TUTORIAL MICROCONTROLADORES PIC
— *INICIACIÓN RÁPIDA* —

MIGUEL TORRES TORRITI

Contenidos

Version 1.0 – 26 Abril 2007 – `http://www.ing.puc.cl/~mtorrest/publicaciones.htm`.

Objetivos

Mediante explicaciones paso a paso, este documento busca iniciar rápida y económicamente al lector en el desarrollo de circuitos basados en microcontroladores. Para dicho propósito se empleará el popular microcontrolador PIC16F84, el cual por su simplicidad permite implementar circuitos con componentes de bajo costo disponibles comercialmente, y sin requerir de instrumentación de laboratorio complejo. También las herramientas de software de desarrollo que se presentan son de libre distribución y pueden conseguirse fácilmente sin costo.

Los pasos que deberá completar en este tutorial para desarrollar una aplicación basada en microcontroladores se resumen en:

(1) *Selección del microcontrolador*: En este tutorial se utilizaré el PIC16F84x por simplicidad y costo. Para aplicaciones más avanzadas se recomiendan el PIC16F87x o el PIC18F452.
(2) *Selección/Instalación del compilador/ensamblador*: Se empleará el compilador *PICC Lite* de la empresa *HI-TECH Software LLC.* [3], junto con el software de desarrollo *MPLAB IDE* de *Microchip Technology Inc.* [1]. Obtenga estos software de los sitios webs que se indican en las referencias e instálelos.
(3) *Selección/Instalación del software programador*: En este tutorial se utilizará el software programador PROG84. Obtenga el software programador PROG84 de las referencias que se indican en la sección 2.3 o 2.3.1. Para aplicaciones más avanzadas se recomiendan el IC-Prog o el WinPIC.
(4) *Construcción del circuito programador*: En este tutorial se empleará el circuito programador JDM PIC Programmer 2 (JDM2). La construcción del circuito programador JDM2 se explica en la sección 2.3.2 o en las referencias de la sección 2.3. El JDM2 es un programador simple y versátil para aplicaciones avanzadas. Alternativamente, se recomiendan el Multi PIC Programmer 5 Ver. 2 o el Progrmador PIC Pablin II. Si está iniciándose en la programación de microcontroladores y necesita desarrollar una aplicación rápidamente, tal vez sea recomendable que invierta en un programador de PIC comercial como los de Olimex, que son una alternativa económica a los programadores PICStart Plus de Microchip.
(5) *Desarrollo del software y programación*: Para escribir el programa que se incorporará al microcontrolador se empleará el ambiente de desarrollo MPLAB IDE. Es posible prescindir de este ambiente y escribir el código en un procesador de textos simple. Sin

embargo, la MPLAB IDE provee una serie de herramientas que facilitan el desarrollo y la administración del código.

(6) *Construcción del circuito base de la aplicación.* Los circuitos bases de la aplicación se presentan en en los ejemplos de este documento.

Al completar este tutorial, usted debería ser capaz de aplicar los pasos anteriores al desarrollo de aplicaciones basadas en otros microcontroladores además del PIC16F84.

1. Microcontroladores

1.1. **Introducción.**

Los micrconctoladores son computadores digitales integrados en un chip que cuentan con un microprocesador o unidad de procesamiento central (CPU), una memoria para almacenar el programa, una memoria para almacenar datos y puertos de entrada salida. A diferencia de los microprocesadores de propósito general, como los que se usan en los computadores PC, los microcontroladores son unidades autosuficientes y más económicas.

El funcionamiento de los microcontroladores está determinado por el programa almacenado en su memoria. Este puede escribirse en distintos leguajes de programación. Además, la mayoría de los microcontroladores actuales pueden reprogramarse repetidas veces.

Por las características mencionadas y su alta flexibilidad, los microcontroladores son ampliamente utilizados como el cerebro de una gran variedad de sistemas embebidos que controlan máquinas, componentes de sistemas complejos, como aplicaciones industriales de automatización y robótica, domótica, equipos médicos, sistemas aeroespaciales, e incluso dispositivos de la vida diaria como automóviles, hornos de microondas, teléfonos y televisores.

Frecuentemente se emplea la notación μC o las siglas MCU (por *microcontroller unit* para referirse a los microcontroladores. De ahora en adelante, los microcontroladores serán referidos en este documento por μC.

1.2. **Características de los Microcontroladores.**

Las principales características de los μC son:

- **Unidad de Procesamiento Central (CPU)**: Típicamente de 8 bits, pero también las hay de 4, 32 y hasta 64 bits con *arquitectura Harvard*, con memoria/bus de datos separada de la memoria/bus de instrucciones de programa, o *arquitectura de von Neumann*, también llamada *arquitectura Princeton*, con memoria/bus de datos y memoria/bus de programa compartidas.
- **Memoria de Programa**: Es una memoria ROM (*Read-Only Memory*), EPROM (*Electrically Programable ROM*), EEPROM (*Electrically Erasable/Programable ROM*) o Flash que almacena el código del programa que típicamente puede ser de 1 kilobyte a varios megabytes.
- **Memoria de Datos**: Es una memoria RAM (*Random Access Memory*) que típicamente puede ser de 1, 2 4, 8, 16, 32 kilobytes.
- **Generador del Reloj**: Usualmente un cristal de cuarzo de frecuencias que genera una señal oscilatoria de entre 1 a 40 MHz, o también resonadores o circuitos RC.
- **Interfaz de Entrada/Salida**: Puertos paralelos, seriales (UARTs, *Universal Asynchronous Receiver/Transmitter*), I^2C (*Inter-Integrated Circuit*), Interfaces de Periféricos

Seriales (SPIs, *Serial Peripheral Interfaces*), Red de Area de Controladores (CAN, *Controller Area Network*), USB (*Universal Serial Bus*).

- **Otras opciones**:
  - Conversores Análogo-Digitales (A/D, *analog-to-digital*) para convertir un nivel de voltaje en un cierto pin a un valor digital manipulable por el programa del microcontrolador.
  - Moduladores por Ancho de Pulso (PWM, *Pulse-Width Modulation*) para generar ondas cuadradas de frecuencia fija pero con ancho de pulso modificable.

La alta integración de subsistemas que componen un μC reduce el número de chips, la cantidad de pistas y espacio que se requeriría en un circuito impreso si se implementase un sistema equivalente usando chips separados.

Un aspecto de especial interés para el desarrollador de circuitos basados en microcontroladores son las interfaces de entrada/salida. A través de los pines del chip asociados a las interaces de entrada/salida el μC puede interactuar con otros circuitos externos enviandoles señales de comando o recibiendo estímulos correspondientes a variables externas. Por lo general varios pines de datos son bidireccionales, es decir pueden configurarse como entradas o salidas. Cuando son entradas, pueden adquirir datos interpretando el valor de voltaje como un valor lógico 0 o 1, mientras que cuando son salidas pueden entregar una señal binaria de voltaje cuya magnitud dependerá del valor lógico 0 o 1. Monitoreando el valor de las entradas, el microcontrolador puede responder a eventos externos y realizar una cierta acción, como variar las señales de salida de acuerdo al valor en la entradas. Para responder a eventos externos, los μCs cuentan con un recurso conocido como *interrupciones*. Las interrupciones son señales que se generan internamente en el microcontrolador que detienen la ejecución normal del programa para ejecutar alguna subrutina de respuesta al evento. Una vez ejecutada la subrutina de interrupción la ejecución del progama continua en el punto en que se encontraba antes de generarse la interrupción. Un ejemplo típico es el de un botón pulsador conectado a un pin de entrada. Una vez pulsado, se genera una señal de interrupción que iniciará la ejecución de la subrutina de interrupción, que por ejemplo podría activar un pin de salida para encender un led.

No todas las interrupciones necesariamente están asociadas al cambio del estado de los pines de entrada. También hay interrupciones que pueden estar asociadas al valor de una entrada AD, o al cumplimiento de un periodo de tiempo fijado por un *timer* o temporizador. Estas características dependerán del modelo de μC empleado.

1.3. **Proceso de Desarrollo.**

El proceso de desarrollo de una aplicación basada en microcontroladores se compone de las siguientes etapas principales, las cuales se explican en más detalle en las siguientes subsecciones.

- **Desarrollo de software**: Esta etapa corresponde a la escritura y compilaci'on/ensamblaje del programa que regirá las acciones del μC y los sistemas periféricos conectados a este.
- **Programación del** μC: En esta etapa el código de máquina correspondiente al programa desarrollado en la etapa anterior se descarga en la memoria del μC.
- **Prueba y verificación**: Por último, el μC debe conectarse al circuito base y someterse a pruebas para verificar el funcionamiento correcto del programa.

1.3.1. *Desarrollo del software.*

En esta etapa consiste en escribir y compilar/ensamblar el programa que determinará las acciones del μCy su funcionamiento. Existen distintas maneras de desarrollar el programa, dependiendo del lenguaje inicial que se utiliza para escribir el programa:

- Lenguaje Assembly - Lenguaje de Máquina/Código Objeto
  (`.asm`) →[ensamblador]→ (`.hex`, `.o`, `.bin`, `.coff`)
- Lenguaje de Alto Nivel - Lenguaje Assembly - Lenguaje de Máquina/Código Objeto
  (`.c`, `.cpp`) →[compilador]→ (`.asm`) →[ensamblador]→ (`.hex`, `.o`, `.bin`, `.coff`)

En la figura 1 se muestran las dos alternativas típicas que tiene el desarrollador para generar el código de máquina que es entendido por el microcontrolador.

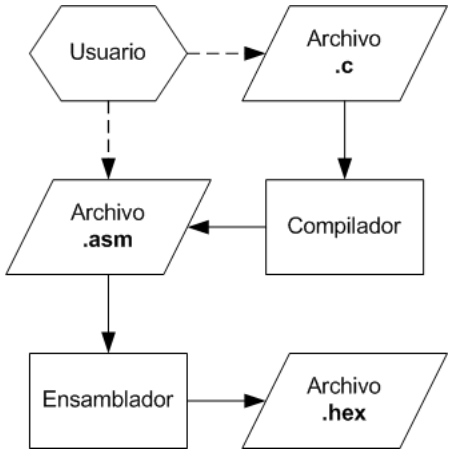

FIGURA 1. Etapas del desarrollo de software.

El método básico es escribir el programa en lenguaje *Assembly*[1] (lenguaje de Ensamblador) en un archivo de texto con extensión `.asm` y luego utilizar una programa ensamblador (*Assembler*) para generar un archivo en *lenguaje de máquina*, también denominado *código de máquina* o *código objeto* (*object code*), compuesto por instrucciones en código binario que son directamente entendidas por la CPU del microcontrolador. El ensamblador normalmente genera un archivo con extensión `.hex` (por hexadecimal), `.o` (por objeto), `.bin` (por binario), o `.coff` (*common object file format*) dependiendo del ensamblador. El lenguaje *Assembly* se compone de instrucciones mnemónicas de bajo nivel, es decir que están ligadas a las características del microcontrolador y con un número mínimo o nulo de abstracciones. Al carecer de abstracciones, el lenguaje *Assembly* es más difícil de emplear, requiere experiencia y un mayor tiempo de desarrollo. La ventaja es que el código de máquina generado a partir de un programa escrito en lenguaje de máquina es por lo general más eficiente, ya que el programa se desarrolla en un nivel cercano a las características del hardware.

Otra alternativa es emplear un lenguaje de alto nivel con una mayor cantidad de abstracciones, la cuales son más fáciles de usar y reducen los tiempos de desarrollo. Tal vez los lenguajes de

[1]En estricto rigor, *Assembly* es el lenguaje y *Assembler* es la herramienta de software que traduce el código *Assembly* a lenguaje de máquina. Sin embargo, es normal en el uso profesional emplear la palabra *Assembler* en forma ambigua para referirse tanto al lenguaje de programación como a la herramienta de software ensambladora.

alto nivel más comúnes para la programación de controladores es el C y C++, pero también existen otros lenguajes variantes del BASIC y el Pascal. Una vez escrito el programa en el lenguaje de alto nivel, será necesario emplear un compilador para traducirlo, ya sea a lenguaje de Ensamblador o directamente a lenguaje de máquina. Es importante considerar que el código de Ensamblador generado por los compiladores tiende a ser más lagro e ineficiente que aquel directamente desarrollado en lenguaje de Ensamblador. Esta desventaja puede ser crítica en ciertas aplicaciones que requieren un programas compactos y de una alta velocidad de ejecución. Un vez que el compilador ha generado el código de Ensamblador (`.asm`), será necesario utilizar un ensamblador para generar el código binario de máquina.

1.3.2. *Programación del* μC.

Este proceso corresponde a utilizar un programa en el PC que toma el código ensamblado (`.hex`, `.o`, `.bin`, `.coff`) para el μC específico, y lo envía mediante algún puerto (serial, paralelo, USB, etc.) a un dispositivo que lo escribe en la memoria del μC. Se acostumbra denominar *programador* tanto al sofware como al hardware involucrados para este propósito, lo cual puede prestarse a confusión. El software programador a veces recibe también el nombre de *downloader*, ya que su propósito es descargar o transferir desde el PC al μC el código ensamblado. En la figura 2 se muestran las componentes involucradas en el proceso de programación del μC. Es importante mencionar que no deben confundirse los términos *desarrollo o programación del software* y *programación del* μC, el primero se refiere a escribir el programa, mientras que el segundo se refiere transferir el código de máquina a la memoria del μC.

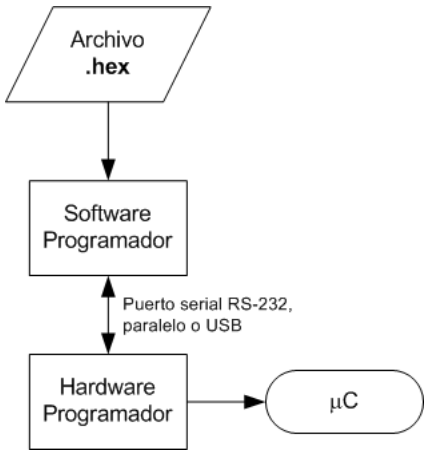

FIGURA 2. Programacion del μC.

1.3.3. *Prueba y verificación.*

Una vez programado el μC, se puede instalar en el circuito final para comprobar su adecuado funcionamiento. Existen heramientas de software que permiten simular el comportamiento de un μC, muy utiles cuando el programa alcanza cierta complejidad. Para resolver problemas en un circuito real, el instrumento más utilizado es el analizador lógico.

2. PIC16F84

2.1. **Introducción.**

El μC PIC16F84 de Microchip Technology Inc. [1] es uno de los microcontroladores más utilizados en proyectos electrónicos pequeos. A diferencia de versiones más avanzadas como el igualmente popular PIC16F87x, el PIC16F84 carece de convertidor A/D, PWM, comunicación serial por hardware y tiene menos memoria y puertos (conjuntos de líneas de datos) disponibles. Sin embargo, su fácil uso, precio reducido, lo han convertido en un μC muy popular y el favorito en un gran rango de aplicaciones. A pesar de que este μC no posee , muchas de las características que este μCno posee pueden ser implementadas por software.

El μC PIC16F84, o su version actual el μC PIC16F84A pertenece a la familiad Microchip de microcontroladores de rango medio de 8 bits con 18 pines. Como se muestra en la figura 3, este tiene 13 líneas de entrada/salida (RA0–RA5, RB0–RB7) con tecnología TTL/CMOS, es decir, 5 V para un estado lógico 1 y 0 V para el estado 0. Requiere un oscilador externo de hasta 20 MHz, se programa mediante un juego de 37 instrucciones en Assembly, que manejan datos de 8 bits, cuenta con un timer, un *watchdog* timer y responde las siguientes interrupciones:

- Cambios de estado en las líneas RB4 a RB7 del puerto B.
- Flanco de subida o bajada en la línea RB0/INT del puerto B.
- *Overflow*[2] del timer.

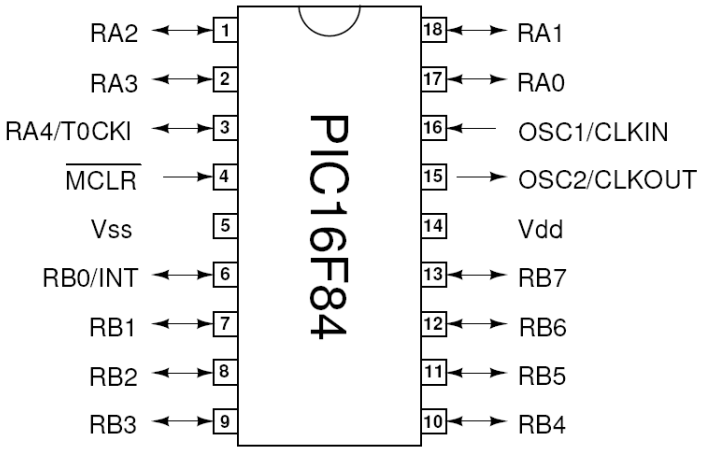

FIGURA 3. Pines del PIC16F84.

2.2. **Desarollo de Software para el PIC16F84.**

El proceso de desarrollo de aplicaciones para el PIC es equivalente al descrito en la figura 1. En lo que respecta al desarrollo de software para los μCs PIC existen varias alternativas empleando compiladores o ensambladores disponibles en forma gratuita como:

- **Assembler**:
  - *MPLAB IDE* [1].
    `http://www.microchip.com/`

[2] *Overflow* se refiere al evento que ocurre al incrementar un registro por sobre el valor máximo de bits que posee

  - `gpasm` parte de `gputils` [2].
    `http://gputils.sourceforge.net/`
- **C**:
  - *HI-TECH PICC Lite* [3].
    `http://www.htsoft.com/products/PICClite.php`
- **Pascal**:
  - *JAL* [4].
    `http://jal.sourceforge.net/`

Existen muchos otros compiladores incluso de lenguajes similares al BASIC que pueden encontrarse a través de Internet [6]. Un buen compilador de C para PIC es el *CCS PCWH* [5]. Este dispone también de muchos ejemplos de códigos en C que pueden reutilizarse en diversos proyectos, pero lamentablemente ya no dispone de una versión gratuita para estudiantes. Otros compiladores de C son los creados por los mismos fabricantes de los PIC. Sin embargo, estos tampoco son gratuitos, excepto el compilador para la familia PIC18. A continuación se explican algunas características de cada herramienta de desarrollo de programas para los μCs PIC.

2.2.1. *Desarrollo en Assembly.*

Los programas desarrollados en lenguaje Assembly pueden ser los más eficientes, aunque el proceso de desarrollo normalmente es tedioso y requiere de un aprendizaje más largo que con lenguajes de más alto nivel. Para seguir esta vía, se recomienda leer la hoja de datos (*datasheet* del PIC16F84, la cual permite familiarizarse con el conjunto de instrucciones disponibles. Existen varias opciones para ensamblar el código y generar código de máquina. Sin embargo, se recomienda utilizar las soluciones más probadas. En el caso de Windows, Microchip Technology Inc. provee el ensamblador por defecto de para los PICs a través de un ambiente integrado de desarrollo llamado MPLAB IDE (*Integrated Development Environment*), el cual se puede descargar en forma gratuita de su sitio web [1]. Por otro lado, en el ambiente Linux/BSD el `gpasm` es el ensablador del conjunto de utilidades `gputils` (GNU PIC Utilities), las cuales se pueden obtener gratuitamente [2].

2.2.2. *Desarrollo en C usando* HI-TECH PICC Lite.

El *PICC Lite* es la versión gratuita para estudiantes del compilador PICC desarrollado por HI-TECH Software. El PICC Lite tiene las mismas características profesionales de la versión comercial, aunque tiene ciertas limitaciones principalmente en términos de la longitud de los programas que se pueden desarrollar. En todo caso estas restricciones no limitan el desarrollo de aplicaciones razonablemente complejas y avanzadas. La disponibilidad del PICC LITE en el futuro no está asegurada, pero existen otros compiladores disponibles grauitamente con características similares. El uso de C puede ser muy conveniente para aquellos que ya estan familiarizados con este lenguaje y deseen iniciarse rápidamente en el desarrollo de circuitos basados en μCs PIC. El compilador genera directamente un archivo `.hex` con el código de máquina a partir del programa en C. El compilador posee numerosas opciones y ejemplos que se encuentran adecuadamente documentados.

2.2.3. *Desarrollo en JAL.*

El *JAL* es un lenguaje parecido al Pascal desarrollado por Wouter van Ooijen especificamente para μCs PIC y algunos μCs SX [4]. Al ser parecido al Pascal, el leguaje es más intuitivo que el C. El *JAL* también cuenta con librerías para realizar múltiples tareas comunes y al

igual que varios comipladores de C, este compila y ensambla a la vez entregando el código de máquina en un archivo `.hex`.

2.3. **Programación del μC PIC16F84.**

Es importante recordar que en este documento la expresión *programación del* μC se refiere al proceso de descargar a la memoria del μC el código de máquina generado por el ensamblador. Este proceso no debe confundirse con el proceso de escribir el programa tratado en la sección anterior, si bien en la práctica la expresión *programación del* μC se utiliza en forma ambigua para referirse tanto a la escritura del programa como a la descarga del mismo en la memoria del μC.

En Internet se pueden encontrar disponibles en forma gratuita numerosas duplas software-hardware para programar μCs PIC bajo diferentes sistemas operativos. Por lo general estas duplas software-hardware son interdependientes, lo que puede dificultar encontrar una combinación que funcione correctamente y sea a la vez simple y general. A continuación se presenta un resumen de las combinaciones software-hardware para programadores. Todos los programadores funcionan bajo Windows o DOS y se programan a través del puerto serial RS-232 a menos que se indique algo distinto. En el siguiente resumen, SW significa *software* y HW significa *hardware*.

- **SW: ElCheapo + HW: ElCheapo**
  PIC Programmer by Myke Predko
  `http://www.rentron.com/Myke4.htm`
  Observaciones: Este programador emplea un circuito similar al COM84 con cuatro partes adicionales y alimentación externa, pero usa el puerto paralelo. El diseño considera además de factores como costo y simplicidad, la flexibilidad de uso en diferentes computadores. Ya que aparentemente alternativas más minimalistas como el COM84 no funcionan correctamente en ciertos computadores que no entregan suficiente corriente por el puerto serial.
  Chips: Solamente probado con el PIC16F84.

- **SW: IC-Prog + HW: JDM / Fluffy2 / Ludpipo / Multi PIC Programmer 5 Ver. 2* / Pablin II** / ProPic II / Shaer / Tait /Shaer**
  IC-Prog Prototype Programmer - Bonny Gijzen
  `http://www.ic-prog.com/`
  Observaciones: El IC-Prog es tal vez el software programador más versátil junto con el WinPIC DL4HYF, ya que no sólo son capaces de programar una gran variedad de μCs, sino también pueden trabajar con variados circuitos programadores. Entre los programadores destacan algunos que no se encuentran mencionados en la página de IC-Prog. Uno de ellos es el Multi PIC Programmer 5 Ver. 2, diseñado por Feng para chips de 8/18/28/40 pines. Este está basado en el programador JDM y se alimenta del puerto serial, aunque en algunos aspectos el circuito parece una versión simplificada del ProPIC II:
  * *Multi PIC Programmer 5 Ver. 2*
  `http://feng3.cool.ne.jp/en/pg5v2.html`
  Otro diseño para chips de 8/18/28/40 pines que se alimenta del puerto paralelo es el programador Pablin II de Pablin. Este también se parece al Multi PIC Programmer 5 Ver. 2. El programador Pablin II se puede encontrar en:
  ** *Programador PIC Pablin II*

http://www.pablin.com.ar/electron/circuito/mc/ppp2/index.htm
Ambos programadores, el Multi PIC Programmer 5 y el Pablin II se presentan como opciones muy atractivas por su simplicidad y versatilidad.
Chips: PIC 12Cxx, 16Cxxx, 16Fxx, 16F87x, 18Fxxx, 16F7x, 24Cxx, 93Cxx, 90Sxxx, 59Cxx, 89Cx051, 89S53, 250x0, AVR , 80C51, otros.

- **SW: Picprog 1.8.3 + HW: JDM Jens Madsen PIC-Programmer 2**
Jaakko Hyvätti Picporg 1.8.3 documentation
http://hyvatti.iki.fi/~jaakko/pic/picprog.html
OS: Linux, Windows+Cygwin.

- **SW: PIP-02 + HW: COM84**
Beginners' PIC and AVR Page - Matthew Rowe
http://homepage.ntlworld.com/matthew.rowe/micros/dosgear.htm
http://homepage.ntlworld.com/matthew.rowe/micros/
Observaciones: El circuito programador COM84 es probablmente el más sencillo que existe que se alimenta directamente del puerto serial. Sin embargo, es posible que no funcione en muchos computadores nuevos que no son capaces generar el voltaje de alimentación suficiente con los nuevos circuitos de bajo consumo. En este caso se recomienda emplear el JDM PIC Programmer 2 que es una versión similar pero que funciona bajo todas condiciones.

  Un explicación del circuito COM84 en español puede encontrarse en:
Programador PIC y E2PROM sin fuente (puerto serie) - Pablin
http://www.pablin.com.ar/electron/circuito/mc/com84/index.htm
Existe un circuito más sencillo que el programador COM84 y que solo requiere unas resistencias. Sin embargo, este es sólo para la llamada *programación serial en circuito* o ICSP (*in-circuit serial programming*), es decir la programación del μC en su circuito base. Está técnica tiene ventajas evidentes en simplicidad, pero es una solución específica a cada μC y menos general, por lo que no se considerará en este tutorial. Para información adicional consulte la documentación del fabricante [1] o el ejemplo en:
http://www.jdm.homepage.dk/easypic2.htm
Chips: Solamente probado con el PIC16F84.

- **SW: PIX 1.13 + HW: COM84 / JDM / JDM84 / Ludi / Pixxer /Shaer / Tait**
COM84 PC Board
http://www.piclist.com/techref/piclist/cheapic/COM84.htm
PIX 1.13 - Bengt Lindgrens
http://home.swipnet.se/~w-53783/
http://ftp.iasi.roedu.net/mirrors/ftp.tapr.org/picsig/software/
Chips: Solamente probado con el PIC16F84.

- **SW: PROG84 + HW: COM84 / Ludpipo / Uniprog IV/ BR870**
Prog84, PIC16x84/24c16 programming utils
http://home3.inet.tele.dk/frda/picasm/prog.html
Observaciones: Proyecto de Wim Lewis, actualmente mantenido por Frank Damgaard.
Chips: El software programador PROG84 soportaría los PIC 16C84/16F84, 16C6x/7x/923/924, 16F87x, 12C5xx y 24C16 con los circuitos programadores Ludpipo / Uniprog IV/

BR870. Se ha verificado el funcionamiento del PROG84 con el circuito programador COM84, aunque es posible que por las características de este último sólo sea posible programar el PIC16F84.
OS: Linux, BSD, DOS, Windows.

- **SW: WinPic + HW: COM84 / JDM / JDM PIC Programmer 2 / Otros**
  DL4YHF's WinPic - A PIC Programmer for Windows
  `http://freenet-homepage.de/dl4yhf/winpicpr.html`
  `http://www.qsl.net/dl4yhf/winpicpr.html`
  Chips: Un gran número de la series PIC10/12/16/18. Sin embargo, debe tenerse en cuenta que el circuito programador COM84 sin modificaciones adicionales muy probablemente no podrá programar otro μC que el PIC16F84.

Existen una gran cantidad de otras alternativas, consulte las referencias en [6]. Un listado de programadores para Linux puede encontrarse en:
`http://www.micahcarrick.com/v2/content/category/4/2/8/`

Existen varias opciones de programadores comerciales, como el PICStart Plus II de Microchip, el cual se puede emplear junto con el MPLAB IDE, aunque alternativas más económicas son los programadores de Olimex:
`http://www.olimex.cl/`

Algunos circuitos programadores de μCs PIC ampliamente difundidos y principalmente para la programación de μCs de la serie PIC16x84 son:

- **COM84: Programador por Matthew Rowe**
  `http://homepage.ntlworld.com/matthew.rowe/micros/dosgear.htm`
  `http://www.piclist.com/techref/piclist/cheapic/COM84.htm`
  `http://sciencezero.4hv.org/electronics/com84.htm`
  `http://www.rentron.com/Myke4.htm`

- **JDM: Programadores por Jens Dyekjær Madsen**
  JDM PIC Programmer 2
  `http://www.jdm.homepage.dk/newpic.htm`
  `http://www.geocities.com/leon_heller/pic.html`

- **Multi PIC Programmer 5 Ver. 2 por Feng**
  `http://feng3.cool.ne.jp/en/pg5v2.html`

- **Programador PIC Pablin II**
  `http://www.pablin.com.ar/electron/circuito/mc/ppp2/index.htm`

- **Tait: Programadores por David Tait**
  `http://www.nomad.ee/PIC/`

Por su simplicidad, generalidad, bajo costo y disponibilidad tanto para Linux/BSD como para Windows, se recomienda utilizar la combinación **SW: PROG84 + HW: COM84** en el desarrollo de los ejemplos de este tutorial. Tengase presente que el programador COM84 estaría limitado a los μCs de la familia PIC16F84x, por lo tanto para otros proyectos considere utilizar **SW: IC-Prog/WinPIC + HW: Multi PIC Programmer 5 Ver. 2/Programador PIC Pablin II**. A continuación se explican brevemente el programador PROG84+COM84.

2.3.1. *Software Programador: PROG84.*

El software programador PROG84 fue desarrollado por Wim Lewis y actualmente es mantenido por Frank Damgaard, este puede obtenerse en:
`http://home3.inet.tele.dk/frda/picasm/prog.html`
Ver también:
`http://people.omnigroup.com/wiml/soft/pic/`

Este software programador puede ser utilizado con circuitos programadores por puerto serial o paralelo como el COM84, Ludpipo, Uniprog IV, BR870. Para utilizar el PROG84 deberá configurar el tipo de hardware y el puerto serial editando el archivo `lp_cfg` contenido en el directorio donde PROG84 fue colocado. Ver Anexo C con un ejemplo de archivo de configuración. Una vez configurado y conectado el hardware adecuadamente, la programación se efectua con el siguiente comando:

```
prog84 -x archivo.hex
```

o con

```
prog84 -azC UX -v -x archivo.hex
```

En este último caso la opción `a` indica que el programador verifique que los datos fueron cargados correctamente en el μC, la opción `z` indica que borre la memoria del μCantes de escribir el código del archivo `archivo.hex`, la opción `C` programa los *fuses* como `U` para el *power up timer* (on) y la configuración del oscilador, que en caso de un cristal es `X`. La opción `v` es para aumentar los mensaje (*verbosidad*) de estado del proceso de programación del μC.

2.3.2. *Circuito Programador JDM PIC Programmer 2.*

El circuito programador JDM PIC Programmer 2 (JDM2) es una interfaz de hardware muy simple que permite programar al μC PIC16F84 con muy pocos componentes, ya que se alimenta directamente del puerto serial RS-232 del PC sin requerir una fuente de alimentación externa como se muestra en la figura 4. Al lado izquierdo de esta figura se muestran las líneas del puerto serial, con sus respectivos números de pin en el conector DB9 o DB25. Las resistencias requeridas, tanto R1 de 1.5 $k\Omega$ como R2 de 10 $k\Omega$, pueden ser de baja potencia (0.25 W). Los diodos D1, D2, D5, D6 son diodos rectificadores cualquiera, aunque se recomienda el 1N4148. El diodo D3 debe ser un diodo Zener de 6.2 V como el BZV55C6V2 y debe colocarse en serie con el led como se muestra en el esquemático. El diodo Zener D3 y el led pueden reemplazarse por un único diodo Zener de 8.2 V. El diodo D4 también es un diodo Zener, pero de 5.1V como el BZV55C5V1. Tanto el condensador C1 de 100 μC, 16 V como el condensador C2 22 μC, 16 V deben ser electrolíticos. El condensador C2 también puede reemplazarse por un condensador de 100 μC, 16 V como C1. Dada la simpleza del circuito JDM2, se sugiere construirlo directamente en una placa PCB, de modo que sea más duradero y su operación más confiable.

Para que el PIC16F84 entre en modo de programación debe forzarse un estado lógico 0 en RB6 (pin 12) y RB7 (pin 13), mientras que $\overline{MCLR}$ (pin 4) debe llevarse a 0 por un periodo corto para resetear el μC, e inmediatamente después deber llevase y mantenerse en un voltaje entre 12 y 14 V durante toda la programación. Una vez que el μC ha entrado en modo de programación, se utiliza RB7 (PGD o Program Data) para ingresar serialmente la información, mientras que RB6 (PGC o Program Clock) se utiliza como seal de reloj para aceptar cada bit en RB7. Esto se logra mediante el software programador. Los detalles del protocolo de programación pueden encontrarse en la documentación del fabricante [1].

Es importante destacar que este diseño minimalista requiere que el puerto serial del PC entregue al circuito los niveles de voltaje y potencia adecuados. Por esta razón el programador puede presentar problemas de funcionamiento en ciertos computadores, como aquellos que utilizan circuitos seriales de baja potencia para ahorro de consumo muy comunes en algunos computadores portátiles. En todo caso, el circuito JDM2 puede ser modificado según la especificación de programación ya descrita para utilizar una fuente de alimentación externa.

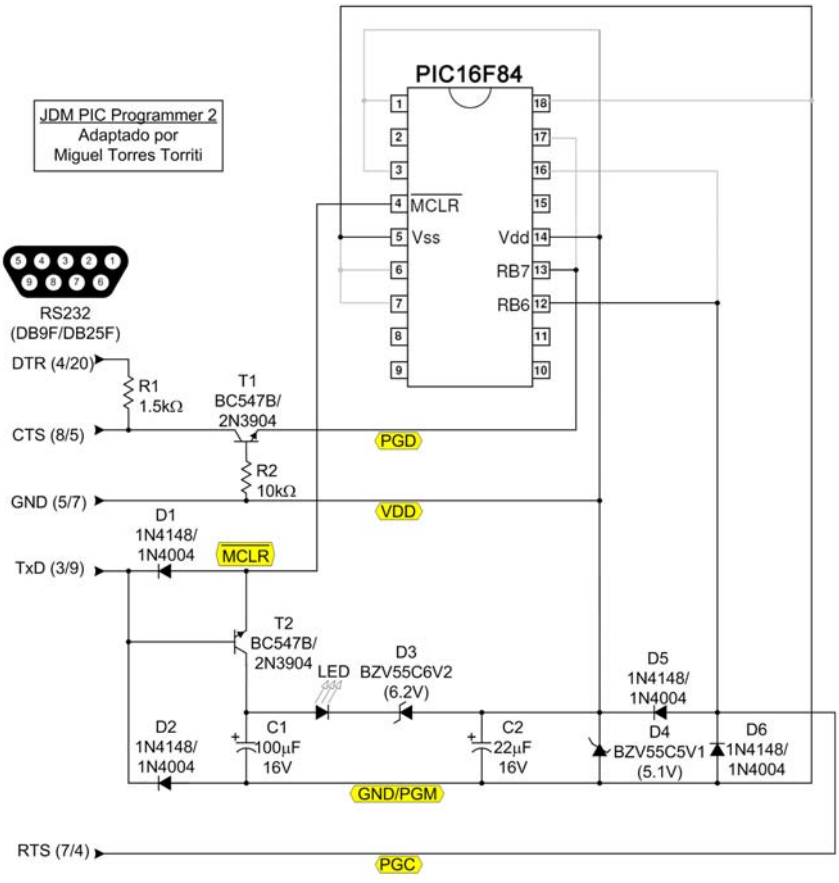

FIGURA 4. Programador serial minimalista JDM PIC Programmer 2.

2.4. **Circuito Base para Operación Regular.**

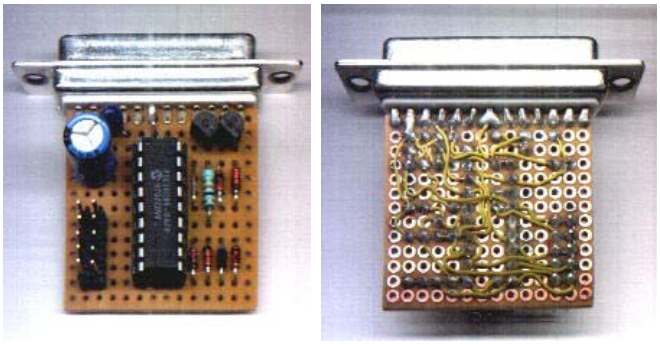

FIGURA 5. Programador JDM2 terminado.
Imagen en `http://www.jdm.homepage.dk/newpic.htm`.

En el circuito final, el PIC16F84 requiere ciertos componentes basicos, iguales para cualquier configuracion: un cristal y condensadores, que proveen la seal de reloj, y las lineas de alimentación, V_{dd}, y tierra, V_{ss}. $\overline{MCLR}$ es el reset, por lo que debe mantenerse en estado lógico 1 durante su funcionamiento. Esto se puede observar en la figura del ejemplo 1.

2.5. **Técnicas Avanzadas: Programación Mediante Bootloaders.**

Los *bootloaders* son un pequeño código binario que se descarga en los μCs y tiene por finalidad simplificar el desarrollo de programas ya que actuan como una especie de sistema operativo mínimo que tiene por finalidad obtener via un pin configurado para funcionar como puerto serial el código del programa que se requiere ejecutar. Para desarrollar programas usando *bootloaders* los pasos básicos son:

- Descargue el código del *bootloader* a la memoria del μC empleando un programador estándar como los mencionados en la subsección anterior.
- Arme un circuito con un conversor serial RS-232 a TTL como el circuito integrado MAX232.
- Compile/ensamble su código con algún compilador/ensamblador de los mencionados anteriormente.
- Coloque el circuito con el μC en modo de comunicación serial y descargue el código binario `.hex` usando el software *bootloader*.
- Re-inicie el circuito con el μC en modo normal para que se ejecute el programa que descargo a través del *bootloader*.

La ventaja de utilizar está t'ecnica es que permite muy facilmente hacer actualizaciones del código de la aplicación, comúnmente denominado *firmware*, que reside en la memoria del μC sin necesidad de sacarlo del circuito para colocarlo en un programador estándar. En este método el programador de microcontroladores solo se emplea una vez para descargar el microsistema operativo *bootloader* al μC.

En los siguientes sitios de Internet podrá encontrar dos *bootloaders* muy recomendables con explicaciones muy detalladas de como emplearlos.

- **PIC16F87x / 16F87xA**
  PIC micro and C - bootloader - Shane Tolmie
  `http://www.microchipc.com/PIC16bootload/`
  EHL elektronik - PIC downloader - Petr Kolomaznik

`http://www.ehl.cz/pic/pic_e.htm`

- **PIC8Fx52**
  PIC micro and C - bootloader - Shane Tolmie
  `http://www.microchipc.com/PIC18bootload/`

3. Ejemplo 1: Programación en JAL

3.1. Descripción.

Un ejemplo muy simple y demostrativo para iniciarse con un μC es hacer parpadear un LED. Para esto se implementará un programa que continuamente cambie el estado de un pin y espere cierto tiempo antes de repetir la acción. En este ejemplo se utilizará el lenguaje JAL para desarrollar el programa. En las siguientes secciones se indican los pasos necesarios para implementar el parpadeo de un LED.

3.2. Paso 1: Instalación y Configuración del Compilador JAL.

(1) Obtenga el compilador JAL en:
`http://jal.sourceforge.net/`
(2) Descomprima el archivo con el compilador.

3.3. Paso 2: Creación del Programa en JAL.

(1) Escriba el código del ejemplo que se muestra a continuación en un procesador de textos simple y grábelo en un archivo con extensión `.jal`.

```
 1: -- Ejemplo 1
 2:
 3: -- Inclusiones
 4: include 16f84_4
 5: include jlib
 6:
 7: -- Configuracion
 8: pin_b7_direction = output
 9:
10: -- Variables
11: var bit led is pin_b7
12: led = high
13:
14: -- Loop Principal
15: forever loop
16: -- Cambio de estado del pin
17: led = ! led
18: -- Retardo
19: delay_1ms(250)
20: end loop
```

Comentarios Generales del Programa: *El alto nivel de JAL permite que el código sea bastante autoexplicativo. Los comentarios se incluyen con el prefijo '--'.*

Es fundamental dejar un espacio en blanco entre el prefijo de comentario y la frase de lo contrario se producirán errores de compilación.

Los archivos en las líneas 4 y 5 son incluidos ya que estos contienen definiciones específicas del PIC16F84, como las correspondientes a las direcciones de memoria para `pin_b7` *y* `pin_b7_direction`*, y las funciones que generan el retardo.*

En la línea 8 se define la dirección de RB7 (pin 13) como pin de salida. En las líneas 11 y 12 se define la variable `led` *y se la asigna a RB7 con un estado incial activo.*

Finalmente, entre las líneas 15 y 20 se encuentra el loop *del programa, el cual se ejecuta indefinidamente. En este se cambia el estado del pin/led (línea 17) y se genera un retardo de 250 ms mediante la función* `delay_1ms` *(línea 19) para producir una pausa en el encendido o apagado del led. La función* `delay_1ms()` *causa un retardo de 1 ms multiplicado por su argumento. Su funcionamiento se basa en el hecho de que una instrucción (en Assembler) tarda 4 ciclos de reloj, por lo que se ejecuta un* loop *auxiliar el número de veces requerido según la velocidad del reloj, tal que el tiempo transcurrido antes de continuar con la ejecución del resto del programa sea igual al especificado. La velocidad del reloj se encuentra especificada en el programa dado que se incluyó la versión de 4 MHz (línea 4) de las funciones específicas de PIC16F84.*

3.4. **Paso 3: Compilación del Programa en JAL.**

(1) En el directorio donde guardó el archivo del ejemplo compile el código mediante la siguiente línea de comando:

```
<directorio>\jal -s<directorio>\lib archivo.jal
```

donde `<directorio>` se refiere al directorio donde instaló JAL, `archivo.jal` es el archivo donde guardó el código del ejemplo. El parámetro `-slib` indica al compilador que debe buscar las librerías utilizadas en la carpeta `lib`. JAL generará los archivos `archivo.asm` y `archivo.hex`, que corresponden al código en Assembly y al código de máquina ensamblado, respectivamente.

Alternativamente, se puede crear una carpeta llamada, por ejemplo, `work` en el directorio donde instaló JAL y colocar el código del ejemplo en dicha carpeta. Luego bastaría con ejecutar el siguiente comando de compilación desde la carpeta `work`:

```
..\jal -s..\lib archivo.jal
```

3.5. **Paso 4: Programación del μC.**

Puede programar el μC según lo explicado en las secciones 2.3, 2.3.1, 2.3.2 del presente documento, o bien usando el MPLAB IDE si posee algún programador Microchip como el PICSTART Plus. A continación se explican los pasos de programación desde MPLAB IDE en el caso que el archivo con el código de máquina (`.hex`) haya sido previamente generado fuera del MPLAB IDE como en este ejemplo.

(1) Inicie la aplicación MPLAB IDE.
(2) Ejecute la opción `File/Import...` de la barra de menú y seleccione el archivo `.hex` que desea descargar al μC.
(3) Verifique las opciones de configuración de MPLAB IDE. En particular verifique en la opción `Configure/Select Device...` de la barra de menú que el dispositivo corresponde al μC PIC 16F84 o 16F84A. Verifique también que los bits de configuración de la opción `Configure/Configuration Bits...` de la barra de menú son: Oscillator

= XT, Watchdog Timer = Off, Power Up Timer = Off, Code Protect = Off (a menos que desee otra configuración particular).
(4) Seleccione el hardware programador, por ejemplo el PICSTART Plus, empleando la opción `Programmer/Select Programmer` de la barra de menú.
(5) Habilite el hardware programador con la opción `Programmer/Enable Programmer`.
(6) Borre la memoría del μC presionando el icono de página en blanco a μC o la opción `Programmer/Erase Flash Device` de la barra de menú.
(7) Programe el dispositivo presionando el icono de página llena (amarilla) a μC o la opción `Programmer/Program` de la barra de menú.
(8) Normalmente el programador entregará un mensaje indicando si la programación fue exitosa. Sin embargo, si desea verificar si efectivamente la programación se realizó en forma correcta, emplee el botón de verificación (página con check) o la opción `Programmer/Verify` de la barra de menú. También puede ver los contenidos de la memoria del μC utilizando la opción `View/Program Memory` de la barra de menú. Si la programación se realizó en forma correcta tanto el código compilado como el código leido de vuelta del μC deben tener el mismo *checksum* que se muestra en la barra de menú del MPLAB IDE.

3.6. **Circuito del Ejemplo 1.**

En la figura 6 se muestra el circuito para el Ejemplo 1. Con excepción de los componentes conectados a RB7 (pin 13), los cuales corresponden a elementos específicos del ejemplo, el resto de los componentes constituyen la circuitería mínima para el funcionamiento del PIC16F84. El LED conectado a RB7 se encenderá de acuerdo al estado de la variable `led` en el código del programa del microcontrolador. Cuando el estado lógico de la variable `led` es 1, un voltaje de 5 V será aplicado al LED. Para limitar el exceso de corriente que podría dañar el LED o el puerto del PIC se coloca una resistencia en serie con el LED. Una configuración alternativa es conectar desde V_{dd} (5 V) el LED en serie con la resitencia a RB7. Esto causaría que el LED se encienda cuando la variable `led` tenga un estado lógico 0. Al diseñar otras aplicaciones es importante tener presente la corriente máxima que puede entregar o recibir un pin en el caso de la lógica positiva o negativa, respectivamente.

4. Ejemplo 2: Programación en PICC Lite

4.1. **Descripción.**

Este ejemplo ilustra la respuesta a estímulos externos utilizando interrupciones implementada en lenguaje C para el compilador PICC Lite. El programa permite encender y apagar un LED al presionar un botón pulsador. Antes de encender o apagar el LED este parpadea tres veces anunciando el cambio de estado.

4.2. **Paso 1: Instalación y Configuración del Compilador PICC Lite.**

(1) Obtenga el MPLAB IDE en:
`http://www.microchip.com/`
(2) Instale el MPLAB siguiendo todos los de la instalación.
(3) Obtenga el compilador PICC Lite en:
`http://www.htsoft.com/products/compilers/PICClite.php`
(4) Instale el compilador siguiendo todos los pasos de la instalación.

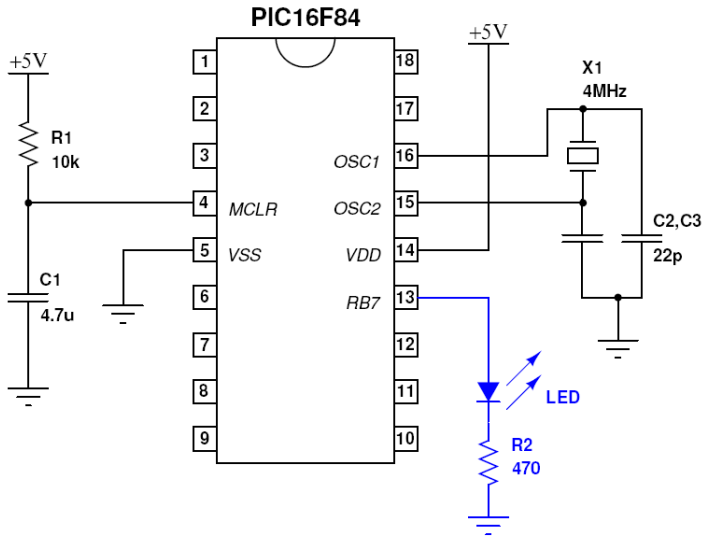

FIGURA 6. Circuito del ejemplo 1.

(5) Una vez instalado el PICC Lite, inicie el MPLAB IDE y actualice la lista de compiladores para que funcione junto con PICC Lite mediante los siguientes pasos:
   (a) En la opción de la barra de menú elija `Project/Set Language Tool Locations/HI-TECH Universal Toolsuite...`
   (b) Elija la opción `HI-TECH Universal ToolSuite/Executables/HI-TECH C Compiler` y fije la ubicación donde instaló el compilador `picl.exe`, por ejemplo: `C:\Program Files\HI-TECH Software\PICC\lite\9.60\bin\picl.exe`
   (c) Si es necesario agregue la ubicación donde se ubiquen archivos *include* (`.h`) bajo la opción `HI-TECH Universal ToolSuite/Default Search Path & Directories/Include Search Path`.

4.3. **Paso 2: Creación del Proyecto en MPLAB IDE.**

(1) Cree un directorio donde guardará los archivos del proyecto, por ejemplo: `ej2`.
(2) Copie los archivos `delay.h` y `delay.c` ubicados en la directorio: `C:\Program Files\HI-TECH Software\PICC\lite\9.60\samples` o donde haya instalado PICC Lite a la carpeta del proyecto que creo en el paso anterior.
(3) Elija la opción de la barra de menú `Project/Project Wizard` y luego presione `Next`.
(4) Elija el μC seleccionando PIC16F84A.
(5) Elija la herramineta de compilación activa `HI-TECH Universal ToolSuite` y actualice la ubicación del compilador de ser necesario.
(6) Escriba el nombre del proyecto, por ejemplo:`ej2`, y luego fije la ubicación del directorio del proyecto creado en el paso 1.
(7) Seleccione los archivos `delay.h` y `delay.c`, y agréguelos al proyecto con el botón `Add`.
(8) Si todas las opciones elegidas están conformes, presiones `Next`, de lo contrario retroceda a los pasos anteriores y haga las correcciones necesarias.
(9) Elija la opción de la barra de menú `File/New` para crear el archivo que contendrá el programa del ejemplo. Escriba el código que se muestra a continuación y guarde

el archivo con extensión `.c`, por ejemplo con el nombre `ej2.c` en el directorio del proyecto. Para guardar el archivo utilice la opción `File/Save As`.

```
 1: #include <pic.h>
 2: #include "delay.h"
 3:
 4: // Configuration Bits (a.k.a. Fuses)
 5: // Estas lineas son opcionales, ya que se pueden fijar externamente
 6: // en MPLAB IDE bajo la opcion de la barra de menu
 7: // <Configure/Configuration Bits...>
 8: // Definiciones:
 9: //  RC/HS/XT/LP: Oscilador RC, High-speed XTAL, XTAL, Low-power XTAL
10: //  WDT: Watchdog Timer  (EN/DIS: Enable, Disable)
11: //  PWRT: Power-up Timer (EN/DIS: Enable, Disable)
12: //  PROTECT: Code write protection
13: __CONFIG(XT&WDTDIS&PWRTDIS&PROTECT);
14: //__CONFIG(RC&WDTEN&PWRTEN&UNPROTECT);
15:
16: //#define XTAL_FREQ 20MHZ
17:
18: #define PORTBIT(adr, bit) ((unsigned)(&adr)*8+(bit))
19:
20: // Variables Globales
21: static volatile bit led @ PORTBIT(PORTB,7);
22: static volatile bit boton = 0;
23:
24: // Funcion Interrupcion
25: static void interrupt isr(void)
26: {
27:     if(INTF){ // -- Flag Interrupcion en pin INT --
28: //          INTE = 0;       // Deshabilita interrupciones en pin INT
29:             boton = 1;      // Boton presionado
30:             INTF = 0;       // Limpia flag de interrupcion INTF
31:             }
32: }
33:
34: // Rutina Parpadeo
35: void blink(void){ // Parpadea el una veces antes de cambiar
36:                   // de estado
37:     char i;
38:     for(i=0;i<5;i++){
39:        DelayMs(125);}
40:     led = !led;
41:     for(i=0;i<5;i++){
42:        DelayMs(255);}
43:     led = !led;
44:     for(i=0;i<5;i++){
45:        DelayMs(125);}
46:     led = !led;
47:     for(i=0;i<5;i++){
48:        DelayMs(255);}
49:     led = !led;
50: }
51:
52: // Rutina Principal
53: void main(void){
54:     led = 1;
55: //  OPTION  = 0b00000111; // Fija PS0-PS2 Pre-escaladores del
56:                          // Watchdog Timer WDTR
57:     TRISB   = 0b00000001; // Fija el pin RB0 en como entrada
58:                          // (en alta impedancia).
59:     INTE = 1; // Habilitacion de interrupciones externas.
60:     GIE  = 1; // Habilitacion global de interrupciones.
```

```
61:     // ei(); // Habilita todos los tipos de interruptciones
62:     // di(); // Deshabilita todas las interrupciones
63:
64:     for(;;){ // -- Loop sin fin --
65:             if(boton){ // Boton presionado
66:                     boton = 0;      // Limpia flag del boton presionado
67:                     led = !led;     // Cambia el estado del led
68:                     DelayUs(10);    // Espera 10 microsegundos
69:                     blink();
70:             //      INTE = 1;               // Habilita interrupciones en pin INT
71:                     }
72:             }
73: }
```

Comentarios Generales del Programa: *En la línea 1 se incluye las definiciones generales para la compilación empleando PICC Lite. Estas definiciones especifican direcciones de memoria de programa, interrupciones, puertos de entrada y salida de cada* μC. *En la línea 2 se incluye el archivo* `delay.h` *específico a este ejemplo con las definiciones de las funciones de retardo* `delayUs()` *y* `delayMs()`. *Estas funciones reciben como argumento un número entero entre 0 y 255 para gener retardos de microsegundos o milisegundos, respectivamente. En el caso de la función* `delayUs()` *se recomienda utilizar valores bastante menores a 255.*

En lenguaje C `//` *indica el inicio de un comentario en una línea. Comentarios de múltiples líneas pueden realizarse colocando* `/*` *al inicio y* `*/` *al final de la sección de comentarios.*

En la línea 13 se utiliza la macro `__CONFIG()` *para fijar los bits de configuración, también conocidos como* fuses *(fusibles). Estos indican al* μC *el tipo de oscilador, típicamente un cristal (XT), la habilitación del* Watchdog Timer, *la habilitación del* Power-up Timer *y la protecccción contra escritura de la memoria. Esta línea es opcional, si no se fija en el programa, los bits de configuración deberán fijarse mediante la opción* `Configure/Configuration Bits...` *de la barra de menú.*

La línea 16 muestra la manera de definir la frecuencia del cristal. Esta opción ya fue agregada a las opciones de compilación, por lo que no es necesaria en el código. Si se quisiese definir en el código, deberá removerse de las opciones de compilación y colocarse antes de los `#include <...>`, *ya que es requerida por las rutinas de retardo especificadas en* `delay.h`.

La línea 18 permite indicar un bit específico de una dirección de memoria `adr` *mediante el texto* `PORT(adr, bit)`. *Como las direcciones de memoria son bytes, el valor de* `bit` *puede estar entre 0 y 7.*

En la línea 21 se utiliza la definición `PORT(adr, bit)` *para asignar el bit 7 del puerto B a la variable* `led`, *es decir la variable* `led` *está asociada a RB7 en el pin 13. La variable* `boton` *se define de manera similar en la línea 22 y se emplea más adelante para indicar si el pulsador ha sido presionado o no. Ambas variables son declaradas como* `static volatile bit`. *El tipo* `bit` *indica que son un bit de un registro. El calificador* `volatile` *se utiliza para indicar al compilador que no está garantizado que la variables retendrán su valor entre llamadas sucesivas. Esto evita que el optimizador elimine múltiples referencias a las variables que de otro modo pudiesen ser consideradas redundantes, ya que eliminarlas podría alterar el comportamiento del programa.* **Todas las variables que están asociadas a puertos (pines) de entrada/salida o que son modificadas en rutinas de interrupción siempre deben ser declaradas** `volatile`. *El calificador* `static` *se emplea para indicar al*

compilador que reserve una posición fija de la RAM para almacenar la variable. Aquellas variables que no son `static` *se crean en posiciones que son asignadas en forma dinámica a medida que se necesitan.*

Entre las líneas 25 y 32 se define la función interrupción `isr(void)` *empleando el calificador* `interrupt`*. El nombre de la función no tiene importancia. La función interrupción es ejecutada cuando ocurre cualquiera de las interrupciones posibles, por ejemplo un* overflow *del timer o un cambio de estado en el pin RB0. La variable INTF (*flag de interrupción*) cambiará de 0 a 1 cuando se produce una interrupción externa en RB0 (pin 4). Otros flags de interrupción son el del timer (*`TOIF`*) y las interrupciones por cambio de estado en alguno de los pines 4, 5, 6 o 7 del puerto B (*`RBIF`*). Todos estos flags, así como los bits de habilitación de interrupciones* `GIE`, `INTE`, `TOIE`, `RBIE`, *son bits del registro* `INTCON`*. Más información sobre las interrupciones y su manejo se pueden encontrar en la hoja de especificaciones del µC PIC16F84 o también en:*
`http://www.mikroe.com/en/books/picbook/2_07chapter.htm`

En el código del programa la función de interrupción se encarga de reconocer el evento colocando nuevamente en 0 el flag de interrupción `INTF` *y colocando la variable* `boton` *en 1 para indicar que el botón fue presionado.*

La desabilitación de la interrupciones se puede realizar fijando `INTE` *en 0 como se muestra en la línea 28. Sin embargo, en la práctica las funciones de retardo hacen innecesaria la línea 28, por esta razón ha sido comentada. El propósito de deshabilitar las interrupciones externas es evitar nuevas interrupciones por rebotes del botón pulsador. El rebote de un botón se refiere a las oscilaciones que se producen antes de que el botón se cierre completamente y la tensión se nivele en su valor final. Este fenómeno se muestra en la figura 7.*

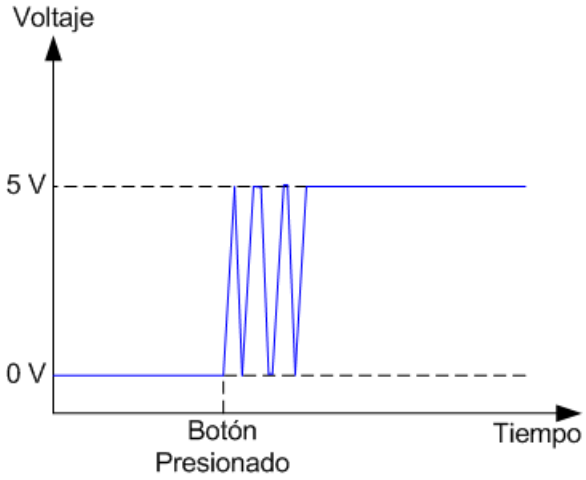

FIGURA 7. Rebote del botón.

Las función `blink(void)` *se utiliza para hacer parpadear el led cambiando su estado repetidas veces por periodos de tiempo controlados por la función* `DelayMs(...)`*.*

La función `main(void)` *contiene parte del programa principal. En primer lugar se enciende el LED colocando su estado en 1 (línea 54). En la línea 57 se fija el pin RB0 como una entrada. El registro TRISB se utiliza para indicar los pines del puerto B que son de alta impedancia (entradas).* **Siempre al iniciar todo programa es importante fijar los bits que serán utilizados como entradas de los puertos**

A y B. Esto se realiza asignando un 1 a los bits correspondientes de los registros `TRISA` **y** `TRISB`. *Luego se habilitan las interrupciones (líneas 59 y 60). Una alternativa es utilizar la función* `ei()` *comentada en la línea 61.*

Por último se inicia un loop *sin fin, el cual permanentemente está detectando el estado de la variable* `boton`*, la cual cambiará a 1 si ocurre una interrupción externa. De ser así, se reconoce que el botón pulsador fue presionado colocando la variable de estado* `boton` *nuevamente en 0. Luego, el estado de la varaible* `led` *se invierte mediante el operador* `!` *de negación, es decir si estaba en 0 se coloca en 1, y si estaba en 1 se coloca en 0. Antes de cambiar de estado finalmente, se hacer parpadear al LED invocando la función* `blink(void)` *creada para dicho propósito. La línea para volver habilitar las interrupciones (línea 70) fue comentada, puesto que estas no son deshabilitadas en ningún momento. Si estas fuese deshabilitadas por la función interrupción en la línea 28, entonces sería necesario volverlas a habilitar en la línea 70 para poder reconocer nuevas interrupciones.*

(10) Agregue el archivo con el código en lenguaje C (creado en el paso anterior al proyecto) empleando la opción `Project/Add Files to Project...`, y seleccione el archivo `ej2.c` creado en el paso anterior. Verifique que el tipo de archivo esta fijado en `Source (*.c, *.as)` y que la opción `Auto` está seleccionada. Presione `Open`, esto debería agregar el archivo al a sección `Source Files` en la ventana con la lista de archivos del proyecto. Otra manera de agregar un archivo es seleccionando con el botón derecho del mouse la sección deseada en la lista de archivo del proyecto. En este caso, la sección sería `Source Files`. Del menú desplegado elija la opción `Add Files` y siga los pasos ya explicados en este punto para seleccionar el archivo.

(11) De la lista de archivos del proyecto, seleccione con el botón derecho del mouse el título del proyecto, por ejemplo `ej2.mcb`. Del menú desplegado elija la opción `Save` para grabar el proyecto.

4.4. **Paso 3: Compilación del Proyecto con PICC Lite.**

(1) Seleccione la opción `Project/Build Options.../Project`. Luego seleccione la sección `Compiler` y agregue la siguiente definición de macro:

```
XTAL_FREQ=20MHZ
```

Esto agregará la opción de compilación `-DXTRAL_FREQ=20MHZ` (ver el paso siguiente). Alternativamente puede definir la macro con el pragma:

```
#define XTAL_FREQ 20MHZ
```

antes de colocar las líneas `#include <...>#` en el código del programa.

La definición de esta macro es necesaria para ajustar el ciclo de operación de las rutinas de retardo a la frecuencia del cristal oscilador.

(2) Existen dos opciones para compilar el programa:

- **MPLAB IDE**: Para compilar el proyecto seleccione la opción de la barra de menú `Project/Build` o presione las teclas `CTRL+F10`. Se abrirá una ventana con el nombre `Output`, la cual mostrará el estado de la compilación.
- **Línea de Comando**: Para compilar el proyecto para la version actual del PIC16F84, el PIC16F84A, con un cristal de 20 MHz desde la línea de comando en una ventana DOS ejecute las siguientes instrucciones:

```
picl --chip=16F84A -DXTAL_FREQ=20MHZ -C delay.c
picl --chip=16F84A -DXTAL_FREQ=20MHZ -C ej2.c
picl --chip=16F84A -DXTAL_FREQ=20MHZ -oej2.cof
```

```
     -mej2.map delay.obj ej2.obj
```

Alternativamente puede ejecutar las instrucciones con más opciones, por ejemplo, fijando el formato de los mensajes de error y advertencias, la generación de código Assembly, definiendo el tipo de dato para los caracteres, etc., como se muestra a continuación.

```
picl -q -g --asmlist --chip=16F84A
     "--errformat=Error   [%n] %f; %l.%c %s"
     "--msgformat=Advisory[%n] %s"
     "--warnformat=Warning [%n] %f; %l.%c %s"
     -P --char=unsigned
     -DXTAL_FREQ=20MHZ -C delay.c

picl -q -g --asmlist --chip=16F84A
     "--errformat=Error   [%n] %f; %l.%c %s"
     "--msgformat=Advisory[%n] %s"
     "--warnformat=Warning [%n] %f; %l.%c %s"
     -P --char=unsigned
     -DXTAL_FREQ=20MHZ -C ej2.c

picl -q -g --asmlist --chip=16F84A
     "--errformat=Error   [%n] %f; %l.%c %s"
     "--msgformat=Advisory[%n] %s"
     "--warnformat=Warning [%n] %f; %l.%c %s"
     -P --char=unsigned
     -DXTAL_FREQ=20MHZ -oej2.cof -mej2.map delay.obj ej2.obj
```

Es importante considerar que las líneas anteriores deben ser ejecutadas en una sola línea. Por razones de espacio estas fueron divididas en tres líneas en este documento.

En ambos casos la compilación se ejecutar por partes, primero de los archivos con las definiciones para las funciones de retardo `delayUs()` y `delayMs()`. Luego se compila el código principal del ejemplo. Finalmente, se realiza el proceso de unión (*linking*) de los códigos objetos `delay.obj` y `ej2.obj` para generar el archivo `ej2.hex` con el código de máquina para el μC. En todos los casos la compilación se realiza con todas las optimizaciones (espacio, velocidad, Assembler) activas. Es importante que estas estén activas para este ejemplo, ya que las funciones de retardo las requieren. En otros casos, es posible desactivar o controlar el nivel de optimizaciones con la opción `--OPT<=type>`, donde `<=type>` puede tomar distintos valores que se especifican en el manual del compilador PICC Lite.

4.5. **Paso 4: Programación del μC.**

Puede programar el μC según lo explicado en las secciones 2.3, 2.3.1, 2.3.2 del presente documento, o bien usando el MPLAB IDE si posee algún programador Microchip como el PICSTART Plus. A continación se explican los pasos de programación desde MPLAB IDE asumiendo que ha realizado los pasos descritos en la etapa de creación del proyecto y compilación. Si el archivo `.hex` fue creado usando el PICC mediante compilación fuera del ambiente MPLAB IDE, deberá entonces realizar los dos primeros pasos de la sección 3.5.

(1) Verifique las opciones de configuración de MPLAB IDE. En particular verifique en la opción `Configure/Select Device...` de la barra de menú que el dispositivo corresponde al μC PIC16F84 o 16F84A. Verifique también que los bits de configuración de la opción `Configure/Configuration Bits...` de la barra de menú son: Oscillator = XT, Watchdog Timer = Off, Power Up Timer = Off, Code Protect = On.
(2) Seleccione el hardware programador, por ejemplo el PICSTART Plus, empleando la opción `Programmer/Select Programmer` de la barra de menú.
(3) Borre la memoría del μC presionando el icono de página en blanco a μC o la opción `Programmer/Erase Flash Device` de la barra de menú.
(4) Programe el dispositivo presionando el icono de página llena (amarilla) a μC o la opción `Programmer/Program` de la barra de menú.
(5) Normalmente el programador entregará un mensaje indicando si la programación fue exitosa. Sin embargo, si desea verificar si efectivamente la programación se realizó en forma correcta, emplee el botón de verificación (página con check) o la opción `Programmer/Verify` de la barra de menú. También puede ver los contenidos de la memoria del μC utilizando la opción `View/Program Memory` de la barra de menú. Si la programación se realizó en forma correcta tanto el código compilado como el código leido de vuelta del μC deben tener el mismo *checksum* que se muestra en la barra de menú del MPLAB IDE.

4.6. **Circuito del Ejemplo 2.**

En la figura 8 se muestra el circuito para el Ejemplo 2. Como se puede apreciar en la figura, el circuito de este ejemplo es muy similar al del Ejemplo 1. La única modificacion al circuito del ejemplo anterior es la adición de un botón pulsador B1 entre 5 V y RB0/INT con una resistencia *Pull-Down* R3. La resistencia *Pull-Down* mantiene el puerto en LOW (0), mientras el pulsador no sea presionado y cambie la señal a HIGH (1). Si se desea que la interrupción se genere al soltar el botón, entonces deberán intercambiarse la resistencia *Pull-Down* por el botón pulsador.

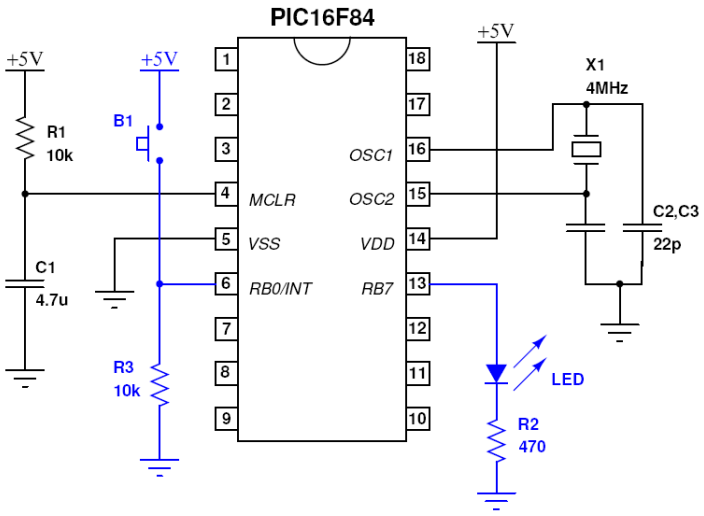

FIGURA 8. Circuito del ejemplo 2.

5. Desarrollos más allá de este Documento

Existe una variedad de μCs y posibles aplicaciones tan grande que puede parecer abrumadora al punto de dificultar o confundir la decisión de cómo abordar nuevos temas y cual es el mejor camino a seguir. En esta sección se sugieren algunos caminos para involucrarse en el desarrollo de aplicaciones más complejas.

Para dominar el PIC16F84 u otros μCs es esencial conocer y comprender a cabalidad la hoja de datos del μC provista por el fabricante. Microchip provee también de documentos llamados *Application Notes* (Notas de Aplicación), los cuales describen técnicas espcíficas, aplicaciones y su implementación con un gran nivel de detalles. Estos documentos son muy ilustrativos, fáciles de comprender y mantienen la rigurosidad técnica de los fabricantes. Por otro lado, existe una gran cantidad de ejemplos y recursos en libros y la Internet. Consulte las númerosas referencias incluidas al final de este documento. Se recomienda en particular leer y hacer los ejemplos presentados en [11]. En la sección de educación de Microchip también es posible encontrar referencias de varios libros sobre los μCs PIC [23].

Las aplicaciones posibles del PIC16F84 solo están limitadas por la imaginación. En Internet pueden encontrarse aplicaciones sorprendentemente creativas a pesar de la simplicidad y restricciones de este μC básico.

5.1. **PIC16F87x.**

El PIC16F87x es el más popular para aplicaciones de nivel intermedio, y es una excelente opción cuando se requieren más líneas de entrada/salida, convertidores A/D, señales PWM, comunicación serial por hardware, entre otros aspectos. Existen versiones con distintas características como el tamaño de la memoria y número de pines.

Una vez que se ha dominado adecuadamente el PIC16F84, adquirir los conocimientos para utilizar este nuevo PIC es considerablemente más fácil. El costo de la versión PIC16F877, que es la que cuenta con más características de la serie, es aproximadamente el doble que el del PIC16F84 con valores entorno a USD $10 versus USD $ 5.

5.2. **Freescale MC68HC08.**

Otros fabricantes pueden tener productos considerablemente diferentes, tanto en su funcionamiento, instrucciones de CPU o método de programación. Sin embargo, los μCs de nivel de entrada (*entry-level*) tienen mucho en común. Este es el caso de los μCs Freescale MC68HC08 (originalmente Motorola HC08). Estos fueron introducidos al mercado para competir con los μCs de Microchip. A pesar que actualmente la popularidad y disponibilidad de herramientas de software para estos μCs es aún menor que la de los μCs PIC, esta familia de dispositivos tiene ciertos atractivos, como un menor costo frente al PIC16F84, convertidores A/D en casi todas sus versiones, señales PWM, oscilador interno, y la disponibilidad gratuita del popular entorno de desarrollo CodeWarrior. Además están disponibles en encapsulados de 8 y 16 pines. Para mayor información consulte la sección de microcontroladores de Freescale (`http://www.freescale.com/`).

Agradecimientos

Este tutorial fue posible gracias a una versión anterior del tutorial preparad por Juan Pablo Caram en Octubre del 2003 para el curso IEE2712 Laboratorio de Circuitos del Dpto. de Ingeniería Eléctrica de la Universidad Católica de Chile.

Apéndice A. Lista de Componentes del Programador

2 Resistencias de 4.7 kΩ (valor no crítico).
1 Resistencia de 10 kΩ (valor no crítico).
1 Condensador electrolítico de 100 μF.
1 Regulador de voltaje 7805.
1 Diodo 1N4148 u otro diodo rectificador de señal.
1 Conector DB9 Hembra para PCB en 90°.
1 Base DIP18.

Apéndice B. Lista de Componentes para los Ejemplos

2 Resistencias de 10 kΩ (valor no crítico).
1 Resistencia de 470 Ω (valor no crítico).
2 Condensadores de 22 pF.
1 Condensador de 4.7 μF (valor no critico).
1 Cristal oscilador de 20 MHz (alternativamente 4 MHz, o 10 MHz).
1 Botón pulsador "siempre abierto" (normalmente abierto).

Apéndice C. Configuración del PROG84

El archivo de configuración del PROG84 se denomina `lp_cfg`. Este debe modificarse para seleccionar el puerto serial a utilizar y el tipo de hardware programador. Para el programador COM84 lo esencial es configurar el puero serial que se desea utilizar (líneas 4 a 7) según se muestra en el siguiente ejemplo:

```
 1: port serial
 2:
 3: ### for serial port:
 4: base= 0x3f8 # com1, ttyS0
 5: #base= 0x2f8 # com2, ttyS1
 6: #base= 0x3e8 # com3, ttyS2
 7: #base= 0x2e8 # com4, ttyS3
 8:
 9: # settings for UniprogIV / BR870
10: # and similar seriel PIC programmers.
11: no_power_C4C8 = 0
12:
13: power: TxD
14: mclr: TxD
15: data: DTR
16: data_f: CTS
17: clock: RTS
```

Referencias

REFERENCIAS BASICAS

[1] Microchip Technology, Inc.
`http://www.microchip.com/`
Sitio del fabricante de los microcontroladores PIC. Aqui encontrará las hojas de datos y Application Notes.

[2] Gnu PIC Utilities (`gputils`). `http://gputils.sourceforge.net/`
Conjunto de herramientas para microcontroladores Microchip PIC, que incluyen compiladores, ensambladores, y simuladores.

[3] HI-TECH Software LLC.
`http://www.htsoft.com/`
HITECH Software LLC. produce el compilador PICC Lite para C. Este puede obtenerse en forma gratuita de su sitio web. La versión completa de este producto es uno de los compiladores más poderosos existentes para microcontroladores PIC.

[4] Jal, por Wouter van Ooijen. `http://jal.sourceforge.net/`
Sitio de la comunidad de desarrollo de JAL. Aquí podrá obtener JAL para Linux/BSD, Windows o MacOS X, toda la documentación asociada y acceder a numerosos ejemplos.

[5] CCS, Inc.
`http://www.ccsinfo.com/`

[6] GNUPIC.
`http://www.gnupic.org/`
Recopilación de recursos para el manejo y desarrollo de aplicaciones con PICs para Linux/BSD. Incluye links a compiladores, ensambladores, desensambladores, simuladores, programadores y otras herramientas. En la sección programadores encontrará un link para bajar el Prog84.

[7] IC-Prog Prototype Programmer.
`http://www.ic-prog.com/`

INTRODUCCION A LOS MICROCONTROLADORES PIC

[8] Wouter van Ooijen. *Starting with PIC microcontrollers.*
`http://www.voti.nl/swp/`
Introducción sencilla y completa, un buen complemento al presente tutorial.

[9] David Tait. Programador de PICs e información relacionada.
`http://people.man.ac.uk/~mbhstdj/piclinks.html`

[10] *Eric's PIC Page.*
`http://www.brouhaha.com/~eric/pic/`

[11] mikroElektronika : books : PIC microcontrollers:
`http://www.mikroe.com/en/books/picbook/picbook.htm`
Este sitio contiene una excelente introducción a los microcontroladores PIC. El sitio presente en forma resumida los capítulos del libro:
The PIC microcontroller, por Nebojsa Matic y Dragan Andric, mikroElectronica, 3^a ed., May, 2000.
Los capítulos del libro abordan la introducción general a los microcontroladores, el lenguaje de ensamblador, MLAB IDE, y varios ejemplos.

TUTORIALES

[12] `http://www.epemag.wimborne.co.uk/pictutorial.pdf`
Tutorial Descriptivo del PIC16F87x por John Becker. Este es útil como referencia pero no es el mejor ejemplo para empezar.

[13] `http://www.microchipc.com/conference/`
Tutorial Descriptivo del PIC18Fxxxx. Util como referencia general pero no para empezar.

[14] PIC Pages `http://www.botkin.org/dale/PIC.htm`
Contiene ejemplos sencillos de aplicación.

EJEMPLOS DE APLICACION

[15] J. Charais, R. Lourens. Software PID Control of an Inverted Pendulum Using the PIC 16F684. *Application Note AN 964*, Microchip Technology Inc.
http://ww1.microchip.com/downloads/en/AppNotes/00964A.pdf
Interesante ejemplo de aplicación de microcontroladores al control de motores.

[16] Pablin Electrónica. Proyectos y circuitos de ejemplo.
http://www.pablin.com.ar/electron/
http://www.pablin.com.ar/electron/download/index.htm (*Software para electrónica.*)
http://www.pablin.com.ar/electron/proyecto/picnet/index.htm
Entradas y Salidas de un PIC controladas por Internet – Excelente ejemplo de una aplicación sencilla, pero de gran utilidad práctica. El proyecto original es de Wichit Sirichote, ver referencia más abajo.

[17] Build Your Own Microcontroller Projects:
http://www.kmitl.ac.th/~kswichit%20/
http://www.kmitl.ac.th/~kswichit%20/easyserver0.9/easyserver0.9.htm
REMOTE[RS232] Using Easy-Server V0.9 – Excelente ejemplo de una aplicación sencialla, pero de gran utilidad práctica. Este proyecto también se encuentra traducido al español en la página web de Pablin; ver referencia anterior.

[18] Peter Anderson's PIC Page:
http://www.phanderson.com/PIC/
Presenta ejemplos con interrupciones, rutinas aritméticas, interfaces serial, entre otros.

[19] Paul Hills, Using microcontrollers in your robot:
http://homepages.which.net/~paul.hills/Embedded/Embedded.html

BOOTLOADERS

[20] PIC micro and C bootloader by Shane Tolmie:
http://www.microchipc.com/PIC16bootload/
http://www.microchipc.com/PIC18bootload/

REFERENCIAS ADICIONALES

[21] IEE2172 Laboratorio de Circuitos UC http://www2.ing.puc.cl/~iee2172/?q=node/2

[22] Links about Electronics Microprocessors, Programmable Components, PICs by Hobby-Electronics:
http://www.hobby-electronics.info/links/icup.php?PHPSESSID=2aa545e794080ff3d135ba1b9caa6b4a#pic

[23] Microchip Technology's University Corner:
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1441
Este sitio contiene una extensa lista de libros publicados por diversos autores sobre los microcontroladores PIC. La lista también incluye datos sobre el idioma, nivel y tipo de microcontrolador que se emplea en el libro.

SOFTWARE PARA SIMULACION DE CIRCUITOS

[24] Microcap:
http://www.spectrum-soft.com/
Software de simulación de circuitos analógicos y digitales que permite simular los circuitos a partir de su descripción en base a esquemáticos. Esta basado en Spice y permite realizar análisis transiente, DC y AC.

[25] Multisim:
http://www.ni.com/multisim/
Originalmente llamado Electronics Workbench*, es muy similar a Microcap, también basado en SPICE. Actualmente es desarrollado por National Instruments y ha incorporado otras herramientas, como el desarrollo de PCBs a partir de los esquemáticos o herramientas para facilitar la enseñanza y aprendizaje de electrónica.*

SOFTWARE PARA DISEÑO DE PCBs y ESQUEMATICOS

[26] Eagle:
`http://www.cadsoft.de/`
Excelente software para el diseño de PCBs y dibujo de esquemáticos que funciona bajo Linux/BSD o Windows. Tiene una librería de componentes muy completa y es muy fácil agregar componentes nuevos. Permite generar los PCBs directamente a partir de los esquemáticos. `http://www.cadsoft.de/`

Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Vicuña Mackenna 4860, Santiago, Chile, *Tel.* : 56 (2) 354-2000