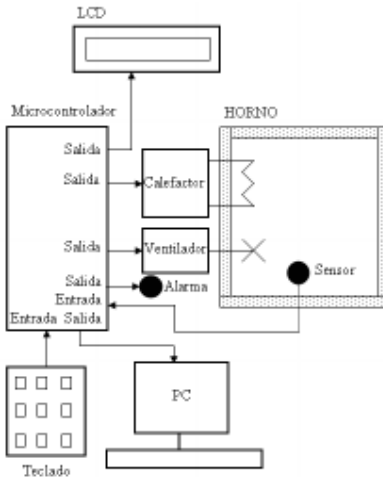


Figura 2. 4: Controlador de Temperatura más sofisticado.
Fuente: Dogan, Ibrahim. Programación de microcontroladores PIC.

Un microcontrolador es una poderosa herramienta que al diseñador le permite manipular los datos de E/S de forma sofisticada bajo el control de un programa (Melchor Hernández, 2009). Los microcontroladores se clasifican por el número de bits, los más populares y económicos son los microcontroladores de 8 bits su uso es muy frecuente para diversas aplicaciones; mientras que los microcontroladores de 16 bits y 32 bits son robustos, pero costosos y, normalmente, pero no son recomendables para aplicaciones de menor envergadura.

La arquitectura es del tipo Harvard, tal como se ilustra en la figura 2.5, la cual consiste de un microprocesador, la memoria interna, puertos de entrada y salida de datos (E/S). El microprocesador consta de la unidad de procesamiento central (CPU) y de la unidad de control (UC). La CPU es el cerebro del microcontrolador y en ella se ejecutan todas operaciones

aritméticas y lógicas. La Unidad de Control (UC) controla las operaciones internas del microprocesador y envía las señales de control a las demás partes del microcontrolador para realizar las acciones deseadas.

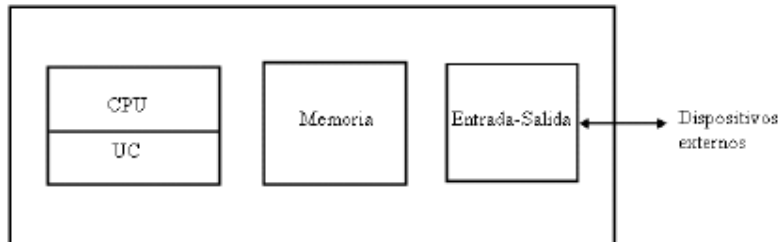


Figura 2. 5: Arquitectura mínima de un Microcontrolador.
Fuente: Dogan, Ibrahim. Programación de microcontroladores PIC.

La memoria es una parte importante de un sistema microcontrolador. En función del tipo de memoria utilizada se puede clasificar en dos grupos: la memoria de programa y la memoria de datos. En la memoria de programa se almacena el programa escrito por el programador o usuario; esta memoria es normalmente de tipo no volátil, es decir, los datos no se pierden al desconectar la fuente de alimentación. En la memoria de datos se almacenan los datos temporales usados por los programas; esta memoria es normalmente volátil, es decir, los datos se pierden al desconectar la alimentación. Básicamente, existen cinco tipos de memorias como se verá a continuación.

2.2.1. Memoria RAM.

RAM significa *Random Access Memory* (memoria de acceso aleatorio). Esta memoria es de propósito general y normalmente se usa para almacenar los datos usados por los programas. La memoria RAM es volátil, es decir, sus datos se pierden al desconectar la alimentación. La mayoría de los microcontroladores poseen una cierta cantidad de memoria RAM interna. Un valor típico es de 256 Bytes, aunque algunos microcontroladores pueden tener más y otros menos. En general, es posible aumentar la memoria añadiendo algunos chips externos de memoria.

2.2.2. Memoria ROM.

ROM equivale a Read Only Memory (memoria de sólo lectura). Este tipo de memoria almacena el programa del microcontrolador y los datos que son constantes. Las memorias ROM se programan durante el proceso de fabricación por lo que no se puede alterar su contenido. Las memorias ROM son útiles, solamente, si se ha desarrollado una versión final del programa y se desean fabricar varios miles de copias de estas.

2.2.3. Memoria EPROM.

EPROM significa Erasable Programmable Read Only Memory (Memoria de sólo lectura programable y borrable). Esta memoria es muy similar a la memoria ROM, pero la EPROM puede programarse con un dispositivo programador adecuado. Según lo enunciado por Dogan (2008), las memorias EPROM tienen una pequeña ventana de vidrio transparente colocada encima del chip para que sus datos puedan borrarse al aplicar luz ultravioleta (UV).⁸

Se fabrican muchas versiones de microcontroladores con memorias EPROM para el desarrollo de las aplicaciones. Estas memorias se borran y se reprograman hasta que el usuario quede satisfecho con el programa final. Nuevamente, Dogan (2008) indica que: “algunas versiones de EPROMs, conocidas como OTP (*One Time Programmable*, o programable una sola vez), pueden programarse empleando un dispositivo de grabación adecuado pero no pueden borrarse. Las memorias OTP son mucho más baratas que las EPROMs. La memoria OTP es útil después del desarrollo completo de un proyecto y se necesita realizar muchas copias de la memoria de programa”.

2.2.4. Memoria EEPROM

EEPROM significa Electrically Erasable Programmable Read Only Memory (memoria de sólo lectura, programable y borrable eléctricamente).

⁸ Aunque también se encuentra disponible online en la página web: <http://cidcamicros.blogspot.com/>

Según Dogan (2008), la memoria EEPROM: “se trata de una memoria no volátil por lo que se puede borrar y programar bajo el control del programa, utilizadas para guardar información de configuración, valores máximos y mínimos, datos de identificación, etc. Algunos microcontroladores tienen memorias EEPROM internas (por ejemplo, el PIC16F84 contiene una memoria EEPROM de 64 Bytes, donde cada Byte puede borrarse y programarse directamente a través del programa). Las memorias EEPROM son, normalmente, muy lentas”.⁹

2.2.5. Memoria EEPROM Flash

Se trata de otra versión de memoria del tipo EEPROM, que se ha difundido en aplicaciones con microcontroladores y se usa para almacenar el programa de usuario. La memoria EEPROM Flash es no volátil y, usualmente, su velocidad de trabajo es alta. El dato se borra y luego se reprograma mediante un dispositivo de programación. Para realizar la programación, es imprescindible borrar totalmente la memoria.

2.3. Los recursos de los microcontroladores

Los microcontroladores procedentes de los diversos fabricantes tienen diferentes arquitecturas y posibilidades. Algunos pueden ser adecuados para una aplicación en particular mientras que otros pueden ser totalmente inadecuados para la misma aplicación. En este apartado se describen los recursos de hardware con los que cuentan, generalmente, los microcontroladores.

2.3.1. Tensión de alimentación

En general los microcontroladores funcionan con una tensión lógica estándar de voltaje de $+5 V_{DC}$. Algunos microcontroladores pueden trabajar con

⁹ También disponible online en la página web: http://cidcamicros.blogspot.com/2011/09/lecturas-informativas_1442.html

una tensión de voltaje tan pequeña de $+2,7V_{DC}$ y otros toleran hasta $+6V_{DC}$ sin problemas. En las hojas de datos de los fabricantes se especifican los límites de la tensión de alimentación, por lo que se recomienda su verificación. En la figura 2.6 se muestra el esquemático básico de conexiones de un microcontrolador PIC.

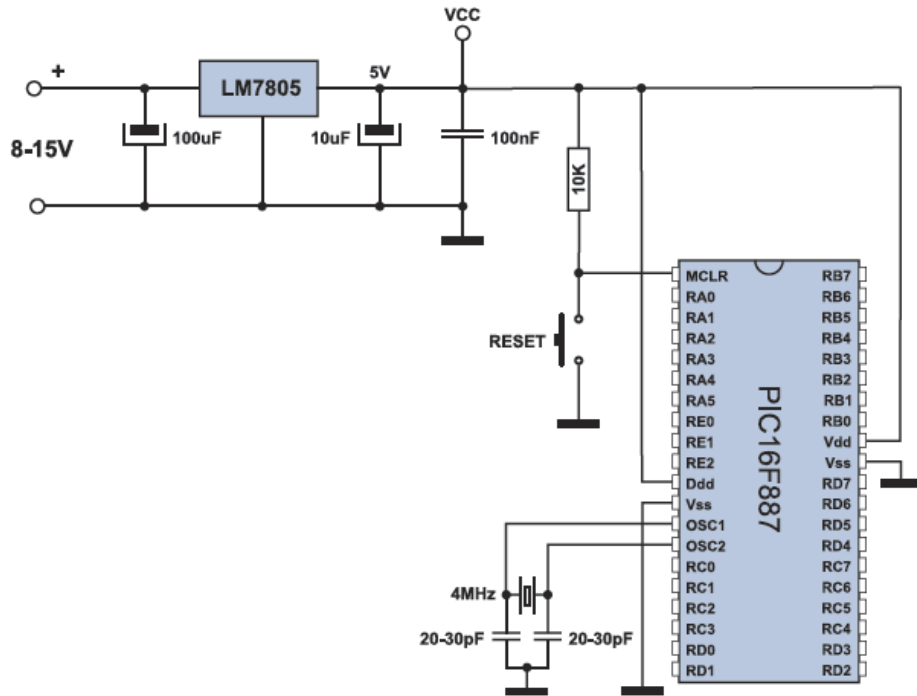


Figura 2. 6: Conexión básica de un Microcontrolador PIC.
Fuente:(Verle, 2010)

Para obtener la tensión de alimentación deseada se suelen usar los circuitos reguladores de tensión de voltaje como el LM7805 (véase la figura 2.6), independientemente de cómo se alimenta al dispositivo electrónico, si a través de baterías o con un cargador de celular.

Es decir, si el PIC16F887 que se muestra en la figura 2.6 opera con un voltaje de $+5V_{DC}$, pero el circuito se alimenta con un voltaje de $+9V_{DC}$, debemos regular el voltaje a $+5V_{DC}$ mediante el regulador LM7805.

2.3.2. Oscilador de Reloj.

Para que los microcontroladores funcionen correctamente necesitan de un oscilador de reloj. Tradicionalmente, el reloj se implementa, conectando dispositivos de temporización externos al microcontrolador, tal y como se muestra en la figura 2.7.

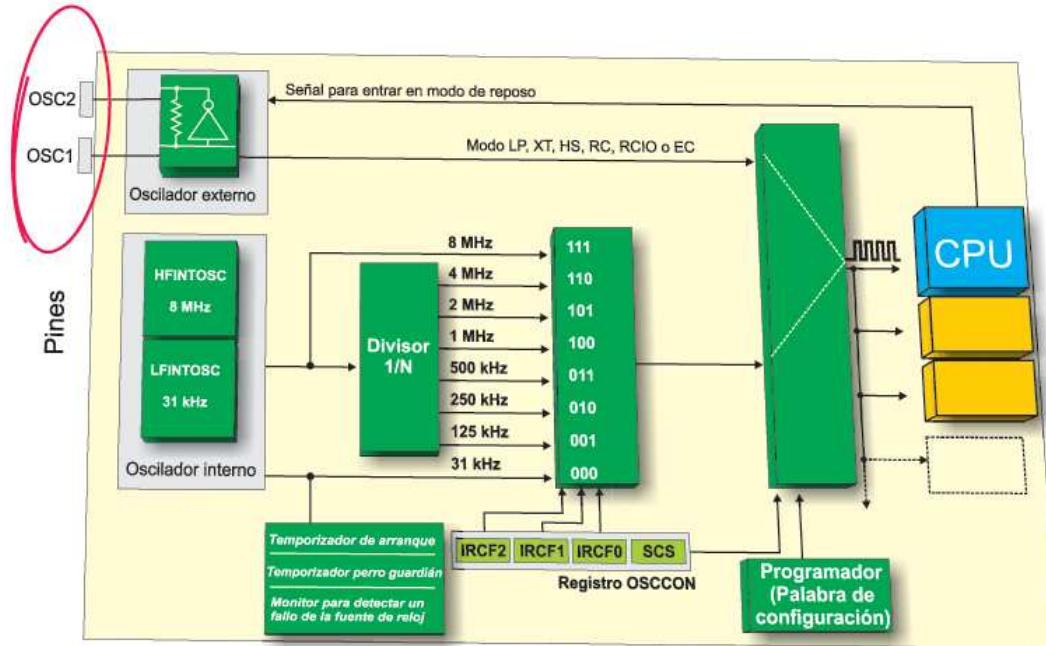


Figura 2. 7: Conexión a un oscilador externo.
Fuente:(Verle, 2010)

La mayoría de los microcontroladores pueden generar señales de reloj externamente si conectamos en OCS1 y OSC2 (ver figura 2.7) un cristal de cuarzo (véase figura 2.8) y dos capacitores pequeños.



Figura 2. 8: Cristal de cuarzo de 1 MHz.
Fuente:(Verle, 2010)

La figura 2.8 se muestran los osciladores externos en modo EC (external clock), mientras que la figura 2.9 se muestran los osciladores en modo LP (baja potencia), XT y HS(alta velocidad).

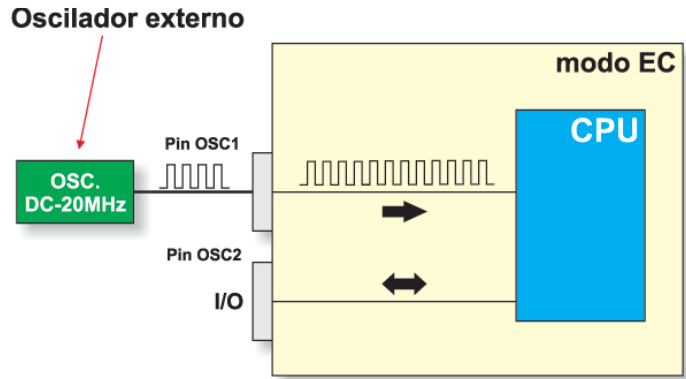


Figura 2. 9: Oscilador externo en modo EC.
Fuente:(Verle, 2010)

Mientras que otros microcontroladores pueden funcionar con resonadores o con un par resistor– capacitor (RC) externo (véase la figura 2.11).

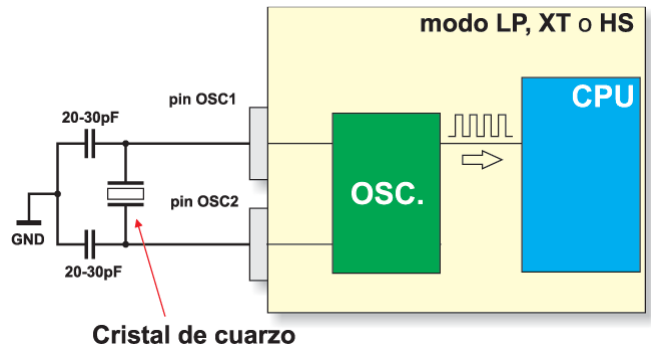


Figura 2. 10: Oscilador externo en modo LP, XT o HS.
Fuente:(Verle, 2010)

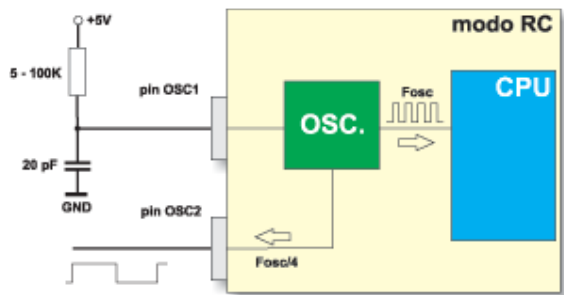


Figura 2. 11: Oscilador externo RC (resonante – capacitivo).
Fuente:(Verle, 2010)

Algunos microcontroladores poseen circuitos de temporización integrados y, por tanto, no requieren de ningún componente de temporización externo. Si la aplicación no requiere gran exactitud en cuanto a tiempo, y sí un diseño simple de bajo coste, entonces se pueden utilizar redes de temporización RC externas o internas (si se disponen). Para que el microprocesador ejecute una instrucción, primero debe buscarla en memoria y luego decodificarla. Ambas operaciones consumen varios ciclos de reloj y se conoce como ciclo de instrucción. En los microcontroladores PIC, un ciclo de instrucción dura cuatro ciclos de reloj. Así, el microcontrolador realmente funciona a una frecuencia de reloj que es la cuarta parte de la frecuencia real del oscilador.

2.3.3. Temporizadores

Melchor (2009) expresa que: “los temporizadores son un elemento importante en cualquier microcontrolador. Un temporizador es básicamente un contador que se activa con una señal de reloj interna o externa al microcontrolador. El temporizador puede ser de 8 bits o de 16 bits. Por programa, se puede cargar el valor de conteo del temporizador, así como arrancar y detener al mismo. La mayoría de los temporizadores pueden configurarse para generar una interrupción cuando se alcance un cierto número en el conteo (usualmente cuando se desbordan)”.

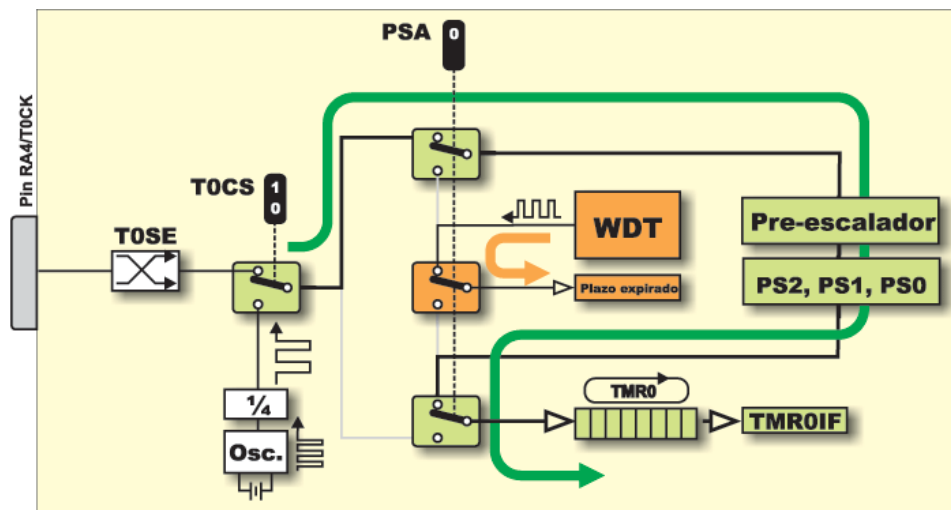


Figura 2. 12: Temporizador Timer0, Timer1 y Timer2.
Fuente:(Verle, 2010)

De acuerdo a lo mostrado por la figura 2.12, Verle (2010), explica a través de una aplicación en el PIC16F887 que dispone de tres temporizadores/contadores independientes, denominados Timer0, Timer1, Timer2. Adicionalmente, indica que el Timer0 tiene una gama de aplicaciones prácticas, muy convenientes y fáciles.

2.3.4. Perro guardián (Watchdog)

La mayoría de los microcontroladores cuentan con la posibilidad de tener un watchdog (perro guardián). El watchdog es básicamente un temporizador, el cual se debe refrescar cada cierto tiempo dentro del programa de usuario. Cuando esto no ocurre, el microcontrolador se reinicia de forma automática. El temporizador watchdog se usa para detectar algún problema en el sistema, como por ejemplo, que el programa caiga en un bucle infinito. La figura 2.13 muestra el diagrama de bloques del watchdog.

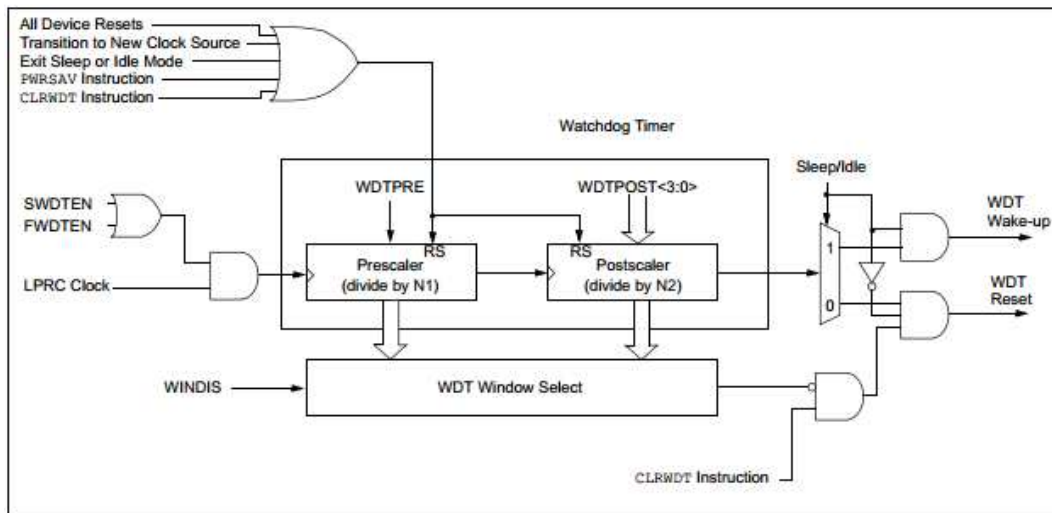


Figura 2. 13: Diagrama de bloques del Watchdog.

Fuente:

ftp://193.170.235.123/Lernbehelfe/PIC%20Handb%FCcher/dsPIC33F_Reference_Manual/9_Watch_Dog_Timer.pdf

El watchdog es un recurso de seguridad que evita la ejecución de un software sin control y que detiene la ejecución en el microcontrolador, de cualquier programa o código sin sentido e indeseado. Las facilidades del

watchdog son normalmente usadas en sistemas en tiempo real dónde se requiere verificar, regularmente, la culminación exitosa de una o más actividades.

2.3.5. Circuito Reset

El Reset es un pin de entrada, que permite reiniciar al microcontrolador. Según Dogan I. (2011): el Reset pone al microcontrolador en un estado conocido de forma tal que la ejecución del programa comienza en la dirección 0 de la memoria de programa. La acción del Reset externo se logra, normalmente, conectando un interruptor de tipo pulsador (push-button) a la entrada Reset, de forma que el microcontrolador puede inicializarse cuando se aprieta el interruptor, tal como se muestra en la figura 2.14(Dogan I. , 2011).

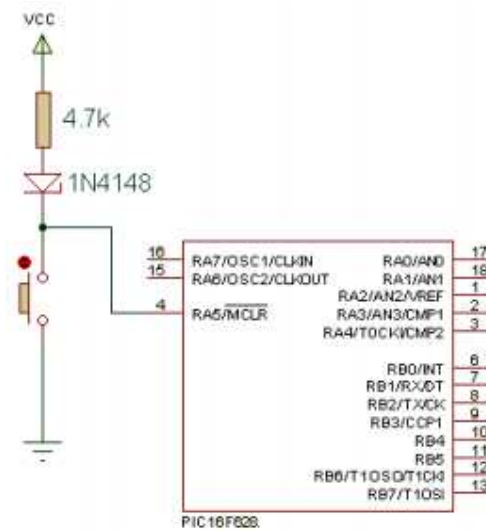


Figura 2. 14: Conexión de Reset al microcontrolador.
Fuente: http://www.galeon.com/oswagar2/Prac4_pic.pdf

2.3.6. Interrupciones.

Las interrupciones son un concepto muy importante en los microcontroladores. La interrupción permite que el microcontrolador responda rápidamente a eventos externos e internos (por ejemplo, un temporizador). Cuando ocurre una interrupción el microcontrolador sale del flujo normal de

ejecución de su programa y salta a una parte especial del programa, conocida como la *Interrupt Service Routine* (ISR) o rutina de atención a la interrupción.

Así, se ejecuta el código del programa dentro de la ISR y tras el retorno de la ISR, el programa reanuda su flujo normal de ejecución. La ISR comienza en una dirección fija de la memoria de programa. Esta dirección se conoce como dirección del vector de interrupciones. Por ejemplo, en el microcontrolador PIC16F84, la dirección de inicio de la ISR es la localización 4 de la memoria de programa.

Algunos microcontroladores que poseen múltiples interrupciones poseen una sola dirección del vector de interrupciones, mientras otros tienen varias direcciones del vector de interrupciones, una por cada fuente de interrupción. Las interrupciones pueden anidarse de forma que la solicitud de una nueva interrupción pueda detener la ejecución de otra interrupción. Otro aspecto importante de los microcontroladores con múltiples interrupciones es la posibilidad de que cada una de las fuentes de interrupción posea diferentes niveles de prioridad.

2.4. Arquitecturas de los Microcontroladores.

Los microcontroladores emplean normalmente dos tipos de arquitecturas:

- a) Von Neumann, y
- b) Harvard.

2.4.1. Arquitectura Von Neumann.

La figura 2.15 se muestra la arquitectura Von Neumann, muy utilizada por un elevado número de microcontroladores, la misma permite acceder a todo el espacio de memoria mediante un único bus compartido para las instrucciones y los datos.

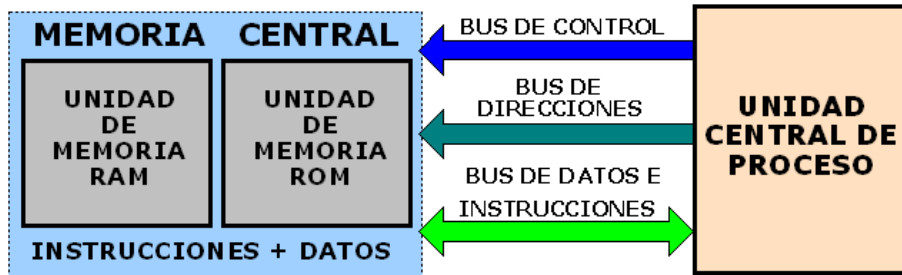


Figura 2. 15: Arquitectura Von Neumann.

Fuente: <http://hardware-technologies.blogspot.com/2012/09/modelos-de-von-neumann-y-harvard.html>

2.4.2. Arquitectura Harvard.

Mientras que la figura 2.16 muestra la arquitectura Harvard (usada por los microcontroladores PIC), las instrucciones y los datos se transfieren por buses diferentes, lo cual permite transferir códigos y datos de forma simultánea, y de esta forma obtener una mejor eficiencia en su funcionamiento.

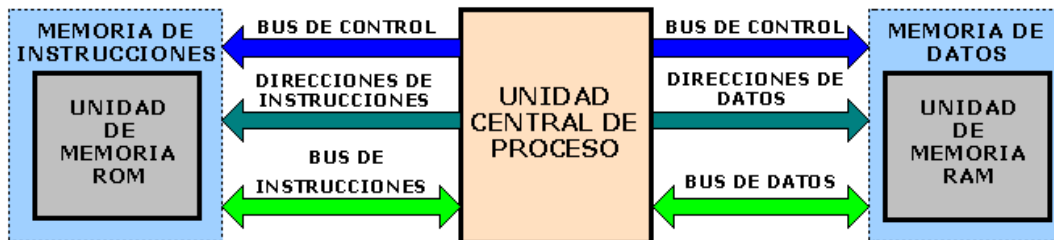


Figura 2. 16: Arquitectura Harvard.

Fuente: <http://hardware-technologies.blogspot.com/2012/09/modelos-de-von-neumann-y-harvard.html>

2.5. Repertorio de Instrucciones RISC y CISC

Con respecto al repertorio de instrucciones, los microcontroladores se clasifican en los siguientes tipos: RISC (Reduced Instruction Set Computer, ordenadores con un juego reducido de instrucciones) y CISC (Complex Instruction Set Computer, ordenadores con un juego de instrucciones complejo). En un microcontrolador RISC de 8 bits, la longitud de los datos es de 8 bits pero las instrucciones se codifican en palabras mayores de 8 bits

(normalmente 12, 14, o 16 bits) que ocupan una sola dirección en la memoria de programa.

Así, las instrucciones se buscan y se ejecutan en un único ciclo, lo cual mejora la eficiencia de su funcionamiento. Los microcontroladores PIC son dispositivos RISC que no poseen más de 35 instrucciones. En un microcontrolador CISC, tanto los datos como las instrucciones tienen una longitud de 8 bits. Los microcontroladores CISC normalmente tienen más de 200 instrucciones. Los datos y los códigos circulan por el mismo bus y no pueden transferirse de forma simultánea.

2.6. La Familia de Microcontroladores PIC.

La familia de microcontroladores PIC son fabricados por la firma Microchip Technology Inc. Estos microcontroladores son muy populares y se utilizan en diversas aplicaciones comerciales e industriales. Cada año se venden aproximadamente unos 120 millones de dispositivos. La arquitectura de los microcontroladores PIC es de tipo Harvard modificada, con un repertorio de instrucciones reducido (RISC) y dos buses de acceso, lo que ha proporcionado un diseño rápido y flexible, que ha evolucionado desde los dispositivos de 6 terminales hasta los de 80 terminales, con memorias de programa desde 384 Bytes hasta los 128 KBytes.

Los microcontroladores PIC se venden con diferentes especificaciones en función del:

a. Tipo de memoria:

- Flash.
- Programable una vez u OTP (One-time-programmable).
- Memoria de sólo lectura o ROM (Read-only-memory).
- Sin memoria ROM

b. Número de terminales de entrada/salida (E/S):

- 4 a 18 terminales.
- 20 a 28 terminales.
- 32 a 44 terminales.
- 45 y más terminales.

c. Tamaño de la memoria:

- 0,5 a 1 KByte.
- 2 a 4 KBytes.
- 8 a 16 KBytes.
- 24 a 32 KBytes.
- 48 a 64 KBytes.
- 96 a 128 KBytes.

d. Características especiales: -

- Bus CAN.
- Bus USB.
- LCD.
- Control de motores.
- Radiofrecuencia.

A pesar de que existen diversos modelos de microcontroladores PIC, un elemento positivo es que los nuevos modelos son compatibles con los precedentes. Así, un programa desarrollado para un modelo, puede ejecutarse en otros modelos de la familia de una forma muy fácil y, en la mayoría de los casos, sin modificaciones. El juego básico de instrucciones en lenguaje ensamblador consta de tan sólo 33 instrucciones y varios miembros de la familia (excepto los dispositivos más modernos) utilizan el mismo juego de instrucciones. Esto garantiza que un programa desarrollado para un modelo pueda ser ejecutado sin ningún cambio en otro modelo similar. Todos los microcontroladores PIC presentan los siguientes recursos:

- Juego de instrucciones reducido.
- Puertos digitales de E/S.
- Temporizador interno con prescalador de 8 bits.
- Reset durante el encendido. Temporizador watchdog.
- Modo de reposo y de bajo consumo.
- Alta corriente de entrega y absorción.
- Modos de direccionamiento directo, indirecto y relativo.
- Interfaz de reloj externa.
- Memoria de datos RAM.
- Memoria de programa EPROM o Flash.

Algunos dispositivos proporcionan los siguientes recursos adicionales:

- Canales de entrada analógicos.
- Comparadores analógicos.
- Circuitos temporizadores adicionales.
- Memoria de datos EEPROM.
- Interrupciones internas y externas.
- Oscilador interno.
- Salida modulada por tamaño del pulso.
- Interfaz serie USART.

Algunos dispositivos más complejos poseen los siguientes recursos:

- Interfaz con bus CAN (Controller Area Network).
- Interfaz con bus Interfaz con bus SPI.
- Interfaz directa LCD.
- Interfaz con bus USB (Universal Serial Bus).
- Control de motores.

A pesar de que existen varios cientos de modelos de microcontroladores PIC, la selección de un microcontrolador para una aplicación no es una tarea difícil y, para ello, debe tener en cuenta los siguientes factores:

- Número de terminales de E/S.
- Periféricos requeridos (por ejemplo USART, USB).
- Tamaño mínimo de la memoria del programa.
- Tamaño mínimo de memoria RAM.
- Necesidad del uso de memoria de datos no volátil de tipo EEPROM.
- Velocidad.
- Tamaño físico.
- Coste.

Un aspecto importante a tener en cuenta es que pudieran existir muchos modelos que satisfagan todos los requisitos anteriores. Entonces se deberá escoger el modelo que satisfaga los requisitos mínimos y que no ofrezca más de lo que se pueda necesitar. Por ejemplo, si se requiere un microcontrolador con sólo 8 terminales de E/S, y hay dos microcontroladores idénticos: uno con 8 terminales y otro con 16 terminales de E/S, se deberá seleccionar el de 8 terminales de E/S.

A pesar de que existen varios cientos de modelos de microcontroladores PIC, estos se pueden clasificar en los tres grupos siguientes:

- Los de palabras de instrucción de 12 bits (por ejemplo 12C5XX, 16C5X).
- Los de palabras de instrucción de 14 bits (por ejemplo 16F8X, 16F87X).
- Los de palabras de instrucción de 16 bits (por ejemplo 17C7XX, 18C2XX).

Los tres grupos comparten la misma arquitectura RISC y el mismo juego de instrucciones, con la particularidad de que los modelos de 14 bits presentan algunas instrucciones adicionales, y en los de 16 bits, el número de instrucciones es mucho mayor. Como las instrucciones ocupan sólo una palabra en la memoria, así se aumenta la eficacia del código y se reduce la memoria de programa requerida. Las instrucciones y los datos se transfieren por buses separados, por lo que se incrementa el rendimiento global del sistema.

CAPÍTULO 3: PROGRAMACIÓN VIRTUALMATLAB.

3.1. Introducción a MATLAB.

MatLab, desde las primeras versiones dispuso de help y demo, para iniciación rápida. La información suministrada a través de los menús de estas ayudas, correspondientes a las últimas versiones, crecieron de forma exponencial, siendo de utilidad práctica disponer de una guía en esta tesis tipo resumen de MatLab, en donde se encuentren los comandos de uso más frecuente, a la vez que se muestren sus aplicaciones prácticas en ejercicios, desde lo más sencillo, hasta otros de mayor complejidad. Este Capítulo es adecuado para principiantes absolutos, y de afianzamiento a los ya iniciados.

3.2. Iniciando con Matlab

Al arrancar MatLab, presenta una pantalla dividida en varias ventanas, configurables desde *Desktop Layout* del menú de View; en una de las ventanas estará el cursor parpadeando a la derecha de «>>», es la ventana de comandos desde donde se ejecutan los mismos, las otras son informativas:

```
>> 3+4
```

```
ans=
```

```
7
```

```
3*5
```

```
ans=
```

```
15
```

```
15/3
```

```
ans=
```

```
5
```

```
>> 15\3
```

```
ans =
```

```
0.2000
```

```
>> 2^3
```

ans =

8

>>sin(2*pi*30/360)

ans =

0.5000

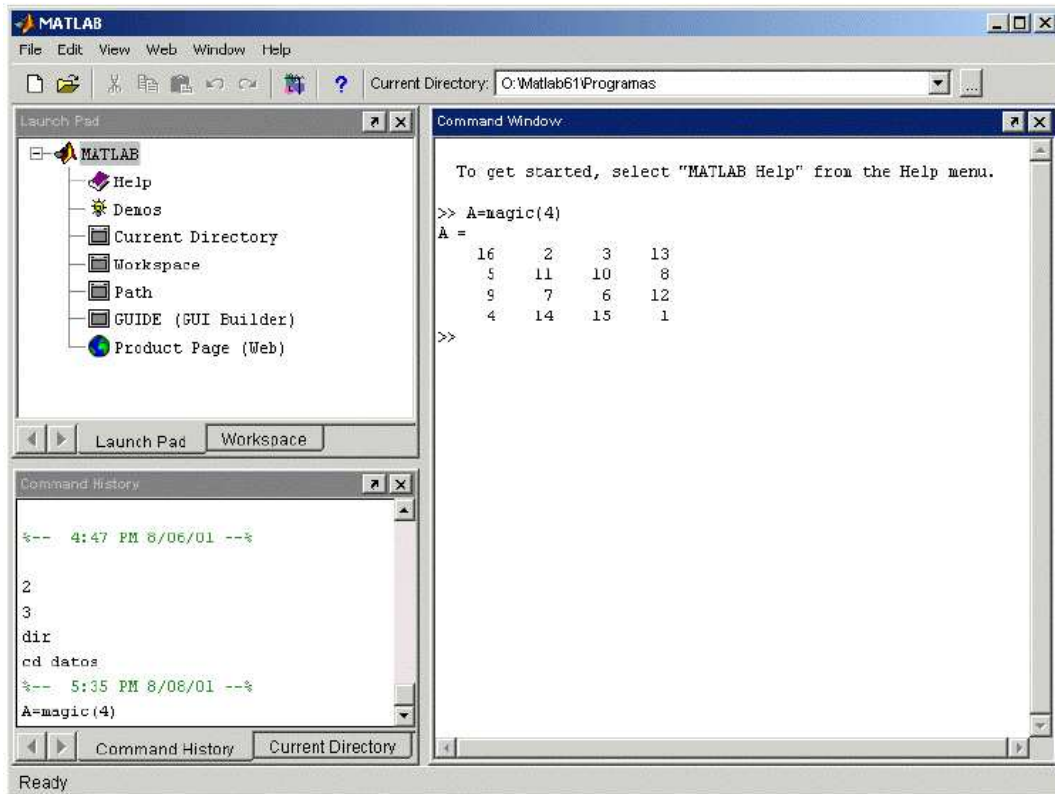


Figura 3. 1: Ventanas de Matlab

Fuente: Javier García de Jalón

La parte más importante de la ventana inicial es la **Command Window**, que aparece en la parte derecha. En esta sub-ventana es donde se ejecutan los comandos de MATLAB, a continuación del **prompt** (aviso) característico (>>), que indica que el programa está preparado para recibir instrucciones. En la pantalla mostrada en la Figura 1 se ha ejecutado el comando **A=magic (4)**, mostrándose a continuación el resultado proporcionado por MATLAB.

En la parte superior izquierda de la pantalla aparecen dos ventanas también muy útiles: en la parte superior aparece la ventana **Launch Pad**, que se puede alternar con **Workspace** clicando en la pestaña correspondiente. **Launch Pad** da acceso a todos los módulos o componentes de MATLAB que se tengan instalados, como por ejemplo al **Help** o a las **Demos**. El **Workspace** contiene información sobre todas las variables que se hayan definido en esta sesión.

En la parte inferior derecha aparecen otras dos ventanas, **Command History** y **CurrentDirectory**, que se pueden mostrar alternativamente por medio de las pestañas correspondientes. La ventana **Command History** muestra los últimos comandos ejecutados en la **Command Window**.

Estos comandos se pueden volver a ejecutar haciendo doble clic sobre ellos. Clicando sobre un comando con el botón derecho del ratón se muestra un menú contextual con las posibilidades disponibles en ese momento. Para editar uno de estos comandos hay que copiarlo antes a la **Command Window**. Por otra parte, la ventana **Current Directory** muestra los ficheros del directorio activo o actual.

A diferencia de versiones anteriores de MATLAB en que el directorio activo se debía cambiar desde la **Command Window**, a partir de la versión 6.0 se puede cambiar desde la propia ventana (o desde la barra de herramientas, debajo de la barra de menús) con los métodos de navegación de directorios propios de **Windows**. Clicando dos veces sobre uno de los ficheros ***.m** del directorio activo se abre el **editor de ficheros** de MATLAB, herramienta fundamental para la programación.

Puede hacerse que al arrancar se ejecute automáticamente un fichero, de modo que aparezca por ejemplo un saludo inicial personalizado. Esto se hace mediante un **fichero de comandos** que se ejecuta de modo automático cada

vez que se entra en el programa (el fichero **startup.m**, que debe estar en un directorio determinado, por ejemplo **C:\MatlabR12\Work**).

3.3. Command Window.

Ésta es la ventana en la que se ejecutan interactivamente las instrucciones de MATLAB y en donde se muestran los resultados correspondientes, si es el caso. En cierta forma es **la ventana más importante** y la única que existía en versiones anteriores de la aplicación. En esta nueva versión se han añadido algunas mejoras significativas, como las siguientes:

1. Se permiten líneas de comandos muy largas que automáticamente siguen en la línea siguiente al llegar al margen derecho de la ventana. Para ello hay que activar la opción **Wrap Lines**, en el menú **File/Preferences/Command Window**.
2. Clicando con el botón derecho sobre el nombre de una función que aparezca en esta ventana se tiene acceso a la página del **Help** sobre dicha función. Si el código fuente (fichero ***.m**) está disponible, también se puede acceder al fichero correspondiente por medio del **Editor/Debugger**.
3. Comenzando a teclear el nombre de una función y pulsando la tecla **Tab**, MATLAB completa automáticamente el nombre de la función, o bien muestra en la línea siguiente todas las funciones disponibles que comienzan con las letras tecleadas por el usuario.
4. Cuando al ejecutar un fichero ***.m** se produce un error y se obtiene el correspondiente mensaje en la **Command Window**, MATLAB muestra mediante un subrayado un enlace a la línea del fichero fuente en la que se ha producido el error. Clicando en ese enlace se va a la línea correspondiente del fichero por medio del **Editor/Debugger**.

3.4. Launch Pad.

El **Launch Pad** es un componente muy general que da acceso a otros componentes de MATLAB, sin tener que recurrir a los menús o a otros comandos. Entre ellos se pueden citar al **Help Browser**, a las **Demos**, al **Current Directory**, al **Workspace**, al **Path** y a GUIDE (**Graphic Interface Builder**).

3.4.1. Command History Browser.

El **Command History Browser** ofrece acceso a las sentencias que se han ejecutado anteriormente en la **Command Window**. Estas sentencias están también accesibles por medio de las teclas \uparrow y \downarrow como en las versiones anteriores, pero esta ventana facilita mucho el tener una visión más general de lo hecho anteriormente y seleccionar lo que realmente se desea repetir.

Las sentencias anteriores se pueden volver a ejecutar mediante un doble clic o por medio del menú contextual que se abre al clicar sobre ellas con el botón derecho. También se pueden copiar y volcar sobre la línea de comandos, pero se ha de copiar toda la línea, sin que se admita la copia de un fragmento de la sentencia. Existen opciones para borrar algunas o todas las líneas de esta ventana.

3.4.2. Current Directory Browser.

El concepto de **directorio activo** o **directorio actual** es muy importante en MATLAB. Los programas de MATLAB se encuentran en fichero con la extensión **.m**. Estos ficheros se ejecutan tecleando su nombre en la línea de comandos (sin la extensión). No todos los ficheros **.m** que se encuentren en el disco duro o en otras unidades lógicas montadas en una red local son accesibles.

Para que un fichero **.m* se pueda ejecutar es necesario que se cumpla una de las dos condiciones siguientes:

1. Que esté en el **directorio actual**. MATLAB mantiene en todo momento un único directorio con esta condición. Este directorio es el primer sitio en el que MATLAB busca cuando desde la línea de comandos se le pide que ejecute un fichero.
2. Que esté en uno de los directorios indicados en el **Path** de MATLAB. El **Path** es una lista ordenada de directorios en los que el programa busca los ficheros o las funciones que ha de ejecutar. Muchos de los directorios del **Path** son propios de MATLAB, pero los usuarios también pueden añadir sus propios directorios, normalmente al principio o al final de la lista. En un próximo apartado se verá cómo se controla el **Path**.

El comando ***pwd*** (de *print working directory*) permite saber cuál es el **directorio actual**. Para cambiar de **directorio actual** se puede utilizar el comando ***cd*** (de *change directory*) en la línea de comandos, seguido del nombre del directorio, para el cual se puede utilizar un **path** absoluto (por ejemplo ***cd C:\Matlab\Ejemplos***) o relativo (***cd Ejemplos***). Para subir un nivel en la jerarquía de directorios se utiliza el comando ***cd ..***, y ***cd ../../*** para subir dos niveles. Éste es el mismo sistema que se sigue para cambiar de directorio en las ventanas de MS-DOS. MATLAB permite utilizar tanto la barra normal (/) como la barra invertida (\), indistintamente. El comando ***cd*** era el único sistema de cambio de **directorio actual**.

El **Current Directory Browser** permite explorar los directorios del ordenador en forma análoga a la del **Explorador** u otras aplicaciones de **Windows**. Cuando se llega al directorio deseado se muestran los ficheros y ficheros allí contenidos. El **Current Directory Browser** permite ordenarlos por fecha, tamaño, nombre, etc. El directorio actual cambia automáticamente en función del directorio seleccionado con este browser, y también se puede

cambiar desde la propia barra de herramientas del **Matlab Desktop**. Los ficheros ***.m** mostrados **Current Directory Browser** se pueden abrir con el **Editor/Debugger** mediante un doble clic.

A partir del menú contextual que se abre desde el **Current Directory Browser** se tiene la posibilidad de añadir ese directorio al **Path** del MATLAB.

3.4.3. Workspace Browser y Array Editor.

El espacio de trabajo de MATLAB (**Workspace**) es el conjunto de variables y de funciones de usuario que en un determinado momento están definidas en la memoria del programa. Para obtener información sobre el **Workspace** desde la línea de comandos se pueden utilizar los comandos **who** y **whos**. El segundo proporciona una información más detallada que el primero. Por ejemplo, una salida típica del comando **whos** es la siguiente:

```
>>whos
Name           Size           Bytes Class
A              3x3            72 double array
B              3x3            72 double array
C              3x3            72 double array
D              3x3            72 double array
```

Grand total is 36 elements using 288 bytes

Éstas son las variables del **espacio de trabajo base** (el de la línea de comandos de MATLAB). Más adelante se verá que cada función tiene su propio espacio de trabajo, con variables cuyos nombres no interfieren con las variables de los otros espacios de trabajo.

La ventana **Workspace Browser** constituye un entorno gráfico para ver las variables definidas en el espacio de trabajo. Se activa con el comando **View/Workspace**. La Figura 3.2 muestra el aspecto inicial del **Workspace Browser** cuando se abre desde un determinado programa. Haciendo doble clic por ejemplo sobre la matriz **BARS** aparece una nueva ventana (o pestaña, si la ventana ya existía) del **Array Editor**, en la que se muestran y pueden ser modificados los elementos de dicha matriz.

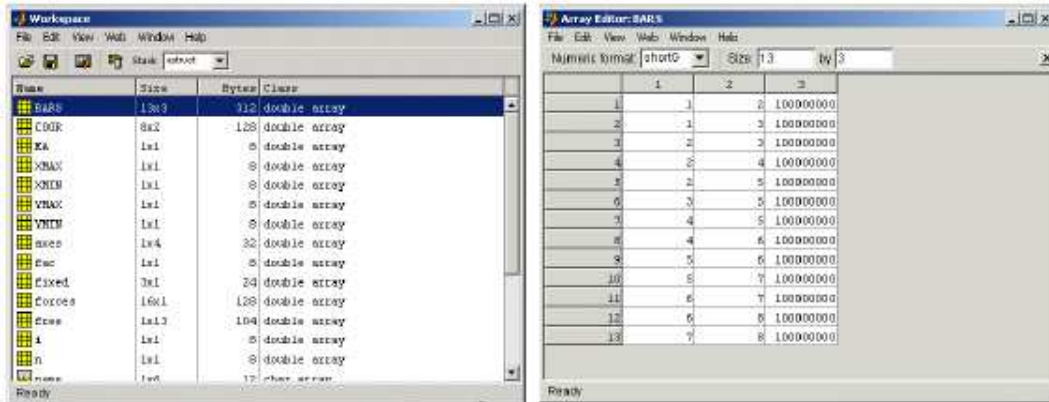


Figura 3. 2: Workspace Browser con elementos definidos y Array Editor (Editor de Matrices).

Fuente: Javier García de Jalón

Es importante insistir en que cada una de las funciones de MATLAB tiene su propio espacio de trabajo, al que en principio sólo pertenecen las variables recibidas como argumentos o definidas dentro de la propia función. En la barra de herramientas del **Workspace Browser** aparece una lista desplegable (**Stack**) con los espacios de trabajo del programa actual. Hay que tener en cuenta que cuando se termina de ejecutar una función y se devuelve el control al programa que la había llamado, las variables definidas en la función dejan de existir (salvo que se hayan declarado como **persistent**) y también deja de existir su espacio de trabajo.

Si se desean examinar otras matrices y/o vectores, al hacer doble clic sobre ellas el **Array Editor** las muestra en la misma ventana como pestañas diferentes. Clicando con el botón derecho sobre alguna de las variables del **Workspace Browser** se abre un menú contextual que ofrece algunas posibilidades interesantes, como por ejemplo la de **representar gráficamente** dicha variable.

El **Array Editor** no sólo permite ver los valores de los elementos de cualquier matriz o vector definido en el programa: es también posible modificar estos valores clicando sobre la celda correspondiente. La ventana del **Array**

Editor incluye una lista desplegable en la que se puede elegir el formato en el que se desea ver los datos.

El **Array Editor** es muy útil también para entender bien ciertos algoritmos, ejecutando paso a paso un programa y viendo cómo cambian los valores de las distintas variables. Es posible aparcar o situar las ventanas o pestañas del **Array Editor** en la misma ventana del **Editor/Debugger**, que se va a ver a continuación.

3.5. El Editor/Debugger.

En MATLAB tienen particular importancia los ya citados **ficheros-M** (o **M-files**). Son ficheros de texto ASCII, con la extensión **.m**, que contienen **conjuntos de comandos** o **definición defunciones** (estos últimos son un poco más complicados y se verán más adelante). La importancia de estos **ficheros-M** es que al teclear su nombre en la línea de comandos y pulsar **Intro**, se ejecutan uno tras otro todos los comandos contenidos en dicho fichero. El poder guardar instrucciones y grandes matrices en un fichero permite ahorrar mucho trabajo de teclado.

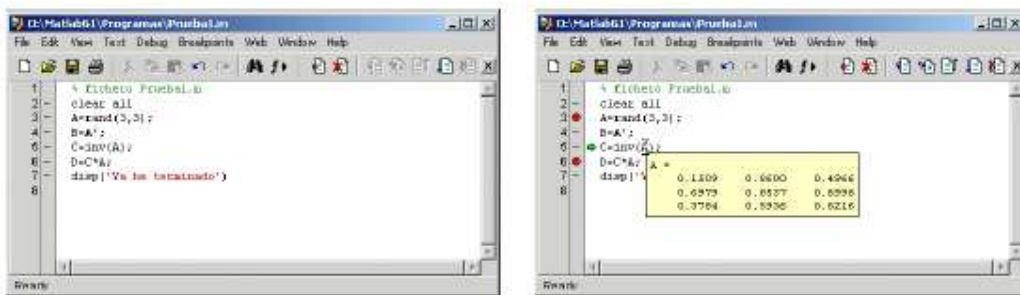


Figura 3. 3: Ventana del Editor/Debugger y Ejecución interactiva con el Editor/Debugger

Fuente: Javier García de Jalón

Aunque los ficheros **.m** se pueden crear con cualquier editor de ficheros ASCII tal como **Notepad**, MATLAB dispone de un **editor** que permite tanto crear y modificar estos ficheros, como ejecutarlos paso a paso para ver si contienen errores (proceso de **Debugo** depuración). La Figura 3.3 muestra la

ventana principal del **Editor/Debugger**, en la que se ha tecleado un **fichero-M** llamado **Prueba1.m**, que contiene un comentario y seis sentencias⁴. El **Editor** muestra con diferentes colores los diferentes tipos o elementos constitutivos de los comandos (en *verde* los comentarios, en *rojo* las cadenas de caracteres, etc.). El **Editor** se preocupa también de que las comillas o paréntesis que se abren, no se queden sin el correspondiente elemento de cierre. Colocando el cursor antes o después de una apertura o cierre de corchete o paréntesis y pulsando las teclas (⌈) o (⌋), el **Editor** muestra con qué cierre o apertura de corchete o paréntesis se empareja el elemento considerado; si no se empareja con ninguno, aparece con una rayita de tachado.

Seleccionando varias líneas y clicando con el botón derecho aparece un menú contextual que permite entre otras cosas **comentar con el carácter %** todas las líneas seleccionadas.

La Figura 3.4 corresponde a una ejecución de este fichero de comandos controlada con el **Debugger**. Dicha ejecución se comienza eligiendo el comando **Run** en el menú **Debug**, pulsando la tecla **F5**, clicando en el botón **Continue** () de la barra de herramientas del **Editor** o tecleando el nombre del fichero en la línea de comandos de la **Command Window**. Los puntos rojos que aparecen en el margen izquierdo son **breakpoints**(puntos en los que se detiene la ejecución de programa); la flecha verde indica la sentencia en que está detenida la ejecución (antes de ejecutar dicha sentencia); cuando el cursor se coloca sobre una variable (en este caso sobre la matriz **A**) aparece una pequeña ventana con los valores numéricos de esa variable.

En la Figura 3.4 puede apreciarse también que están activados los botones que corresponden al **Debugger**. El significado de estos botones, que aparece al colocar el cursor sobre cada uno de ellos, es el siguiente:

- a. **Set/Clear Breakpoint**. Coloca o borra un **breakpoint** en la línea en que está el cursor.

- b. Clear AllBreakpoints.** Elimina todos los **breakpoints** que haya en el fichero.
- c. Step.** Avanzar un paso sin entrar en las funciones de usuario que se llamen en esa línea.
- d. Step In.** Avanzar un paso, y si en ese paso hay una llamada a una función cuyo fichero
- e. *.m** está accesible, entra en dicha función.
- f. StepOut.** Salir de la función que se está ejecutando en ese momento.
- g. Continue.** Continuar la ejecución hasta el siguiente **breakpoint**.
- h. QuitDebugging.** Terminar la ejecución del **Debugger**.
- i. Stack.** En la parte derecha de la barra de herramientas aparece esta lista desplegable (no visible en la Figura 3.4) mediante la cual se puede elegir el *contexto*, es decir el **espacio de trabajo** o el ámbito de las variables que se quieren examinar. Ya se ha comentado que el espacio de trabajo base (el de las variables creadas desde la línea de comandos) y el espacio de trabajo de cada una de las funciones son diferentes.

El **Debugger** es un programa que hay que conocer muy bien, pues es muy útil para detectar y corregir errores. Es también enormemente útil para aprender métodos numéricos y técnicos de programación. Para aprender a manejar el **Debugger** lo mejor es practicar.

Cuando se está ejecutando un programa con el **Debugger**, en cualquier momento se puede ir a la línea de comandos de MATLAB y teclear una expresión para ver su resultado. También se puede seleccionar con el ratón una sub-expresión en cualquier línea vista en el **Editor/Debugger**, clicar con el botón derecho y en el menú contextual que se abre elegir **EvaluateSelection**. El resultado de evaluar esa sub-expresión aparece en la línea de comandos de MATLAB.

Ya en las versiones anteriores MATLAB disponía de un **Debugger alfanumérico** que se utilizaba desde la línea de comandos y en el que está basado el nuevo **Debugger gráfico** del que se ha hablado anteriormente. De hecho, al realizar operaciones con el **Debugger gráfico** van apareciendo las correspondientes instrucciones en la línea de comandos de MATLAB. Para más información sobre los comandos del **Debugger alfanumérico**, buscar en la sección “*Editing and Debugging M-Files*” en **Help/Matlab/UsingMatlab/DevelopmentEnvironment**.

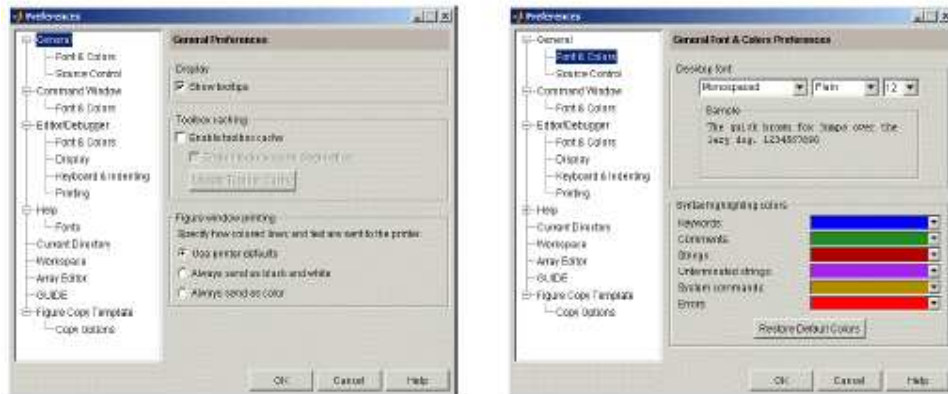


Figura 3. 4: Comando *Preferences* del menú *File*.
Fuente: Javier García de Jalón

3.6. Espacio de trabajo

Los datos y variables usadas residen en el espacio de trabajo, *workspace*, accesible desde la ventana de comandos. Para inspeccionar el contenido de este espacio se utilizan los comandos *who* y *whos*. Los archivos directamente accesibles desde el espacio de trabajo, se muestran mediante *what*. En el siguiente ejemplo se muestran sus características:

```
>> t = linspace(1,10,4) % Crea un vector de 4 elementos desde
                        % 1 a 10.
t =
    1     4     7    10
```

```

>> t = t(:)           % Crear el vector t en columna.
t =
     1
     4
     7
    10

>> A = 2*t; B = 2;
>> who

Your variables are:

t  A  B

>> whos

  Name      Size      Bytes  Class
  ----      -
  t         1 x 4         32  double array
  A         1 x 4         32  double array
  B         1 x 1          8  double array

Grand total is 9 elements using 72 bytes

>> what

M-files in the current directory D:\MatLab\work
AjusNL      Datos      E cudifP
AjusNLFun   E cudif      Fun
MAT-files in the current directory D:\MatLab\work
Datos
MDL-files in the current directory D:\MatLab\work
Bcont      Bfuntab      BnoLineal    Bseffsis
Bfuentes   Bmat         Bsalidas     Ecu2

```

En el espacio de trabajo se crearon 3 variables, 2 de 4 elementos, y una de 1 elemento, de modo que son 9 elementos a 8 bytes por elemento, lo que hace un total de 72 bytes. A partir de la versión 6, release 12, MatLab incorporó workspace, potenciando la capacidad de whos.

3.7. Variables.

En MatLab no es necesario hacer declaraciones previas acerca de las variables. El contenido de las variables de caracteres ha de ir delimitado por el signo «'».

```

>> numero_de_visitantes = 25

numero_de_visitantes =

    25

```

De ese modo se crean variables numéricas, `numero_de_visitantes`, que almacenan su valores en una matriz, en este caso la matriz es de 1 1, y su valor es 25.

```
» size(numero_de_visitantes) % Dimensión de variables.  
ans =  
    1    1  
» Nombre='Pepe';           % Variable de caracteres.  
» size (Nombre)  
ans =  
    1    4
```

Los nombres de las variables deben seguir estas reglas: 1. Se forman con las letras del abecedario, los dígitos 0 a 9 y el signo «_», distinguiéndose mayúsculas de minúsculas. Los nombres de las variables han de comenzar por una letra y no deben contener espacios en blanco. Los nombres de las variables no pueden coincidir con los nombres de las keywords, nombres reservados. 2. 3. La lista de los nombres reservados se obtiene por medio de `iskeyword`:

```
» iskeyword  
ans =  
'break'  
'case'  
'catch'  
'continue'  
'else'  
'elseif'  
'end'  
'for'  
'function'  
'global'  
'if'  
'otherwise'  
'persistent'  
'return'  
'swirch'  
'try'  
'while'
```

Los nombres de las variables pueden ser tan extensos como se quiera, pero MatLab sólo reconoce los 31 primeros caracteres. Las variables se eliminan del espacio de trabajo con el comando clear: clearclear variables clear global clearfunctionsclearallclear pipo* Elimina las variables del espacio de trabajo. Es equivalente al comando anterior. Elimina las variables globales. Elimina todas las funciones compiladas. Elimina todas las variables, globales y funciones. Elimina las variables que empiezan por pipo. MatLab suministra amplia información adicional mediante helpclear.

3.8. Formato de números.

MatLab presenta los resultados numéricos en varios formatos, según se expresa a continuación:

```
>> help format
```

```
FORMAT Set output format.
All computations in MATLAB are done in double precision.
FORMAT may be used to switch between different output
display formats as follows:
  FORMAT          Default. Same as SHORT.
  FORMAT SHORT    Scaled fixed point format with 5 digits.
  FORMAT LONG     Scaled fixed point format with 15 digits.
  FORMAT SHORT E  Floating point format with 5 digits.
  FORMAT LONG E   Floating point format with 15 digits.
  FORMAT SHORT G  Best of fixed or floating point for-
                  mat with 5 digits.
  FORMAT LONG G   Best of fixed or floating point format
                  with 15 digits.
  FORMAT HEX      Hexadecimal format.
  FORMAT +        The symbols +, - and blank are printed
                  for
                  positive, negative and zero elements.
                  Imaginary parts are ignored.
  FORMAT BANK     Fixed format for dollars and cents.
  FORMAT RAT      Approximation by ratio of small integers.

Spacing:
  FORMAT COMPACT  Suppress extra line-feeds.
  FORMAT LOOSE    Puts the extra line-feeds back in.
```

Al mostrar resultados numéricos, MatLab sigue estas dos reglas:

1. MatLab intenta mostrar números enteros. Si el entero es muy grande, se presenta en formato exponencial, con 5 cifras significativas. Los números con decimales se muestran con 4 o 5 cifras significativas.
2. Los números en valor absoluto menores de 0,01 y mayores de 1.000, se muestran en formato exponencial.

A continuación se muestran ejemplos demostrativos de formatos numéricos:

```
>> sqrt(2)

ans =

    1.4142

>> format long
>> sqrt(2)

ans =

    1.41421356237310

>> format Long e
>> sqrt (2)

ans =

    1.41421356237310e+00

>> format short
>> A= [10000 0.0001]

ans =

    1.0e+04 *
    1.0000    0.0000

>> format short g
>> A

A =

    10000    0.0001
>> format rat
>> A

A =

    10000 1/10000
```

3.9. Programas

MatLab acepta comandos directos, para inmediatamente producir el resultado o ejecutar una serie de comandos almacenados en un archivo, con la extensión «.m». Un archivo.m, consiste en una secuencia de sentencias MatLab, posiblemente incluyendo referencias a otros archivo.m, o recursivamente a sí mismo. A estos archivos los llamamos programas MatLab, en inglés scripts. Las variables de los programas se mantienen en el espacio de trabajo, pudiendo ser invocadas en cualquier momento para ver su contenido. En una sentencia, lo que sigue a % no se ejecuta, se considera un comentario. Si se desea construir una tabla con inversos, cuadrados y raíces

cuadradas de 1 a 10, se edita un archivo, Numeros.m, con cualquier editor, tal como el bloc de notas del sistema operativo, o con el editor propio de MatLab, según:

```
% -----Numeros.m-----
x=1:10; % Crea un vector de 1 a 10 de 1 en 1. Vector en línea.
x=x'; % Transposición. Vector en columna.
x=[x,1./x,x.^2,sqrt(x)]; % Matriz de 4 columnas.
% -----
```

El programa se invoca ejecutando «Numeros». Como en el programa todas las sentencias se finalizaron con «;», no se muestra ningún valor numérico. Al ejecutar «x», se obtendrá la tabla deseada:

```
>> Numeros
>> x

x =

    1.0000    1.0000    1.0000    1.0000
    2.0000    0.5000    4.0000    1.4142
    3.0000    0.3333    9.0000    1.7321
    4.0000    0.2500   16.0000    2.0000
    5.0000    0.2000   25.0000    2.2361
    6.0000    0.1667   36.0000    2.4495
    7.0000    0.1429   49.0000    2.6458
    8.0000    0.1250   64.0000    2.8284
    9.0000    0.1111   81.0000    3.0000
   10.0000    0.1000  100.0000    3.1623
```

Pulsando la tecla, ↑ se consigue las líneas de los comandos previamente ejecutados.

3.10. Funciones

El otro tipo de archivos utilizado por MatLab son las funciones, cuya primera característica es que sus variables son locales en su entorno y no definidas en el espacio de trabajo, ni en otras funciones. Buena parte de la potencia de MatLab se basa en su extenso conjunto de funciones, las básicas y las distribuidas de forma separada para aplicaciones específicas, MatLabtoolboxes, y otras que desarrollan los usuarios. Las funciones toman unas variables de entrada para calcular unos datos de salida, sea:

$$Fun(x) = \frac{1}{(x-1)^2 + 0,1} + \frac{1}{(x-3)^2 + 0,2} - 5$$

Almacenado en el archivo «Fun.m», cuyo contenido es:

```
function y=Fun(x)
%-----Fun.m -----
%      Esto es un ejemplo
%      de una función.
y=1./((x-1).^2+0.1)+1./((x-3).^2+0.2)-5;
%-----
```

Para evaluar Fun gráficamente, se lanza con las siguientes instrucciones.

```
>> x=-2:0.01:6; % Vector de -2 a 6, a incrementos de 0,01.
>> y=Fun(x);    % Guardando el vector Fun(x) en y.
>> plot(x,y),grid % Representación con rejilla.
```

El resultado que se obtiene se muestra en la gráfica de la figura 3.5. En Fun(x), x es el argumento o entrada de Fun, para dar unos resultados de salida que se almacenan en y, que se muestran gráficamente. Hay funciones del sistema o construidas por un usuario, que toman diferente número de argumentos de entrada que de salida. Así la función max, toma un vector de argumentos y puede suministrar una o dos salidas, según se use:

```
>> A = [1 2 1 5 2 3];
>> max(A) % Suministrará el valor máximo de A.
ans =
     5

>> [X, i] = max(A) % X, valor máximo, i posición del máximo.
X =
     5
i =
     4
```

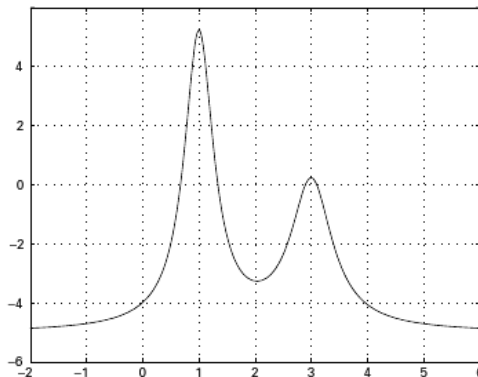


Figura 3. 5: Representación gráfica de Fun.
Fuente: Gil Rodríguez, M

3.10.1. Reglas de construcción de funciones

1. El nombre de la función y del archivo deben ser idénticos.
2. Los nombres de las funciones se rigen por las normas de los nombres de las variables.
3. La primera línea ejecutable de una función debe ser la línea de declaración de función.
4. Las variables del interior de las funciones son variables locales.
5. El conjunto de líneas consecutivas de comentarios que siguen a function, componen el texto de ayuda de esa función, obtenible mediante help y lookfor.
6. Una función termina al encontrar un return o al llegar a la última línea de la función.
7. Si una función llama a un programa, éste es evaluado en el espacio de trabajo de la función y no en el workspace de MatLab.
8. Cada función tiene su espacio de trabajo separado del de MatLab, de modo que la conexión entre estos ambientes se realiza a través de las variables de entrada y salida de la función.
9. Para compartir variables del interior de las funciones y del espacio de trabajo, se declaran variables globales donde se necesiten mediante la instrucción global.

```
>>global X a b      % Declaración de variables globales
```

Para facilitar el manejo de funciones, MatLab incorporó recientemente @, y feval, para mejorar eval, cuya utilidad se expone en el siguiente ejemplo:

```

F =
    @Fun
>> feval(F,2)

ans =
    -3.2576

Se consigue el mismo resultado con:

>> eval('Fun(2)')

ans =
    -3.2576
>> Fun(2)

ans =
    -3.2576
>> F(2)=@cos    % Creación directa de F(2).

F =
    @Fun    @cos

```

La eficiencia de feval es considerablemente superior a eval, ya que el primero evalúa directamente lo que se invoca, mientras que eval llama al interpretador completo de MatLab. La diferencia en tiempo de ejecución de ambas funciones se pone de manifiesto con:

```

>> tic, for i = 1:100000, a = eval('Fun(i)'); end, toc
elapsed_time =
    14.3210
>> tic, for i = 1:100000, a = feval('Fun',i); end, toc
elapsed_time =
    4.0960

```

3.10.2. Funciones en línea

Un segundo modo de definir funciones, sin editar archivos, se logra con inline:

```

>> syms x y
>> f = inline('x.^2 + y.^2')

f =

      Inline function:
      f(x) = x.^2 + y.^2
>> f(3,4)

ans =

      25
>> feval(f,3,4)

ans =

      25

```

3.11. Números Complejos.

MatLab admite operaciones con números complejos, permitiendo usar indistintamente la i y la j , según se muestra en lo siguiente:

```

>> a = sqrt(-1)

a =

      0 + 1.0000i
>> conj(a)

ans =

      0 - 1.0000i
>> sqrt(a)

ans =

      0.7071 + 0.7071i
>> exp(2i)

ans =

      -0.4161 + 0.9093i
>> A = (3 + 4i)*(2 - j)

A =

      10.0000 + 5.0000i
>> r = real(A)

r =

      10
>> I = imag(A)

I =

      5
>> r = abs(A)

r =

      11.1803
>> Angulo = angle(A)

```

3.12. Manejo de vectores y matrices

La forma más sencilla de crear un vector es mediante el uso de [], vector en línea, o con []', vector en columna. Los elementos se separan por espacios o comas, el «;» se reserva para anexar en columna:

```
>> t = [3 5 7, 8, 9]
t =
    3  5  7  8  9
```

También se generan vectores mediante las instrucciones linspace y logspace, ambos con dos o tres argumentos, y con «:», ya mencionado:

```
>> x = logspace(0,2,5) % Vector de 5 componentes de 10^0 a
10^2.
x =
    1.000    3.1623   10.0000   31.6228   100.000
```

Con el siguiente ejemplo se muestra la creación y manejo de matrices:

```
>> x = 0:4
x =
    0  1  2  3  4
>> y = x.^2 % El punto antes del exponente
y = % hace que la exponenciación
    0  1  4  9  16 % sea elemento a elemento.
>> a = [x;y] % Crear una matriz anexando
% vectores.
a =
    0  1  2  3  4
    0  1  4  9  16
>> A = a' % Crear la matriz A, transpues-
% ta de a.
A =
    0  0
    1  1
    2  4
    3  9
    4  16
```

3.13. Tipos de Gráficos.

Desde las primeras versiones, MatLab traía suficientes utilidades gráficas, que en las versiones posteriores fueron incorporando cantidades ingentes de nuevas facilidades. Se reseñan los principales tipos de construcción de gráficos:

- `fplot`. Para representación de funciones: `fplot('Fun',[-pi pi])`.
- `plot`. Representación de x frente a y: `plot(x,y)`.
- `plotyy`. Representación en los ejes opuestos de ordenadas: `plotyy(x1,y1,x2,y2)`.
- `plotmatrix`. Matriz de representaciones: `plotmatrix(x,y)`.
- `bar`. Representación con barras: `bar(x,y,ancho,tipo)`.
- `stairs`. Representación en escalones: `stairs(x,y)`.
- `errorbar`. Representación acompañada de un parámetro de desviación: `errorbar(x,y,e)`.
- `stem`. Representación discreta: `stem(x,y)`.
- `pie`. Representación en tarta: `pie(x)`.
- `plot3`. Representación en 3-d: `plot3(x,y,z)`.
- `semilogy`. Representación semilogarítmica en el eje y: `semilogy(x,y)`.
- `semilogx`. Representación en el eje x: `semilogx(x,y)`.
- `loglog`. Representación logarítmica en los dos ejes: `loglog(x,y)`.

En la siguiente figura 3.6, se representan los principales tipos de gráficos mencionados.

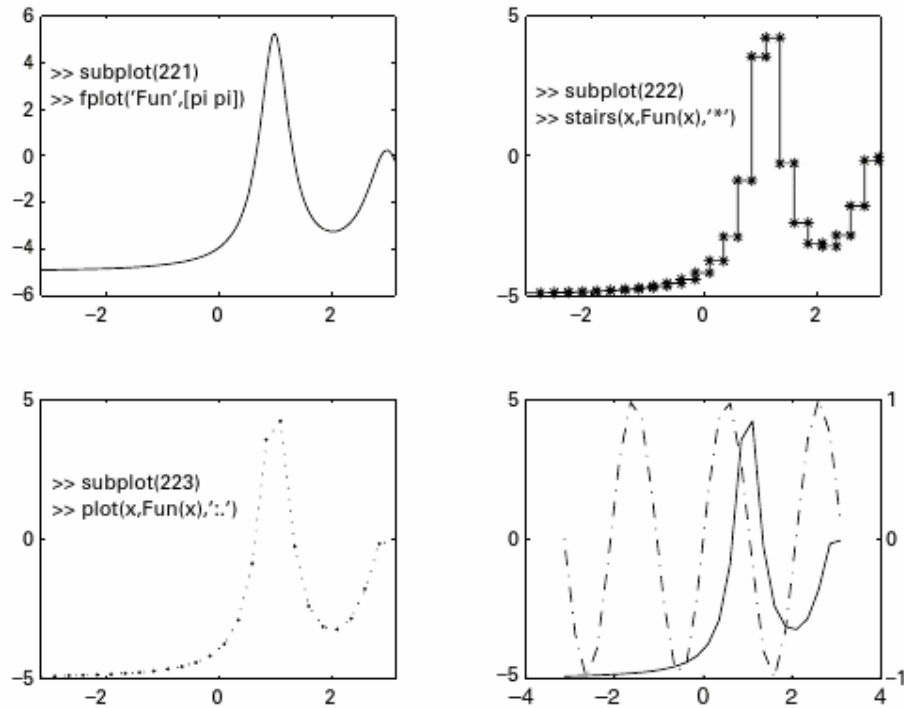


Figura 3. 6: Diferentes tipos de gráficos.
Fuente: Gil Rodríguez, M.

3.14. Entornografico MatLab (GUIDE).

GUIDE es un entorno de programación visual disponible en MATLAB para realizar y ejecutar programas que necesiten ingreso continuo de datos. Tiene las características básicas de todos los programas visuales como Visual Basic o Visual C++.

3.14.1. Inicio

Para iniciar nuestro proyecto, debemos dar clic en GUIDE, tal como se muestra en la figura 3.7.

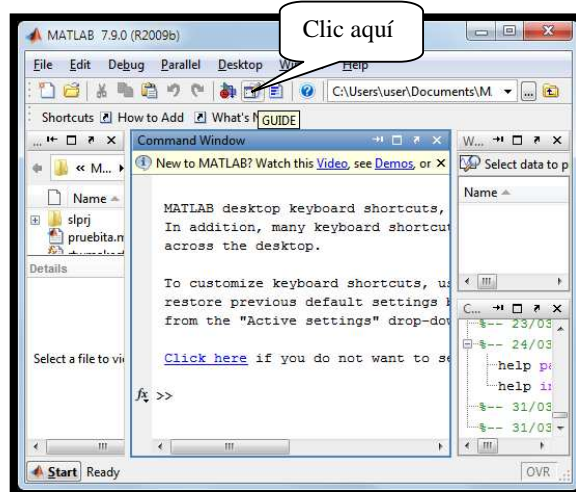


Figura 3. 7: Ventana de MatLab para iniciar programación gráfica GUIDE.
Fuente: El Autor.

Posteriormente, se abre la siguiente ventana o cuadro de dialogo, tal como se muestra en la figura 3.8.

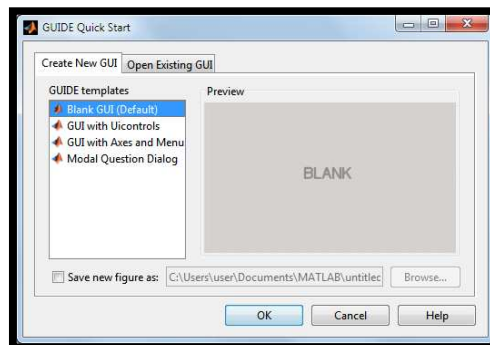


Figura 3. 8: Ventana de GUIDE Quick Start.
Fuente: El Autor.

Escogemos la opción *Blank GUI (Default)*(véase la figura 3.8) también denominada “opción de interfaz gráfica de usuario en blanco” (la misma viene predeterminada), nos presenta un formulario nuevo, en el cual podemos diseñar nuestro programa. En la figura 3.9 se muestra la opción escogida del entorno de programación gráfica:

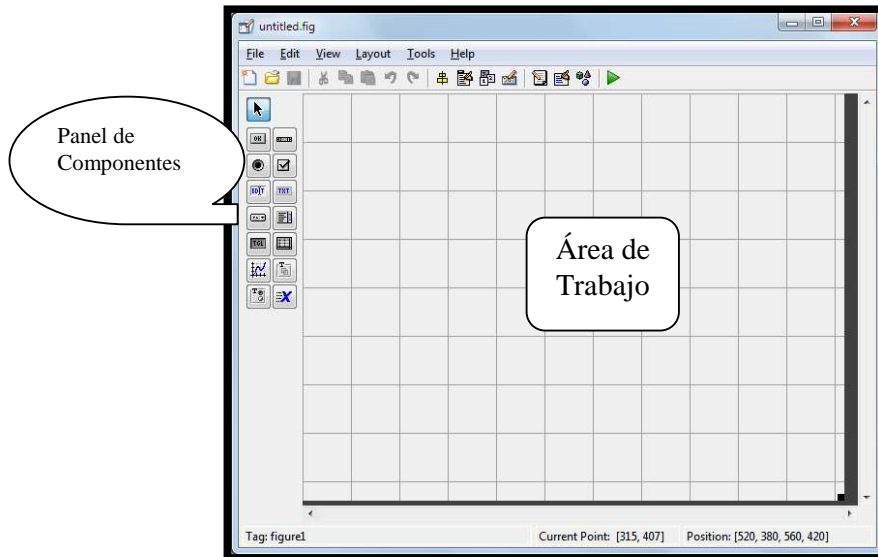










Figura 3. 9: Ventana del Entorno Gráfico (GUIDE) de MatLab.
Fuente: El Autor.

La figura 3.10 muestra la barra de herramientas del GUIDE.



Figura 3. 10: Barra de Herramientas del Entorno Gráfico (GUIDE) de MatLab.
Fuente: El Autor.

A continuación se describen cada uno de los iconos de la barra de herramientas del entorno gráfico:

-  Alinear objetos.
-  Editor de menú.
-  Editor de orden de etiqueta.
-  Navegador de objetos.
-  Editor del M-file.
-  Editor de Barra de Herramientas
-  Accede al panel de componentes
-  Grabar y Ejecutar

Para obtener la etiqueta de cada elemento de la paleta de componentes ejecutamos: `File>>Preferentes` y seleccionamos `Show names in componentpalette`, en la figura 3.11 se muestra la ventana de componentes.

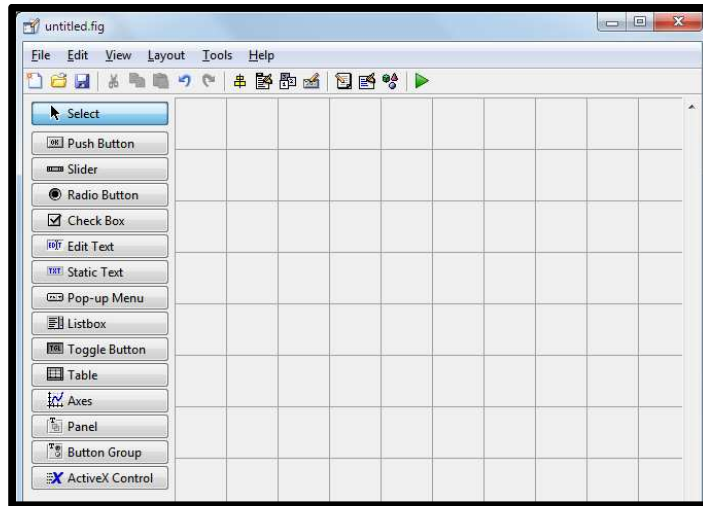


Figura 3. 11: Paleta de Componentes del GUI de MatLab.
Fuente: El Autor.

3.14.2. Propiedad de los componentes

Cada uno de los elementos de GUI, tiene un conjunto de opciones que podemos acceder con clic derecho, tal como se ilustra en la figura 3.12.

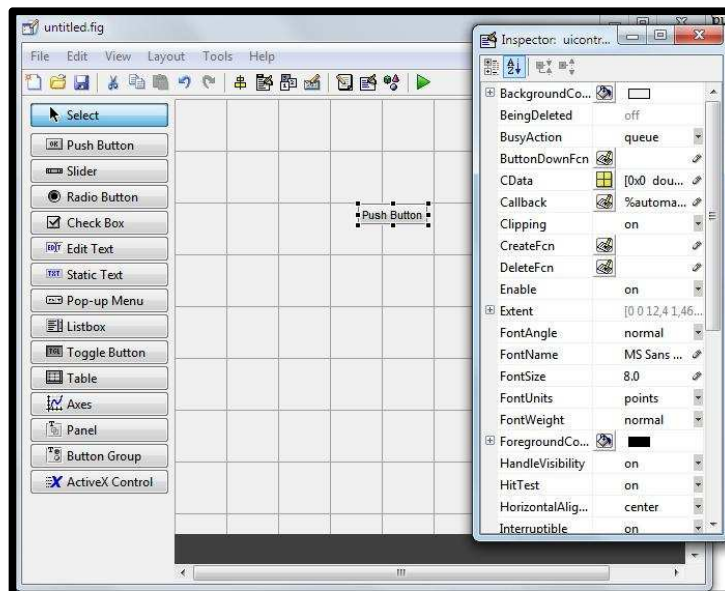


Figura 3. 12: Propiedades de paleta de Componentes del GUI de MatLab.

Fuente: El Autor.

Al hacer clic derecho en el elemento ubicado en el área de diseño, una de las opciones más importantes es View Callbacks, la cual, al ejecutarla, abre el archivo .m asociado a nuestro diseño y nos posiciona en la parte del programa que corresponde a la subrutina que se ejecutará cuando se realice una determinada acción sobre el elemento que estamos editando.

Por ejemplo, al ejecutar View Callbacks>>Callbacks en el Push Button, nos ubicaremos en la parte del programa:

```
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton1 (see GCBO)
% eventdata reserved-to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

3.14.3. Funcionamiento de una aplicación GUI

Una aplicación GUIDE consta de dos archivos: .m y .fig. El archivo .m es el que contiene el código con las correspondencias de los botones de control de la interfaz y el archivo .fig contiene los elementos gráficos.

Cada vez que se adicione un nuevo elemento en la interfaz gráfica, se genera automáticamente código en el archivo .m.

Para ejecutar una Interfaz Gráfica, si la hemos etiquetado con el nombre curso.fig, simplemente ejecutamos en la ventana de comandos >> curso. También podemos acceder haciendo clic derecho en el m-file y seleccionando la opción RUN.

3.14.4. Manejo de datos entre los elementos de la aplicación y el archivo .m

Todos los valores de las propiedades de los elementos (color, valor, posición, string...) y los valores de las variables transitorias del programa se almacenan en una estructura, los cuales son cedidos mediante un único y mismo *identificador* para todos éstos. Tomando el programa listado anteriormente, el identificador se asigna en:

```
handles.output = hObject;
```

handles, es nuestro identificador a los datos de la aplicación. Esta definición de identificador es salvada con la siguiente instrucción:

```
guidata(hObject, handles);
```

guidata, es la sentencia para salvar los datos de la aplicación.

3.14.5. Sentencias Get y Set

La asignación u obtención de valores de los componentes se realiza mediante las sentencias *get* y *set*. Por ejemplo si queremos que la variable *utpl* tenga el valor del

Slider escribimos:

```
utpl= get(handles.slider1, 'Value');
```

CAPÍTULO 4: DESARROLLO EXPERIMENTAL

4.1. Hardware.

Para el desarrollo de proyectos del presente trabajo de titulación, es conveniente mostrar primeramente al lector el hardware y software necesario, y finalmente se mostrarán dos prácticas como aplicación de microcontroladores a través de la plataforma de programación MatLab.

4.1.1. Módulo de entrenamiento.

El módulo de entrenamiento MEI&T es programable, posee un microcontrolador PIC16F886. Este microcontrolador está embebido con una variedad de componentes electrónicos que le da un valor agregado, facilitándonos en gran medida el desarrollo de proyectos electrónicos.

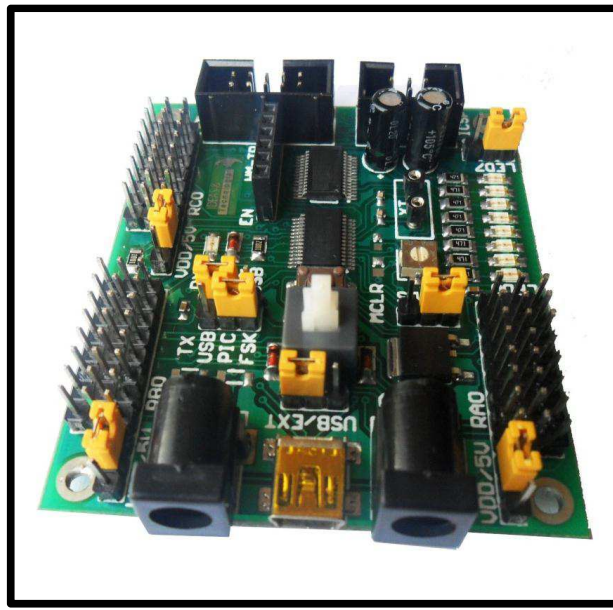


Figura 4. 1: Módulo de entrenamiento MEI&T.
Fuente: El Autor.

Características principales del módulo de entrenamiento:

- -UART, One Wire, SPI e I2C
- -Comunicación ONE WIRE y USART

- -RX y TX (FSK, ASK y QPSK)
- -Potenciómetro integrado
- -10 entradas analógicas
- -24 entradas y salidas digitales
- -8 leds indicadores de salidas digitales
- -Control para 2 motores DC (Dirección y Velocidad)
- -Programación ICSP in circuit
- -Reset manual
- -Switch de ON/OFF
- -Led indicador de power
- -Regulador integrado

El módulo de desarrollo le permite trabajar con señales analógicas y digitales del exterior que pueden provenir desde sensores. Además nos permite generar señales digitales que pueden ser usadas para controlar actuadores, tal como se muestra en la figura 4.2.

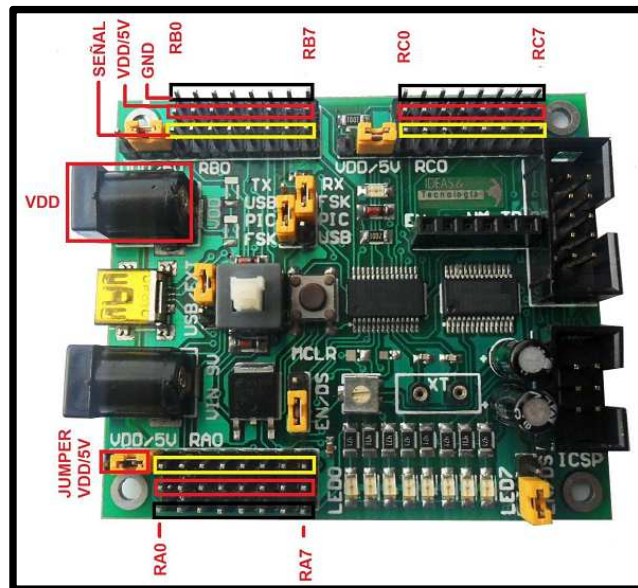


Figura 4. 2: Puertos de Entrada y Salida (E/S) del MEI&T.
Fuente: El Autor.

A continuación, podemos ver más características acerca de las salidas y entradas del módulo entrenamiento:

- Puertos 25/24 Pines IN/OUT
- Puertos: A, B, C, E
- 11 entradas analógicas con 10 bit de resolución
- 3 Timers (Timer0 8bits, Timer1, 2 16bits)
- 2 PWM (CCP) de 10bits, frecuencia máx 20KHZ
- Oscilador interno 31KHz-8MHz
 - AN0=RA0, AN1=RA1, AN2=RA2, AN3=RA3, AN4=RA5, AN8=RB2, AN9=RB3, AN10=RB1 AN11=RB4, AN12=RB0, AN13=RB5
 - DIR1: RA2, NDIR1: RA4
- DIR2: RA3, NDIR2: RA5
- PWM1: RC1 PWM2: RC2
- ADCISENA: RB1 ADCISENB: RB2

4.1.2. Programador P.PIC1&T04

El programador es muy importante ya que nos permite cargar o “quemar” el archivo “.HEX” en el microcontrolador.

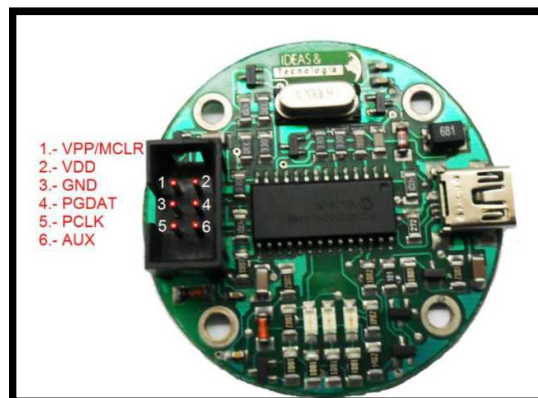


Figura 4. 3: Programador P.PIC1&T04.
Fuente: El Autor.

Características:

- Soporte de todas las familias de pics del fabricante microchip.
- Conexión Mini USB con el computador
- CD con software PICKIT 2

- Programador ICSP mediante bus de datos
- Led indicador de programación
- Led indicador de power
- Led indicador VDD
- VPP/MCLR: Señal para el pin MCLR del PIC
- VDD: Señal de voltaje para el pin VDD del PIC
- GND/VSS: Señal de referencia para el pin VSS del PIC
- PDAT: Señal de datos para el pin PDAT del PIC
- PCLK: Señal de reloj para el pin PCLK del PIC
- AUX: NC

4.1.3. Driver: P.H. I&T 04

Por medio de este driver podemos controlar la velocidad y giro de un motor DC

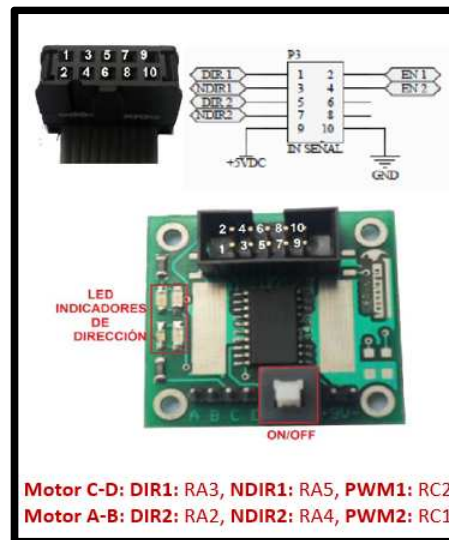


Figura 4. 4: Programador P.PIC I&T04.
Fuente: El Autor.

Características:

- ✓ P.H I&T 04 es un driver para el control de dirección y velocidad (PWM) de dos motores DC totalmente in dependientes.
- ✓ Alimentación: Motor (4.5-36)VDC , Control VCC (4.5-7)V

- ✓ Salidas: A,B-> MOTOR 1 ; C,D -> MOTOR 2
- ✓ Driver L293D
- ✓ Conexión a fuentes para motores.
- ✓ Conexión con M.E I&T 04 con bus datos IDC
- ✓ 600mA por canal, 1.2A Pico
- ✓ CCP1 (PWM1) -> RC2
- ✓ CCP2 (PWM2) -> RC1
- ✓ Usos: Control de motores para robots con tracción diferencial, control de motores para ventilación, motores para tacómetros, etc.

4.2. Programando el Hardware

Para realizar cualquier aplicación debemos programar al módulo y esa programación dependerá de la necesidad que tengamos, para esto requerimos de un programa el cual es Mikrobasic, una vez realizado esto se procede a cargar el programa desarrollado al módulo de entrenamiento. En la figura 4.5 se muestra los pasos para la programación de microcontroladores.



Figura 4. 5: Pasos para la programación.
Fuente: El Autor.

Para programar debemos tener en consideración los componentes y puertos del microcontrolador que vamos a usar, el cual es el PIC16F886, tal como se muestra en la figura 4.6.

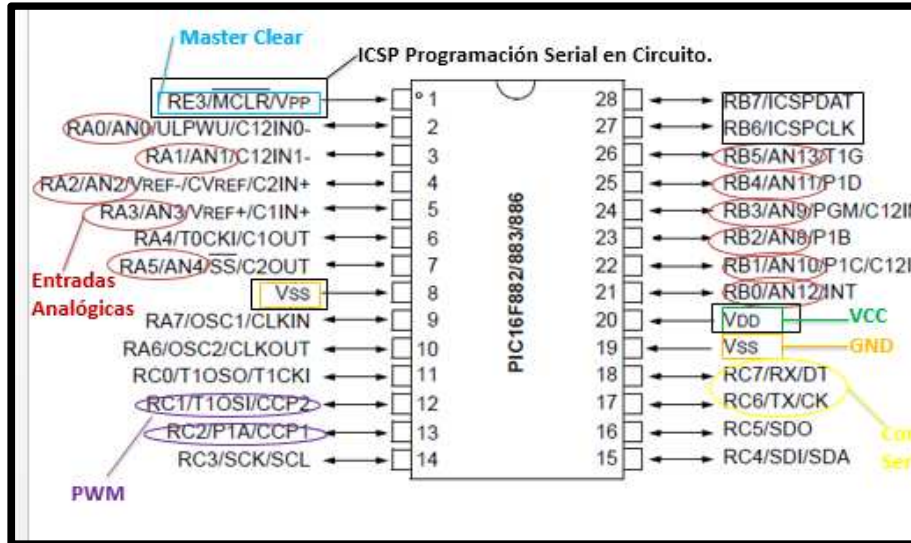


Figura 4. 6: Descripción de pines del PIC16F886.
Fuente: El Autor.

3.14.6. Uso de contador y Leds.

Para la presente experiencia práctica, programaremos al módulo para que encienda los leds de manera indefinida. En la figura 4.7 se observa el diagrama de bloques.

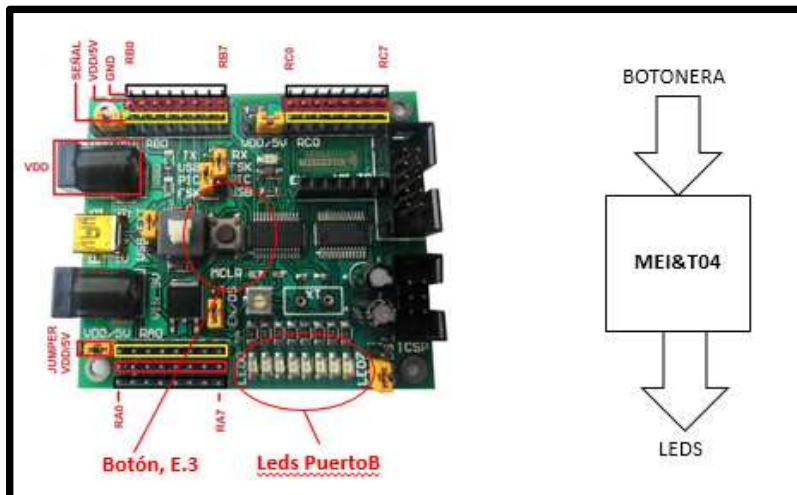


Figura 4. 7: Diagrama de bloques para encendido indefinido de LEDs.
Fuente: El Autor.

A continuación, la figura 4.8 muestra la programación del módulo para el encendido indefinido de los LEDs:

```

program ejercicio1

SYMBOL BOTON=PORTE.3
DIM VALOR AS BYTE

MAIN:
'-----CONFIG IN-OUT-----
' Registro PUERTO A
TRISA = 0X00 ' %00000000=0' PORTA -> out
' Registro PUERTO B
TRISB = 0X00 ' PORTB -> out para los leds
' Registro PUERTO C
TRISC = 0X00 ' PORTB -> out para los leds
' Registro PUERTO E
TRISE = %00001000 ' PORTE.3 <-in
'-----CONFIG IN -> DIG ó ANLG-----
' Seleccion de registro analogico. 1 analogico, 0
digitales
ANSEL = 0X00 ' AN<7:0> AN(____,4,3,2,1,0)
ANSELH = 0X00 ' AN<13:8> AN(____,13,12,11,10,9,8)

'----- SETEAR LAS SALIDAS-----
PORTB=%10101010
delay_ms(300)
PORTB=%01010101
delay_ms(300)
PORTB=0

'-----SET DE VARIABLES-----
VALOR=0

WHILE(1)
IF(BOTON =0) THEN 'SI PRESIONO EL BOTON
MCLR
PORTB= NOT PORTB
WHILE(BOTON=0)WEND 'ESPERO QUE SUELTE
EL BOTON MCLR
END IF

WEND
END.

```

Figura 4. 8: Programación en MikroBasic del encendido indefinido de LEDs.
Fuente: El Autor.

Posteriormente, se procede a cargar el programa una vez compilado y generado el archivo *.hex, en la figura 4.9 se muestra al programador y módulo previo a la carga del encendido de los LEDs.

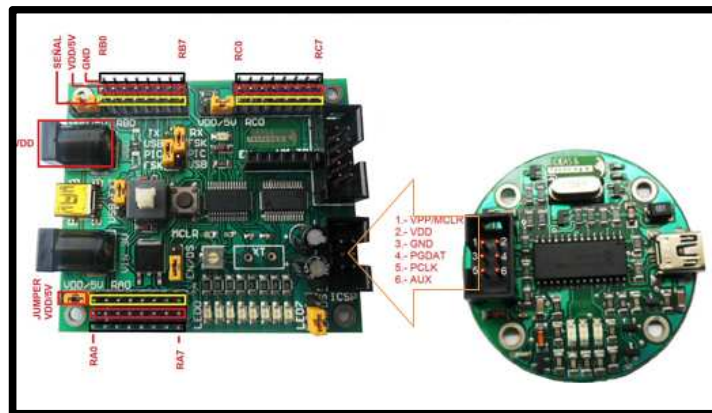


Figura 4. 9: Conexión entre el programador y módulo de entrenamiento.
Fuente: El Autor.

3.14.7. Uso de Botonera – Contadory Leds

En este ejercicio práctico programaremos el módulo para que encienda los leds una vez presionado la botonera, cada vez que presione la botonera incrementará un contador que se verá reflejado en los leds.

En la figura 4.10 se muestra el diagrama de para el uso de la botonera – contador y leds.

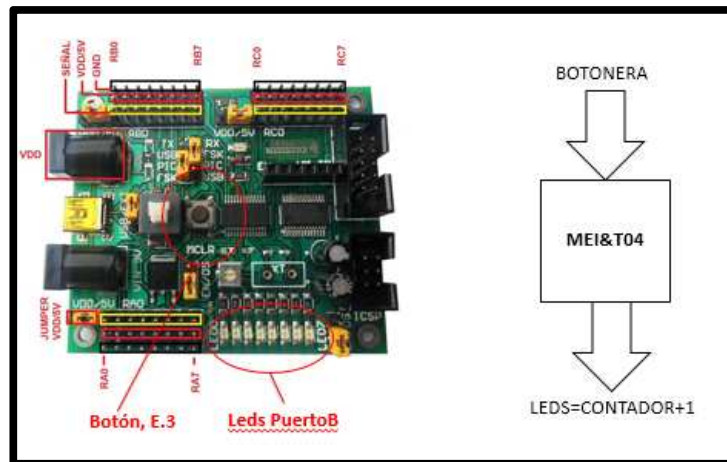


Figura 4. 10: Diagrama de bloques de botonera para encendido de LEDs.
Fuente: El Autor.

A continuación, la figura 4.11 muestra la programación del módulo para encender los leds a través de una botonera.

```

program ejercicio1

SYMBOL BOTON=PORTE.3
DIM VALOR AS BYTE

MAIN:
'-----CONFIG IN-OUT-----
' Registro PUERTO A
TRISA = 0X00 ' %00000000=0' PORTA -> out
' Registro PUERTO B
TRISB = 0X00 ' PORTB -> out para los leds
' Registro PUERTO C
TRISC = 0X00 ' PORTB -> out para los leds
' Registro PUERTO E
TRISE = %00001000 ' PORTE.3 <-in
'-----CONFIG IN -> DIG ó ANLG-----
' Seleccion de registro analogico. 1 analogico, 0
digitales
ANSEL = 0X00 ' AN<7:0> AN(____,4,3,2,1,0)
ANSELH = 0X00 ' AN<13:8> AN(____,13,12,11,10,9,8)

'----- SETEAR LAS SALIDAS-----
PORTB=%10101010
delay_ms(300)
PORTB=%01010101
delay_ms(300)
PORTB=0

'-----SET DE VARIABLES-----
VALOR=0

WHILE(1)
IF(BOTON =0) THEN 'SI PRESIONO EL BOTON
MCLR
PORTB= NOT PORTB
WHILE(BOTON=0)WEND 'ESPERO QUE SUELTE
EL BOTON MCLR
END IF

WEND
END.

```

Figura 4. 11: Programación en MikroBasic de la botonera para encender LEDs.
Fuente: El Autor.

Y en un proceso similar al de la figura 4.9, se debe proceder para programar el microcontrolador PIC16F886, y en forma general siempre se debe seguir el mismo procedimiento para realizar la programación.

4.3. Software: Programación gráfica en MatLab (GUIDE).

En el capítulo 3, se describió brevemente la programación gráfica GUIDE, en el presente acápite, presentamos dos ejemplos básicos de programación GUI. En la figura 4.12 se muestra el diagrama de bloques para programación gráfica u entorno virtual.

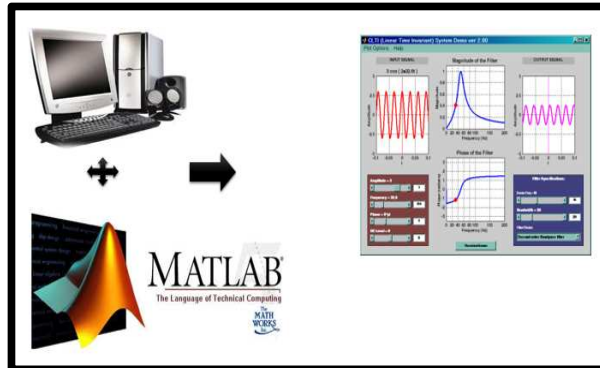


Figura 4. 12: Diagrama de bloques de entorno gráfico.
Fuente: El Autor.

4.3.1. Ejemplo: Programación GUI – Sumador de dos Números.

Por medio de este ejemplo aprenderemos a usar los elementos básicos de la herramienta GUIDE. Se requiere diseñar un programa que sume dos cantidades y presente el resultado al usuario final. Pasos para el desarrollo del programa:

1. Se escogen los elementos necesarios para el desarrollo del programa en este caso seleccionamos 4 “static text”, 2 “edit text” y un “push button”, tal como se muestra en la figura 4.13.

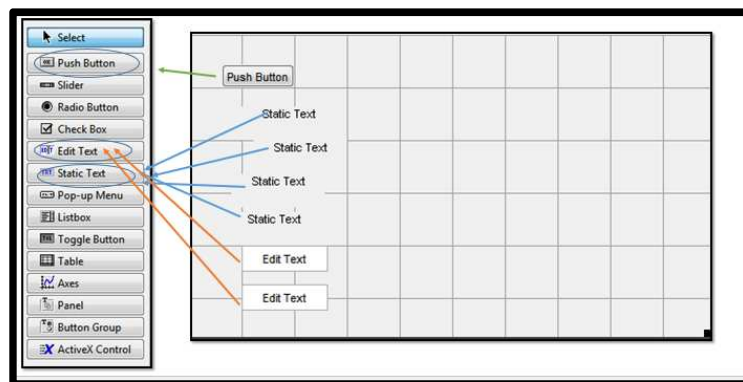


Figura 4. 13: Diagrama de bloques de entorno gráfico.
Fuente: El Autor.

2. Luego procedemos a darle el diseño que sea agradable para el usuario, como son color del fondo de la ventana, tipo o tamaño de letra de cada una de las cajas de texto, etc. Se puede observar en la figura 4.14, que se usó 1 "static text" para el título del programa, 1 "static text" para indicar NUMERO A, 1 "static text" para indicar el NUMERO B y 1 "static text" para presentar el resultado final el cual está vacío y fondo blanco, también se puede observar que los "edit text" es donde ingresarán los números y el "pushbutton" es el botón SUMAR que ejecutará el resultado.

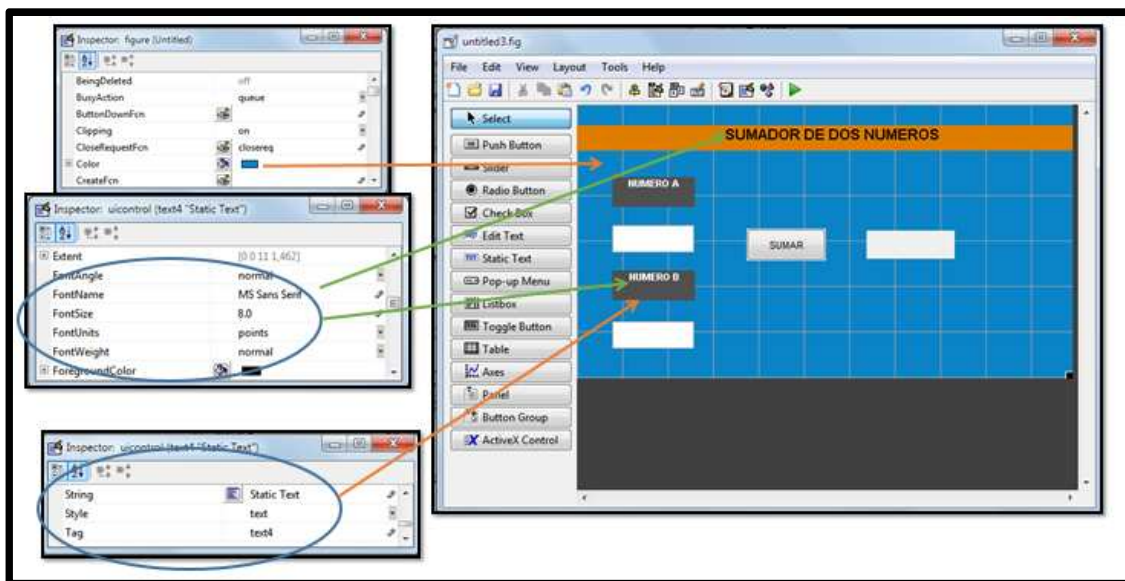


Figura 4. 14: Programación gráfica del sumador de dos números.
Fuente: El Autor.

3. Una vez realizado nuestro diseño procedemos a presionar el botón Guardar-Ejecutar y se generará el código de los objetos creados y también veremos el entorno gráfico que visualizará el usuario final, tal como se muestra en la figura 4.15.

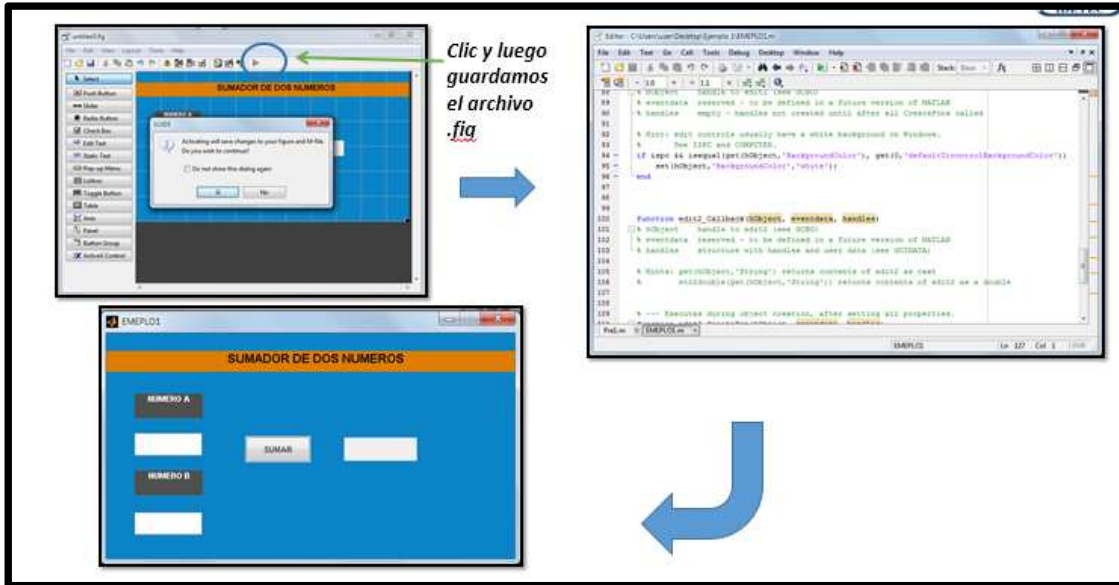


Figura 4. 15: Código de objetos para el sumador de dos números.
Fuente: El Autor.

- Empezaremos a programar los objetos, para esto vamos al entorno de trabajo y escogemos el objeto a programar para ver la subrutina que debemos programar. Damos clic derecho - View Callback – Callback (ver figura 4.16).

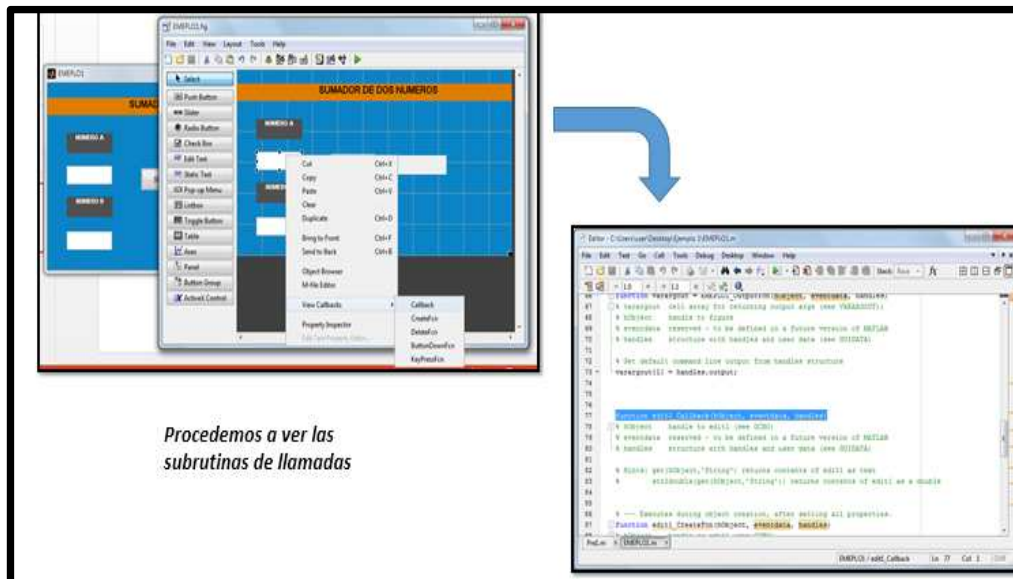


Figura 4. 16: Subrutina del sumador de dos números.
Fuente: El Autor.

5. Procedemos a programar. En este caso programaremos los “edit text” donde el usuario ingresará los números y el “pushbutton” el cual será el boton que ejecutará la suma y la presentará en el “static text”. En la figura 4.17, figura 4.18 y figura 4.19, se muestran los códigos fuentes para guardar el primer, segundo número y la suma de dos números respectivamente.

```
function edit1_Callback(hObject, eventdata, handles)
% hObject      handle to edit1 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%          str2double(get(hObject,'String')) returns contents of edit1 as a double

Val=get(hObject,'String'); %Almacenar valor ingresado
NewVal = str2double(Val); %Transformar a formato double
handles.edit1=NewVal; %Almacenar en identificador
guidata(hObject,handles); %Salvar datos de la aplicación
```

Figura 4. 17: Código fuente que guarda primer número.
Fuente: El Autor.

```
function edit2_Callback(hObject, eventdata, handles)
% hObject      handle to edit2 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%          str2double(get(hObject,'String')) returns contents of edit2 as a double
Val=get(hObject,'String'); %Almacenar valor ingresado
NewVal = str2double(Val); %Transformar a formato double
handles.edit2=NewVal; %Almacenar en identificador
guidata(hObject,handles); %Salvar datos de la aplicación
```

Figura 4. 18: Código fuente que guarda segundo número.
Fuente: El Autor.

```

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject      handle to pushbutton1 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
A=handles.edit1; %asiganmos el valor ingresado en la variable A
B=handles.edit2; %asiganmos el valor ingresado en la variable B
suma=A+B; %realizamos la suma
set(handles.text3, 'String', suma); %la presentamos en el <<static text>>

```

text3 es el nombre del "statictext"

Figura 4. 19: Código fuente de la suma de dos números.
Fuente: El Autor.

- Finalmente, se procede con la ejecución del programa realizado, que se muestra en la figura 4.20.

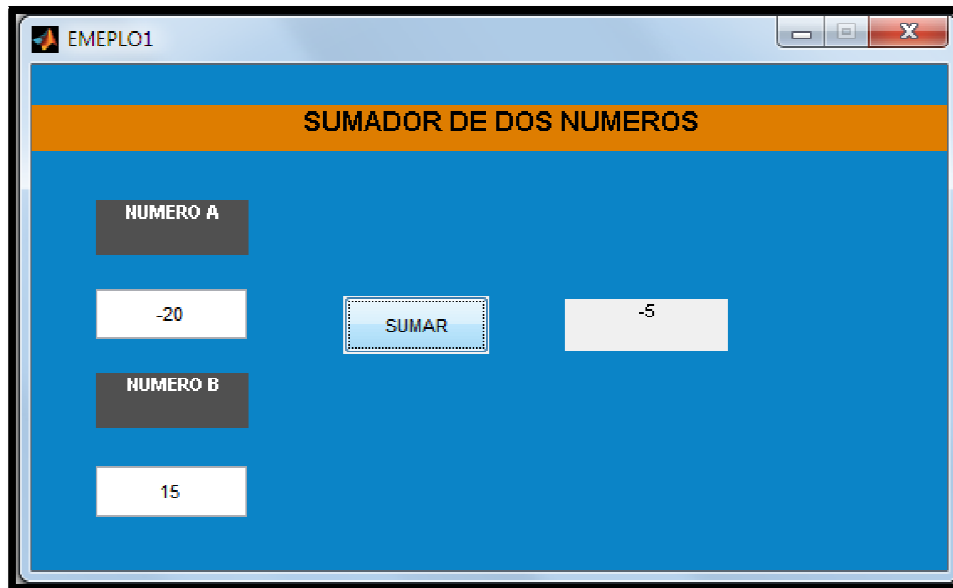


Figura 4. 20: Ventana GUI para la suma de dos números.
Fuente: El Autor.

4.3.2. Ejemplo: Graficador de Funciones

En este ejemplo podemos ver el uso de otros componentes que tiene la herramienta GUIDE. *Se requiere diseñar un programa que nos permita graficar funciones básicas (cuadrada, cúbica, seno, coseno) indicándole el rango.* Básicamente son los mismos pasos del ejemplo anterior, esta vez nos enfocaremos más en la programación que en el entorno gráfico diseñado.

En la figura 4.21 se muestra el entorno gráfico final diseñado y vemos dos nuevos componentes los cuales son “popupmenu” y “Axes”.

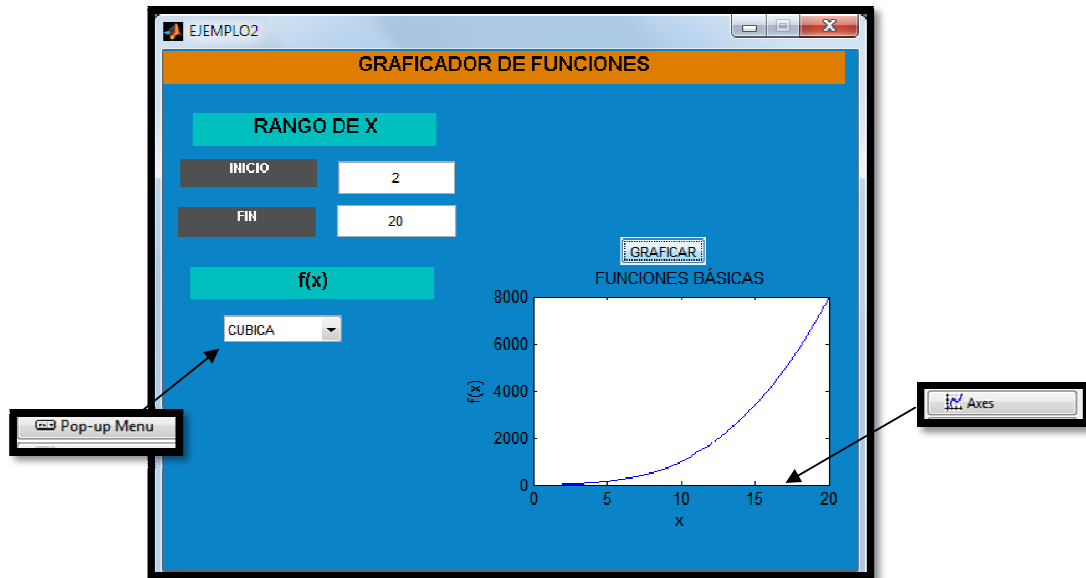


Figura 4. 21: Ventana GUI para graficar funciones.
Fuente: El Autor.

Antes de pasar a la programación revisaremos el componente “Pop-up menu”. Para poder agregar la lista de los funciones básicas debemos dar clic en STRING y se nos abrirá una ventana donde agregaremos la lista que nosotros deseemos, como se lustra en la figura 4.22.

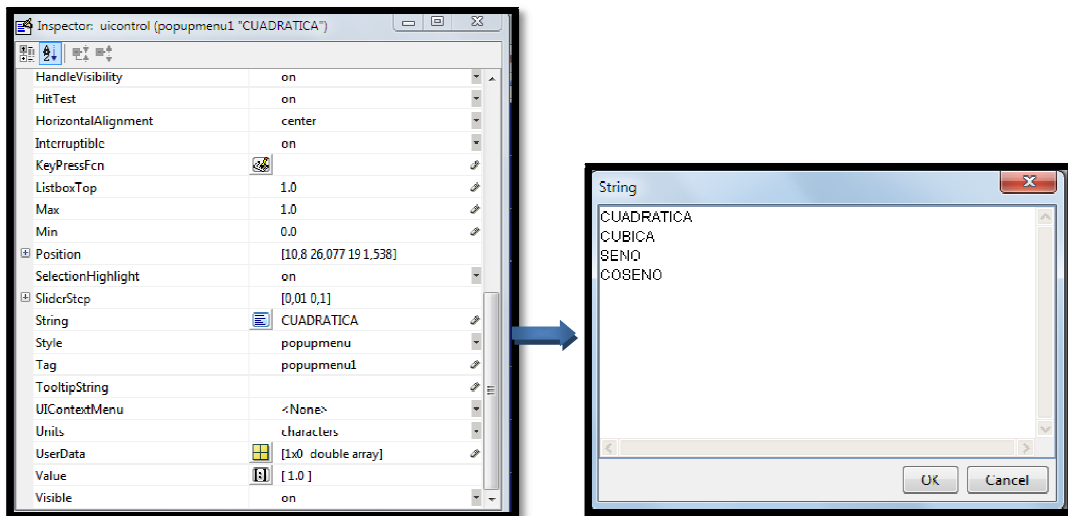


Figura 4. 22: Asignación de String para gráficas de funciones.
Fuente: El Autor.

A continuación, las figuras 4.23 y 4.24 muestran los códigos fuentes, que permiten el inicio del rango de una función.

```
function edit1_Callback(hObject, eventdata, handles)
% hObject      handle to edit1 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%         str2double(get(hObject,'String')) returns contents of edit1 as a double
Val=get(hObject,'String'); %Almacenar valor ingresado
NewVal = str2double(Val); %Transformar a formato double
handles.edit1=NewVal; %Almacenar en identificador
guidata(hObject,handles); %Salvar datos de la aplicación
```

Figura 4. 23: Código fuente para el inicio de una función - 1.
Fuente: El Autor.

En las figuras 4.24 y 4.25 se muestran los códigos fuentes, que permiten guardar el inicio del rango de una función.

```
function edit2_Callback(hObject, eventdata, handles)
% hObject      handle to edit2 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%         str2double(get(hObject,'String')) returns contents of edit2 as a double
Val=get(hObject,'String'); %Almacenar valor ingresado
NewVal = str2double(Val); %Transformar a formato double
handles.edit2=NewVal; %Almacenar en identificador
guidata(hObject,handles); %Salvar datos de la aplicación
```

Figura 4. 24: Código fuente para el inicio de una función - 2.
Fuente: El Autor.

Posteriormente, se programa el popupmenu1 (ver figura 4.25) lo que nos permitirá escoger la función que se desee graficar.

```

function popupmenu1_Callback(hObject, eventdata, handles)
% hObject      handle to popupmenu1 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1 contents as a cell array
%           contents{get(hObject,'Value')} returns selected item from popupmenu1
v=get(handles.popupmenu1,'Value')%Obtenemos el valor del pop menu.
in=handles.edit1 %obtenemo el valor del inicio
f=handles.edit2 %obtenemo el valor del fin
x=in:1:f %Se genera Rango de de x con intervalo de 1
y=0; %Se iniciliza.
switch v

    case 1 %CUADRÁTICA
        y=x.^2
        axis([in f min(y) max(y)]);%LIMITES DE LA GRAFICA
    case 2
        y=x.^3 %CÁBICA
        axis([in f min(y) max(y)]);
    case 3
        y=sin(x) %SENO
        axis([in f min(y) max(y)]);
    case 4
        y=cos(x) %COSENO
        axis([in f min(y) max(y)]);

end
handles.funcion=y %se guarda valor de y;
%en este caso no usamos handles.popupmenu1
handles.valor_x=x %guardamos el valor de x para usarlo en otras subrutinas
guidata(hObject,handles)

```

Figura 4. 25: Asignación de String para gráficas de funciones.
Fuente: El Autor.

En la figura 4.26 se muestra la programación del pushbutton donde se graficará la función escogida anteriormente.

```

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject      handle to pushbutton1 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
plot(handles.valor_x,handles.funcion)% grafica de la funcion plot(x,y)
title('FUNCIONES BÁSICAS')
xlabel('x');
ylabel('f(x)');

```

Figura 4. 26: Código fuente para graficar funciones.
Fuente: El Autor.

4.4. Comunicación Serial.

Se procederá a comunicar la tarjeta de entenamiento mediante el puerto serial. En la figura 4.27 se muestra la representación de la práctica en donde vemos los componentes de la tarjeta la cual vía USB se comunicarán por el

puerto COMX del ordenador. Más adelante podremos ver como obtenemos y enviamos datos mediante un software el cual es Matlab y daremos un entorno gráfico visto en el acápite anterior.

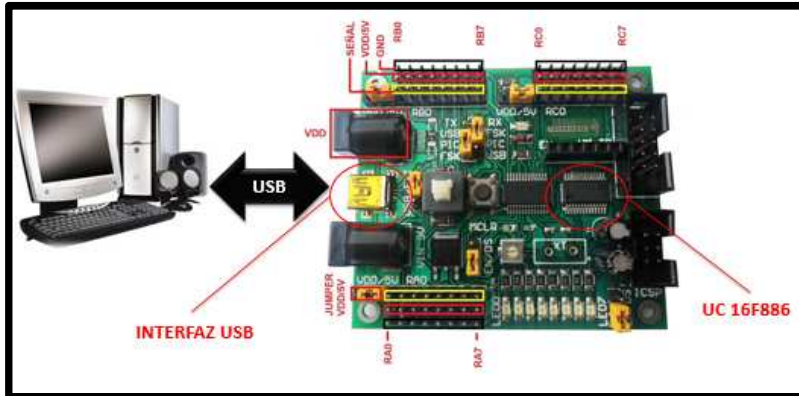


Figura 4. 27: Código fuente para graficar funciones.
Fuente: El Autor.

4.4.1. Ejemplo: Envío de datos por comunicación serial

Programar el módulo MEI&T04 para que envíe por puerto serial el conteo ascendente de una variable de 8 bits (0-255), el envío se realizará cada 500 milisegundos. Monitorear el dato recibido en código ASCII por el AccesPort. A continuación, la figura 4.28 muestra el código fuente que permite grabar en el módulo de entrenamiento.

```

program ejemplo1
'Declarations section
'1)-----SYMBOL-----
SYMBOL LEDES=PORTB
'2)---CREAR Y SET VARIABLES---
DIM CNT AS BYTE
DIM txt as string[3]
CNT=0
'3)---CONFIG IN (1)- OUT (0)---
TRISA=%00000001 '<7,6,5,4,3,2,1,0>
TRISB=%00000000 'LEDS-> OUT
TRISC=%00000000 '<7,6,5,4,3,2,1,0>
TRISE=%00000000 'RE3-> IN BOTONERA
'4)---CONFIG IN: DIG (0)- ANALOG (1)---
ANSEL= %00000000 '<7,6,5,4,3,2,1,0>
'An0->A0, An1->A1, An2->A2, An3->A3, An4->A4
ANSELH= %00000000
'<7,6,5,4,3,2,1,0>
'An8->B2, An9->B3, An10->B1, An11->B4, An12->B0,
An13->B5
'5)--- COMUNICACIÓN-----
UART1_Init(9600)
' Initialize UART module at 9600 bps
Delay_ms(50)

main:
' Main program
while(1)
inc(CNT)
LEDS=CNT
ByteToStr(CNT, txt)
UART1_Write_text(txt)' Dato
delay_ms(500)
wend
end.

```

Figura 4. 28: Código fuente para comunicación serial.
Fuente: El Autor.

Una vez grabado el código procedemos a ver en el programa Access Port los datos enviados serialmente por el módulo de entrenamiento. En la figura 4.29 se muestra la Configuración del access port. Primero se configura donde tendremos que seleccionar los baudios y el puerto, los demás parámetros son por default.

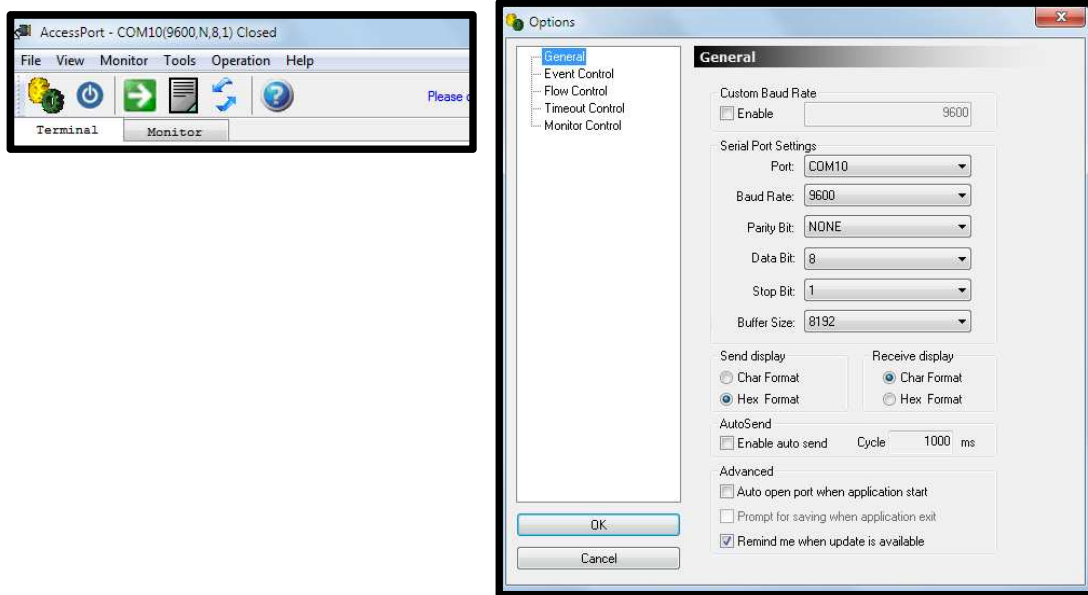


Figura 4. 29: Configuración del Access Port.
Fuente: El Autor.

Una vez realizado esto, se procede a realizar las pruebas correspondientes y ver que datos del módulo está enviando (ver figura 4.30).

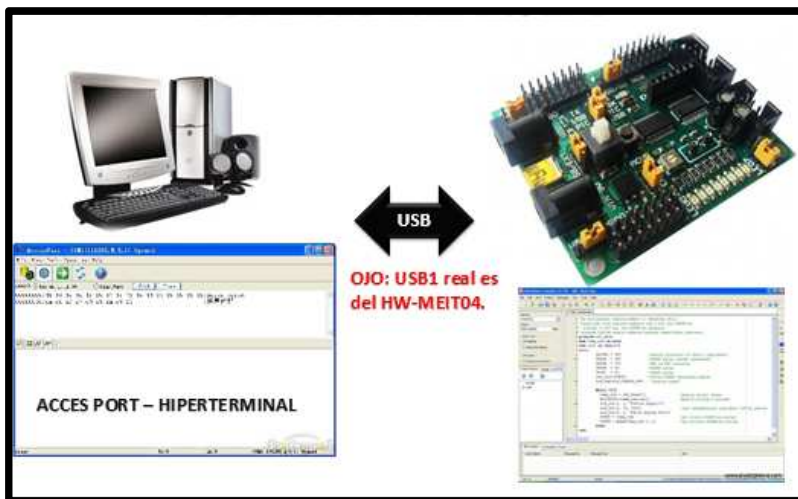


Figura 4. 30: Comunicación Hiperterminal.
Fuente: El Autor.

4.4.2. Recepción/envió de datos por comunicación serial MATLAB

A continuación, se procedió a utilizar el código correcto para enviar y recibir datos por medio de MatLab mediante un entorno gráfico. Para esto, se crea el programa que permite enviar – recibir datos desde el puerto serial del ordenador desde un entorno gráfico utilizando Matlab-GUIDE. En la figura 4.31 se observan todos los componentes del entorno gráfico, usado en inciso anterior, para lo cual se procedió a enfocarse en la programación del programa para comunicación serial con el puerto COM del ordenador.

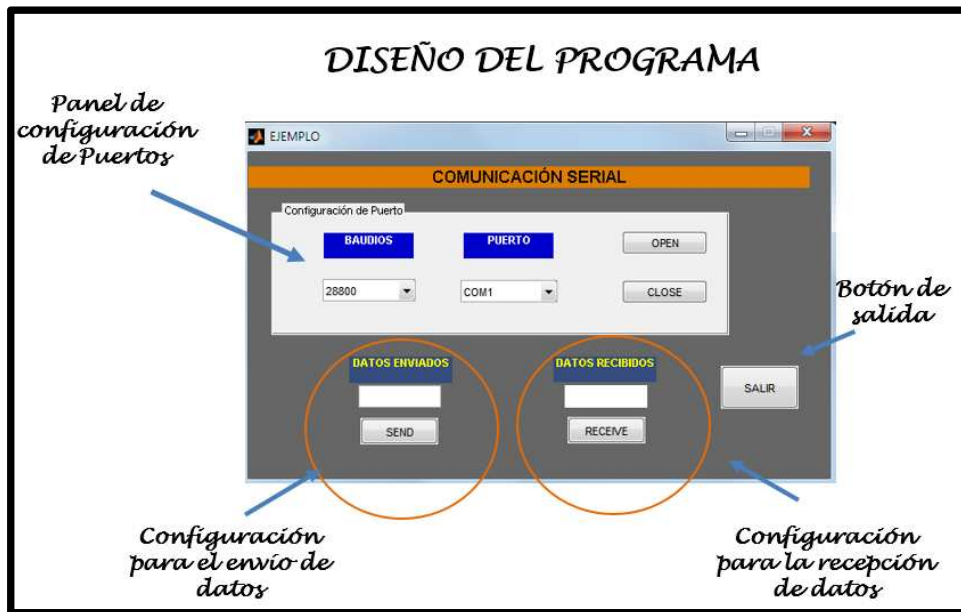


Figura 4. 31: Configuración del Access Port.
Fuente: El Autor.

Para la programación se la dividió en 4 partes básicas, que son:

1. Panel de configuración

En la figura 4.32 se muestra la configuración del popupmenu1 para la selección de los baudios

```

function popupmenu1_Callback(hObject, eventdata, handles)

v=get(handles.popupmenu1,'Value');%obtenemos la posición que corresponde
switch v                                %a los baudios selccionados
    case 1
        baudios=28800
    case 2
        baudios=19200
    case 3
        baudios=9600
end
handles.baudios=baudios;%en handles.baudios se guardara el valor seleccionado
guidata(hObject,handles);

```

Figura 4. 32: Código fuente para configuración de baudios.
Fuente: El Autor.

En la figura 4.33 se muestra el programa que permite la configuración del popupmenu2, la cual permite la selección de los puertos COM para la comunicación serial.

```

function popupmenu2_Callback(hObject, eventdata, handles)

v=get(handles.popupmenu2,'Value');%obtenemos la posición que corresponde a los puertos
switch v                                %COM seleccionados.
    case 1
        puerto='COM1'
    case 2
        puerto='COM2'
    case 3
        puerto='COM3'
    case 4
        puerto='COM4'
    case 5
        puerto='COM5'
    case 6
        puerto='COM6'
    case 7
        puerto='COM7'
    case 8
        puerto='COM8'
    case 9
        puerto='COM9'
    case 10
        puerto='COM10'
    case 11
        puerto='COM11'
    case 12
        puerto='COM12'
    case 13
        puerto='COM13'
    case 14
        puerto='COM14'
    case 15
        puerto='COM15'
end
handles.puerto=puerto;%en handles.puerto se guardara el valor seleccionado
guidata(hObject,handles);

```

Figura 4. 33: Código fuente para configuración y selección de los puertos COM.
Fuente: El Autor.

Aquí se podemos ver como se configuró el pushbutton1 para abrir el puerto serial, tal como se muestra en la figura 4.34.

```
function pushbutton1_Callback(hObject, eventdata, handles)

serpic=serial(handles.puerto);%seteamos en la variable serpic valor del puerto seleccionado
set(serpic,'BaudRate',handles.baudios);%seteamos en la variable serpic valor de los baudios seleccionado
set(serpic,'DataBits',8);%seteamos 8 bit de datos
set(serpic,'Parity','none');% seteamos Paridad ninguna
set(serpic,'Stopbits',1);%seteamos bit de parada
set(serpic,'FlowControl','none');%seteamos flujo de control
set(serpic,'Timeout',1);%seteamos tiempo de comando serial
fopen(serpic);%abrimos el puerto serial
handles.serpic=serpic;% todos los parámetros seteados se guardan en handles.serpic
guidata(hObject,handles); %Salvar datos de la aplicación
```

Figura 4. 34: Código fuente para configuración de pushbutton1.
Fuente: El Autor.

Posteriormente, la figura 4.35 se puede ver la configuración del pushbutton2, para poder cerrar el puerto serial,adicionalmente agregamos un mensaje de información para el usuario cuando realice esta acción.

```
function pushbutton2_Callback(hObject, eventdata, handles)

fclose(handles.serpic);%cerramos el puerto serial
delete(handles.serpic);%eliminamos la varibale handles.serpic
warndlg('Puerto Cerrado','Curso_GUIDE');%mensaje de informacion
```

Figura 4. 35: Código fuente para configuración de pushbutton2.
Fuente: El Autor.

✓ Datos enviados

Para proceder a que los datos sean enviados, se programará el dato ingresado por el usuario el cual se desea enviar por el puerto serial (ver figura 4.36).

```
function edit1_Callback(hObject, eventdata, handles)

v=get(hObject,'String'); %Almacenar valor ingresado
handles.valor_send=v %guardamos el valor en handles.valor_send
guidata(hObject,handles); %Salvar datos de la aplicación
```

Figura 4. 36: Código fuente para configuración de envío de datos.
Fuente: El Autor.

Y posteriormente se programa al pushbutton3 (SEND), el cuál ejecuta el dato ingresado por el usuario para su envío por el puerto serial.

```
function pushbutton3_Callback(hObject, eventdata, handles)

    fprintf(handles.serpico, '%s', handles.valor_send) %enviamos por el puerto serial
    pause(0.2); %formato string el dato ingresado.
```

Figura 4. 37: Código fuente para configuración pushbutton3 para enviar datos.
Fuente: El Autor.

✓ Datos de recepción

Para los datos de recepción, se programa el pushbutton4 (RECEIVE) donde al ejecutar los datos enviados por el puerto serial serán leídos y presentados en un static text.

```
function pushbutton4_Callback(hObject, eventdata, handles)

    valor=fscanf(handles.serpico, '%s') %leemos por el puerto serial formato string
    set(handles.dato_rec, 'String', valor) %presentamos en el static text
    %valor recibido
    pause(0.2)
```

Figura 4. 38: Código fuente para configuración pushbutton4 para recepción de datos.
Fuente: El Autor.

✓ Boton de salida

La figura 4.39 se muestra la programación del pushbutton8 (SALIR) donde al ejecutar le presentará un aviso de pregunta al usuario donde le indica si desea salir o no del programa.

```
function pushbutton8_Callback(hObject, eventdata, handles)

opc=questdlg('¿Desea salir del programa?', 'SALIR', 'Si', 'No', 'No'); %mensaje de pregunta
if strcmp(opc, 'No')
return;
end
clear,clc,close all
```

Figura 4. 39: Código fuente para configuración pushbutton8 para salir del programa.
Fuente: El Autor.

4.4.3. Pruebas.

Para realizar la prueba del programa que se ha diseñado, nos apoyaremos en dos programas, los cuales son VSPE y el Acces port. El VSE nos ayudará creando dos puertos virtuales COM y el Access Port nos ayudará a visualizar que está pasando en el envío/recepción de datos.

El VSE es un programa fácil de manejar el cual nos ayudará a crear puertos vituales para pruebas de comunicación. Esta configuración lo que se le envié para el puerto COM1 también le llegará al puerto COM. En la figura 4.40 se muestra la creación del puerto virtual.

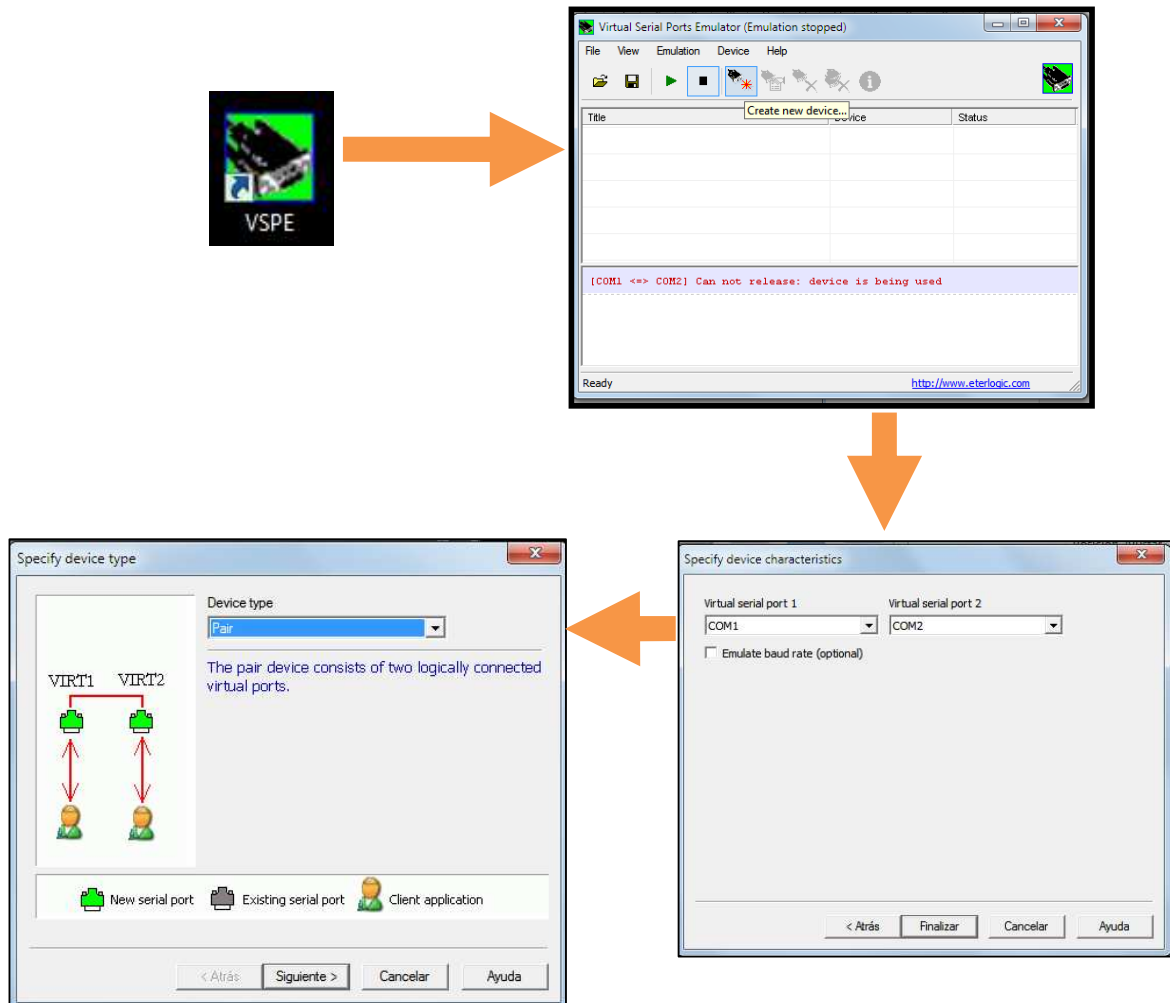


Figura 4. 40: Creación de puertos virtuales.

Fuente: El Autor.

Una vez configurado los puertos virtuales los cuales son COM1 y COM2 procederemos con la ejecución del programa. Configuramos el Puerto, seleccionamos los baudios y el puerto a usar, en la figura 4.41 es el puerto COM1 virtual. Luego procedemos a abrir el puerto dando clic en OPEN.



Figura 4. 41: Configuración del puerto.
Fuente: El Autor.

Una vez configurado correctamente el puerto, enviamos el dato “ideas” a través del puerto serial. El dato “ideas” se observa en la figura 4.42, esto ocurre mediante la configuración de dos puertos virtuales, el Access Port (configurar Puerto COM2) y Baudios 28800.

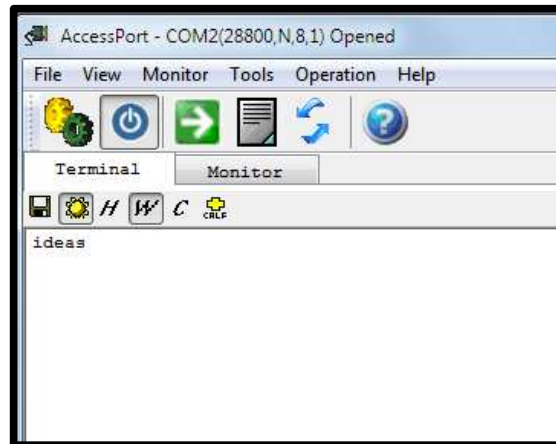


Figura 4. 42: Dato “ideas” enviado por el puerto virtual Access Port.
Fuente: El Autor.

Si queremos enviar datos y leerlos desde Matlab procedemos a enviar datos desde el Access Port, tal como se muestra en la figura 4.43.

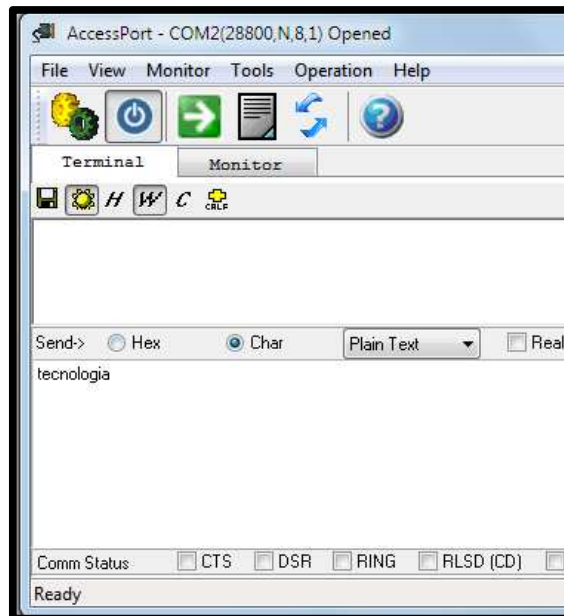


Figura 4. 43: Access Port para enviar datos y lectura desde MatLab.
Fuente: El Autor.



Figura 4. 44: Datos recibidos y lectura en GUI de MatLab.
Fuente: El Autor.

Si se desea salir del programa, se selecciona el botón SALIR, la misma nos pregunta “¿Desea salir del programa?”, tal como se muestra en la figura 4.45.



Figura 4. 45: Salida del programa GUI de MatLab.
Fuente: El Autor.

4.5. Validación Práctica del Control de un Motor DC con PWM y monitoreo de las RPM del motor con encoder óptico desde una interfaz gráfica (Matlab-GUIDE).

Para la validación del presente trabajo de titulación, se procederá a realizar la práctica, se procede a programar el módulo de entrenamiento para poder establecer la correcta comunicación entre Matlab y el mencionado módulo. En la figura 4.46 se muestra las conexiones entre el módulo de entrenamiento, encoder óptico y motor DC.



Figura 4. 46: Conexiones para validación práctica.
Fuente: El Autor.

A continuación, se muestra en la figura 4.47 la programación para el control DC y monitoreo de velocidad del motor DC desde MatLab.

```

program ejemplo11
' control de y monitoreo de velocidad del motor DC desde Matlab...
' Declarations section
'1)-----SYMBOL.-----
SYMBOL DIR1= PORTA.2 'Motor 1
SYMBOL NDIR1= PORTA.4 'Motor 1
SYMBOL DIR2= PORTA.3 'Motor 2
SYMBOL NDIR2= PORTA.5 'Motor 2

SYMBOL BTN=PORTE.3 'Botonera
SYMBOL Encoder=Portb.0 ' sensor Encoder optico *****
SYMBOL POT=PORTA.0 ' Potenciometro
'2)-----CREAR Y SET VARIABLES-----
DIM ADC_POT,BAND,datos AS BYTE
dim RPS,CntDesTMRO as longword
dim txt as string[10]

```

Figura 4. 47: Programación para control y monitoreo del motor DC.
Fuente: El Autor.

En la figura 4.48 se muestra la programación del contador de las RPS por medio de la subrutina de interrupción.

```

'*****interrupcion*****
sub procedure interrupt
  if TestBit(INTCON,TOIF) then 'Timer0 Overflow Interrupt Flag bit
    ClearBit(INTCON,TOIF)
    INC(CntDesTMRO) ' incrementa contador de interrup del TMRO
  end if
  if TestBit(INTCON,INTF) then 'INT External Interrupt Flag bit
    ClearBit(INTCON,INTF)
    RPS= 1000/((TMRO*128)/1000 + (CntDesTMRO*327)/10)
    TMRO=0
    CntDesTMRO=0
  end if
end sub

```

Figura 4. 48: Programación del contador de RPS.
Fuente: El Autor.

En la figura 4.49 se presenta la programación del main principal.

```

'3)-----CONFIG IN (1)- OUT (0)-----
TRISA=%00000001 '<7,6,5,4,3,2,1,0>
TRISB=%00000001 'Leds -> OUT
TRISC=%10000000 '<7,6,5,4,3,2,1,0>
TRISE=%00001000 'RE3 -> IN BOTONERA
'4)---CONFIG IN: DIG (0)- ANALOG (1)---
ANSEL= %00000001 '<_,_,_,An4,An3,An2,An1,An0>
'An0->A0, An1->A1, An2->A2, An3->A3, An4->A5
ANSELH= %00000000 '<_,_,An13,An12,An11,An10,An9,An8>
'An8->B2, An9->B3, An10->B1, An11->B4, An12->B0, An13->B5
'5)--- COMUNICACIÓN-----
UART1_Init(9600) ' Initialize UART module at 9600 bps
Delay_ms(50)
'6)--- PERIFERICO ADICIONAL---
'-----MOTOR AB-----DIR1,NDIR1-----
PWM2_Init(15000) 'Frecuencia de señal periodica PWM a 15KHz
PWM2_Start()
PWM2_Set_Duty(0)
'-----MOTOR CD-----DIR2,NDIR2-----
PWM1_Init(15000) 'Frecuencia de señal periodica PWM a 15KHz
PWM1_Start()
PWM1_Set_Duty(0)
'7) ---- Config otros registros----
' Registro de Opciones
OPTION_REG = %10000110 ' Pull up PORTB desactivado
' TMR0 prescaler 128
' Registro control de interrupciones
INTCON = %11110000 '0xF0 ' Activado interrupciones globales
' Activado interrupciones perifericas
' Activado interrupciones externas ytimer 0
' Main program
ADC_POT=0 BAND=0 dato=0
DIR1=1
NDIR1=0

while(1)
  if (UART1_Data_Ready() <> 0) then      ' If data is received,
    dato = UART1_Read()                  ' read the received data,
    select case band
      case 0
        if (dato=0x24) then
          BAND=1
        end if
      case 1
        PWM2_Set_Duty(dato)
        BAND=2
        if (dato=0) then DIR1=0 NDIR1=0 BAND=0 delay_ms(2000) end if 'frenado dinamico
      case 2
        DIR1=dato
        NDIR1=not(DIR1)
        BAND=0
    end select
  end if
  delay_ms(200)
  UART1_Write(0x24) ' byte de inicio
  WordToStr(RPS,txt)
  UART1_Write_text(txt) ' RPS
wend

```

Figura 4. 49: Programación del Main Principal.
Fuente: El Autor.

Una vez realizado esto procedemos a realizar la interfaz gráfica para el control del motor DC, para lo cual utilizaremos la herramienta Matlab-GUIDE ya tratados y aprendidos con anterioridad, junto con otros componentes adicionales como el slider y check box. Entonces, se utilizará la programación de comunicación serial, para lo cual el panel de configuración de puertos será el mismo (ver figura 4.50).

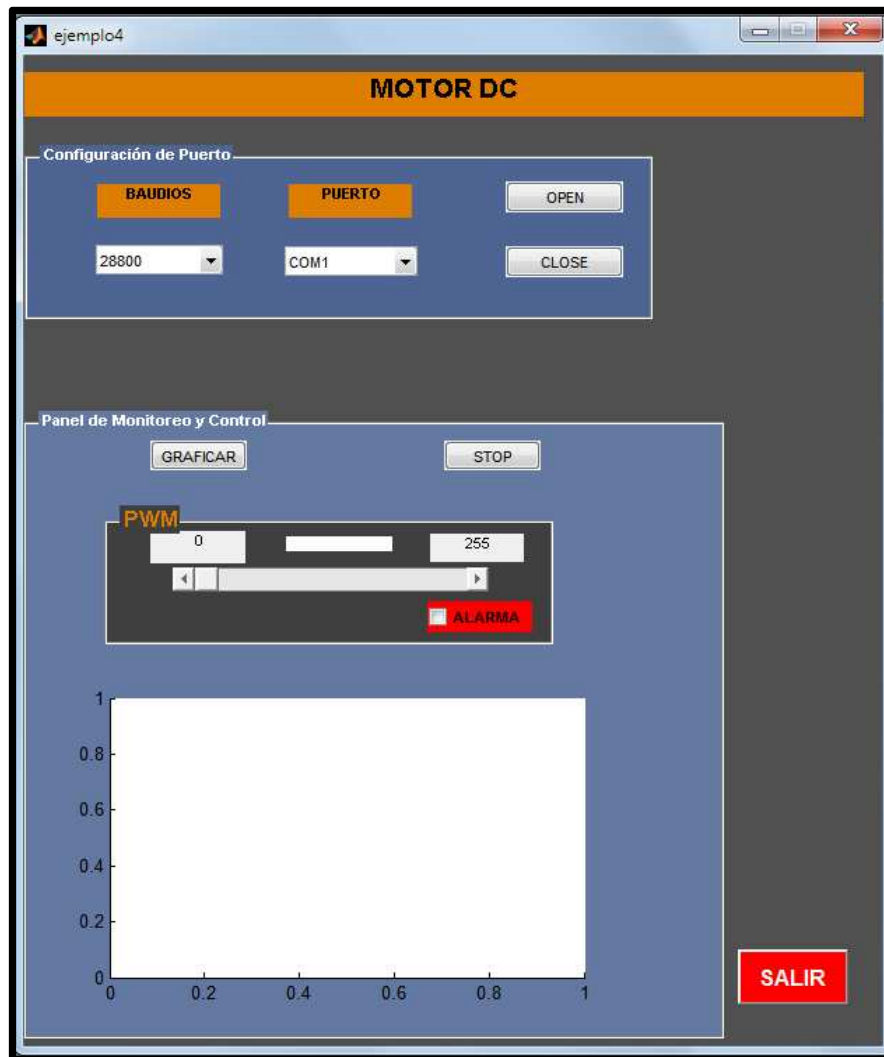


Figura 4. 50: Interfaz Gráfica GUI para configuración de puertos y panel de control.
Fuente: El Autor.

La programación para la presente práctica, se la divide en 3 partes básicas las cuales son:

a. Panel de configuración puertos

Utilizaremos el mismo panel de configuración de puertos visto anteriormente.

b. Panel de Monitoreo y control

Programaremos el slider (ver figura 4.51), aquí obtenemos el valor seteado por el usuario de las PWM, si el PWM seteado es mayor a 240, seteamos para que se ponga un visto en el check box, y finalmente enviaremos al módulo las PWM seteadas con la trama configurada, para que este lo pueda identificar(\$, PWM, GIRO)

```
function slider3_Callback(hObject, eventdata, handles)
v=0;%inicializamos
PWM=get(handles.slider3,'Value');%obtenemos el valor del PWM del slider
set(handles.indicador,'String',PWM)% ponemos el valor <<PWM>> en la caja de texto.

%-----Habilitacion del chek box-----
if PWM < 240
    set(handles.ALARMA,'Value',0) %El check box esta desabilitado
else
    set(handles.ALARMA,'Value',1)%Habilitamos el check box
end
%-----trama de envio de datos-----
a1='$';%24
a3=0;%00 dirección de giro
for i=1:2 %lo enviamos dos veces para asegurarnos que el modulo realice los cambios PWM
    fwrite(handles.serpic,a1,'char');%enviar <<bit inicio >>en forma de bytes por matlab
    fwrite(handles.serpic,PWM,'char');%enviar <<PWM >> en forma de bytes por matlab
    fwrite(handles.serpic,a3,'char');%enviar<<direccion>> bytes por matlab
end
```

Figura 4. 51: Programación del panel de Monitoreo y Control.

Fuente: El Autor.

Ahora, la figura 4.52 muestra la programación del Pushbutton – Graficar, la misma que nos permite graficar las RPS enviadas por el módulo de entrenamiento. Para la programación inicializamos las variables, después se agregan ciertos atributos a la gráfica y por último, graficamos en tiempo real las RPS y verificamos. Si el usuario decide parar (boton STOP) el ciclo infinito que obtiene datos y la gráfica se detiene.

```

function pushbutton1_Callback(hObject, eventdata, handles)
global B;%variable global que nos indicara si debemos parar de graficar
%*****Inicilizaci3n de Variables*****
B=0;
cla;%borra lo que tenga el Axes.
stop=0;%inicializamos
g=zeros(1,1);%inicializo
cont_muestras=1;%inicializo
%*****Atributos de la Gr3fica*****
title('SERIAL COMMUNICATION MATLAB + I&T')
xlabel('Numero de muestras')
ylabel('RPS')
grid on;
hold on;
ylim([0 200]);
%*****Toma de Datos y grafico en tiempo Real*****
while(stop==0)
    xlim([(cont_muestras)-20 (cont_muestras)+5]);
    fscanf(handles.serpic,'%s',1);%Obtiene el $
    volt=fscanf(handles.serpic,'%d');%Obtiene valor de las RPS
    g(cont_muestras)=volt(1);%guardamos en la variable g
    plot(cont_muestras,g(cont_muestras),'X-b','linewidth',2);%graficamos
    drawnow;
    cont_muestras=cont_muestras+1;%aumentamos el contador
    if(B==1)%preguntamos si el usuario decidio poner STOP
        stop=1;%cambiamos la bandera para detener el ciclo infinito
    end
end
%*****

```

Figura 4. 52: Programaci3n Pushbutton – Graficar.
Fuente: El Autor.

Programaci3n del Pushbutton – Stop (ver figura 4.53), se procede a cambiar el valor de la variable global B para que la otra subrutina pueda salir del ciclo infinito. Adem3s, presenta un mensaje el cual, indica al usuario que vuelva a poner el Slider al inicio para que pueda volver a graficar .

```

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
global B
B=1;
warndlg('Volver a encerrar el slider o darle clic en la posicin actual','Curso_GUIDE');

```

Figura 4. 53: Programaci3n Pushbutton – Stop.
Fuente: El Autor.

c. Bot3n de salida

Finalmente, la figura 4.54 se muestra la programaci3n del pussbutton5 (Salir) donde al ejecutar le presentara un aviso de pregunta al usuario donde le indica si desea salir o no del programa

```

function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
opc=questdlg('¿Desea salir del programa?', 'SALIR', 'Si', 'No', 'No');
if strcmp(opc, 'No')
return;
end
clear,clc,close all

```

Figura 4. 54: Programación Pushbutton – Salir.

Fuente: El Autor.

CAPÍTULO 5: CONCLUSIONES Y RECOMENDACIONES.

5.1. Conclusiones.

- A través del Estado del Arte de los Sistemas Microcontroladores, se pudo fundamentar y comprender la importancia de los microcontroladores.
- También se describió brevemente la plataforma de programación MatLab y el uso de la herramienta GUIDE, se pudo contribuir con una herramienta de programación gráfica que ha permitido cumplir con el objetivo general del presente trabajo de titulación.
- La parte experimental nos permitió validar correctamente la aplicación específica de manipular y controlar un motor DC con encoder óptico. Se comprobó que el programa creado para recibir y enviar datos al módulo de entrenamiento de microcontroladores, funciona correctamente.

5.2. Recomendaciones.

- Desarrollar más aplicaciones prácticas de microcontroladores a través de los módulos de entrenamiento que quedan en el Laboratorio de Electrónica, y que manejen a la vez programación de entorno gráfico (GUIDE – MatLab).
- Desarrollar cursos de educación continua u optativas para programar entorno gráfico en GUIDE – Matlab, pero cuya aplicación sean a nivel de microcontroladores.

REFERENCIAS BIBLIOGRÁFICAS

- Ciscar Martínez, V. A. (2010). *Diseño e Implementación de un Sistema Automático para la Caracterización Dinámica de probetas y elementos estructurales de construcción*. Valencia: Universidad Politécnica de Valencia.
- Dogan, I. (2008). *Programación de Microcontroladores PIC*. Barcelona: Marcombo S.A.
- Dogan, I. (2011). *PIC Basic Projects: 30 Projects using PIC BASIC and PIC BASIC PRO*. Burlington: Newnes.
- González Cayuñilo, R., & Pradines Pino, R. (2007). *Análisis de Software para Desarrollo entorno Gráfico LabView y Propuesta de Implementación para Laboratorio en el Instituto de Electricidad y Electrónica en Universidad Austral de Chile*. Valdivia: Universidad Austral de Chile.
- López Hernández, I., & Zuñiga Castro, D. (2009). *Protocolo para crear un sistema para reducir energía mediante el control de temperatura en casas habitación*. Juárez: Repositorio Universidad Autónoma de Ciudad de Juárez.
- Melchor Hernández, N. (2009). *Tarjeta de Desarrollo para Microcontroladores PIC*. México D.F.: Repositorio Instituto Politécnico Nacional (IPN).
- Reyes, C. A. (2008). *Microcontroladores PIC Programación en Basic*. Quito, Ecuador: RISPERGRAF.
- Terven Salinas, J. (17 de 06 de 2013). *Tervenet*. Obtenido de <http://www.tervenet.com/itmaz/micros2012/01%20Introduccion.pdf>
- Valdivieso Noroña, D. (2013). *Construcción de un Controlador de temperatura ambiental y humedad del suelo de un invernadero de tomate riñón orgánico utilizando el Microcontrolador PIC 16F877A*. Quito: Repositorio de la Escuela Politécnica Nacional (EPN).
- Verle, M. (2010). *PIC Microcontrollers - Programming in Basic*. Belgrado: mikroElektronika.