



# **PICDEM™ Lab Flowcode Companion Guide**

---

**Note the following details of the code protection feature on Microchip devices:**

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

---

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

**Trademarks**

The Microchip name and logo, the Microchip logo, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, rPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Octopus, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICKit, PICtail, PIC<sup>32</sup> logo, REAL ICE, rLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2009, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

**QUALITY MANAGEMENT SYSTEM**  
**CERTIFIED BY DNV**  
**== ISO/TS 16949:2002 ==**

*Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC<sup>®</sup> MCUs and dsPIC<sup>®</sup> DSCs, KEELOQ<sup>®</sup> code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.*

**Table of Contents**

**Chapter 1. Getting Started**

1.1 Introduction ..... 9  
 1.2 Highlights ..... 9  
 1.3 Flowcode 3 – Free Version installation ..... 9  
 1.4 Installing the PICDEM™ Lab Development Kit Labs (Flowcode Version) ... 15

**Chapter 2. General Purpose Input/Output Labs**

2.1 Introduction ..... 17  
 2.2 General Purpose Input/Output Labs ..... 17  
 2.3 GPIO Output Labs ..... 17  
   2.3.1 Reference Resources ..... 17  
   2.3.2 Equipment Required for GPIO Output Labs ..... 18  
   2.3.3 PICDEM Lab Development Board Setup for GPIO Output Labs ..... 18  
   2.3.4 Lab 1: Creating a Project in Flowcode to Light LEDs ..... 19  
     2.3.4.1 New Registers Used in This Lab ..... 19  
     2.3.4.2 Overview ..... 19  
     2.3.4.3 Procedure ..... 19  
     2.3.4.4 Testing the Application ..... 29  
   2.3.5 Lab 2: Introducing the Software Control Loop to Flash LEDs  
     (Delay Loop) ..... 30  
     2.3.5.1 Overview ..... 30  
     2.3.5.2 Procedure ..... 31  
     2.3.5.3 Testing the Application ..... 39  
   2.3.6 Lab 3: Using Macros to Rotate LEDs ..... 40  
     2.3.6.1 Overview ..... 40  
     2.3.6.2 Procedure ..... 40  
     2.3.6.3 Testing the Application ..... 48  
 2.4 GPIO Input Labs ..... 49  
   2.4.1 Reference Documentation ..... 49  
   2.4.2 Equipment Required for GPIO Input Labs ..... 49  
   2.4.3 PICDEM Lab Development Board Setup for GPIO Input Labs ..... 49  
   2.4.4 Lab 4: Adding a Push Button Interface using Component Macros ..... 50  
     2.4.4.1 New Registers Used in This Lab ..... 50  
     2.4.4.2 Overview ..... 50  
     2.4.4.3 Procedure ..... 51  
     2.4.4.4 Testing the Application ..... 60  
   2.4.5 Lab 5: Adding an Interrupt and Importing Macros for a Push Button  
     Interface ..... 60  
     2.4.5.1 New Registers Used in This Lab ..... 60  
     2.4.5.2 Overview ..... 60  
     2.4.5.3 Procedure ..... 61  
     2.4.5.4 Testing the Application ..... 68

## Chapter 3. Comparator Peripheral

3.1 Introduction .....	69
3.2 Comparator Labs .....	69
3.2.1 Reference Documentation .....	69
3.2.2 Comparator Labs .....	69
3.2.3 Equipment Required .....	69
3.2.4 Lab 1: Simple Compare .....	70
3.2.4.1 New Registers Used in This Lab .....	70
3.2.4.2 Overview .....	70
3.2.4.3 Procedure .....	71
3.2.4.4 Testing the Application .....	75
3.2.5 Lab 2: Using the Comparator Voltage Reference .....	76
3.2.5.1 New Registers Used in This Lab .....	76
3.2.5.2 Overview .....	76
3.2.5.3 Procedure .....	77
3.2.5.4 Testing the Application .....	78
3.2.6 Lab 3 Higher Resolution Sensor Readings Using a Single Comparator ....	79
3.2.6.1 New Registers Used in This Lab .....	79
3.2.6.2 Overview .....	79
3.2.6.3 Procedure .....	82
3.2.6.4 Testing the Application .....	91

## Chapter 4. Analog-to-Digital Converter Peripheral Labs

4.1 Introduction .....	93
4.2 ADC Labs .....	94
4.2.1 Reference Documentation .....	94
4.2.2 Equipment Required .....	94
4.2.3 Lab 1: Simple ADC .....	95
4.2.3.1 New Registers Used in This Lab .....	95
4.2.3.2 Overview .....	95
4.2.3.3 Procedure .....	96
4.2.3.4 Testing the Application .....	101
4.2.4 Lab 2: Audible Temperature Sensor .....	101
4.2.4.1 Overview .....	101
4.2.4.2 Procedure .....	102
4.2.4.3 Testing the Application .....	110

---

---

## Preface

---

---

### NOTICE TO CUSTOMERS

All documentation becomes dated, and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site ([www.microchip.com](http://www.microchip.com)) to obtain the latest documentation available.

Documents are identified with a “DS” number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is “DSXXXXA”, where “XXXX” is the document number and “A” is the revision level of the document.

For the most up-to-date information on development tools, see the MPLAB® IDE on-line help. Select the Help menu, and then Topics to open a list of available on-line help files.

## INTRODUCTION

This chapter contains general information that will be useful to know before using the PICDEM™ Lab Flowcode Companion Guide. Items discussed in this chapter include:

- Document Layout
- Conventions Used in this Guide
- Warranty Registration
- The Microchip Web Site
- Development Systems Customer Change Notification Service
- Customer Support
- Document Revision History

## DOCUMENT LAYOUT

This document describes how to use the PICDEM™ Lab Flowcode Companion Guide as a development tool to emulate and debug firmware on a target board. The manual layout is as follows:

- **Chapter 1. “Getting Started”**
- **Chapter 2. “General Purpose Input/Output Labs”**
- **Chapter 3. “Comparator Peripheral”**
- **Chapter 4. “Analog-to-Digital Converter Peripheral Labs”**

# PICDEM™ Lab Flowcode Companion Guide

## CONVENTIONS USED IN THIS GUIDE

This manual uses the following documentation conventions:

### DOCUMENTATION CONVENTIONS

Description	Represents	Examples
<b>Arial font:</b>		
Italic characters	Referenced books	<i>MPLAB® IDE User's Guide</i>
	Emphasized text	...is the <i>only</i> compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic text with right angle bracket	A menu path	<u><i>File&gt;Save</i></u>
Bold characters	A dialog button	Click <b>OK</b>
	A tab	Click the <b>Power</b> tab
N'Rnnnn	A number in verilog format, where N is the total number of digits, R is the radix and n is a digit.	4'b0010, 2'hF1
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
<b>Courier New font:</b>		
Plain Courier New	Sample source code	#define START
	Filenames	autoexec.bat
	File paths	c:\mcc18\h
	Keywords	_asm, _endasm, static
	Command-line options	-Opa+, -Opa-
	Bit values	0, 1
	Constants	0xFF, 'A'
Italic Courier New	A variable argument	<i>file.o</i> , where <i>file</i> can be any valid filename
Square brackets [ ]	Optional arguments	mcc18 [options] <i>file</i> [options]
Curly brackets and pipe character: {   }	Choice of mutually exclusive arguments; an OR selection	errorlevel {0 1}
Ellipses...	Replaces repeated text	var_name [, var_name...]
	Represents code supplied by user	void main (void) { ... }

## WARRANTY REGISTRATION

Please complete the enclosed Warranty Registration Card and mail it promptly. Sending in the Warranty Registration Card entitles users to receive new product updates. Interim software releases are available at the Microchip web site.

## THE MICROCHIP WEB SITE

Microchip provides online support via our web site at [www.microchip.com](http://www.microchip.com). This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

## DEVELOPMENT SYSTEMS CUSTOMER CHANGE NOTIFICATION SERVICE

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at [www.microchip.com](http://www.microchip.com), click on Customer Change Notification and follow the registration instructions.

The Development Systems product group categories are:

- **Compilers** – The latest information on Microchip C compilers and other language tools. These include the MPLAB C18 and MPLAB C30 C compilers; MPASM™ and MPLAB ASM30 assemblers; MPLINK™ and MPLAB LINK30 object linkers; and MPLIB™ and MPLAB LIB30 object librarians.
- **Emulators** – The latest information on Microchip in-circuit emulators. This includes the MPLAB ICE 2000 and MPLAB ICE 4000.
- **In-Circuit Debuggers** – The latest information on the Microchip in-circuit debugger, MPLAB ICD 2.
- **MPLAB® IDE** – The latest information on Microchip MPLAB IDE, the Windows® Integrated Development Environment for development systems tools. This list is focused on the MPLAB IDE, MPLAB SIM simulator, MPLAB IDE Project Manager and general editing and debugging features.
- **Programmers** – The latest information on Microchip programmers. These include the MPLAB PM3 and PRO MATE® II device programmers and the PICSTART® Plus and PICKIT™ 1 development programmers.

# PICDEM™ Lab Flowcode Companion Guide

---

## CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or field application engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: <http://support.microchip.com>

## DOCUMENT REVISION HISTORY

### **Revision A (April 2009)**

### **Revision B (August 2009)**

Replaced “infinity” with “infinite” throughout document.



---

---

## Chapter 1. Getting Started

---

---

### 1.1 INTRODUCTION

The contents of this guide are a companion to the PICDEM™ Lab Development Kit User's Guide (DS41369). The reader is encouraged to download a free copy of the main guide at [www.microchip.com/picdemlab](http://www.microchip.com/picdemlab) for reference while completing these labs for information on such topics as:

- PICDEM Lab Development Kit Contents
- PICDEM Lab Development Board Construction and Layout
- Target Power
- PICDEM Lab Development Board Schematics

This chapter is intended to prepare the reader to complete the labs in this user's guide.

### 1.2 HIGHLIGHTS

This chapter discusses:

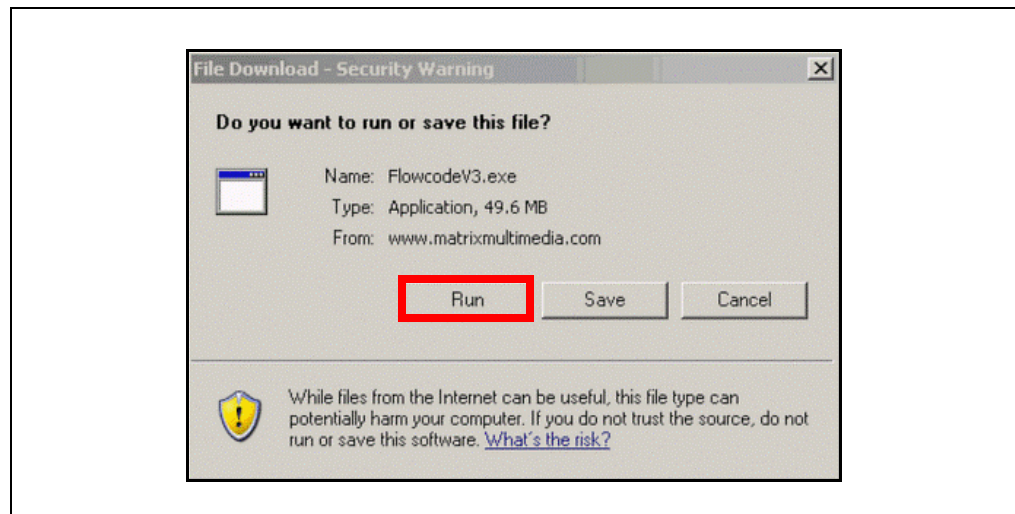
- Flowcode 3 – free version Installation
- Installing the PICDEM Lab Development Kit Labs (Flowcode Version)

### 1.3 FLOWCODE 3 – FREE VERSION INSTALLATION

The following steps will demonstrate how to install the Flowcode 3 – free version software.

1. Navigate to the PICDEM Lab Development Kit Homepage at: [www.microchip.com/picdemlab](http://www.microchip.com/picdemlab) and scroll down to the Install Flowcode V3 – free version link. Click on the link to begin the installation process.
2. In the File Download window, select **Run** (Figure 1-1).

**FIGURE 1-1: FILE DOWNLOAD WINDOW**



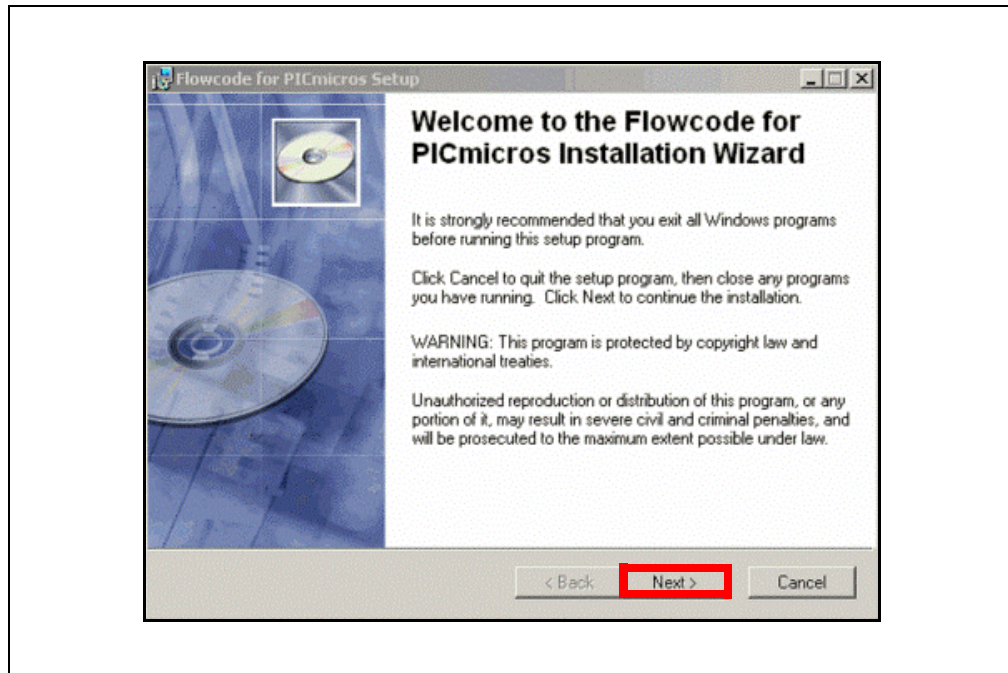
3. If a Security window opens click **Run** (Figure 1-2).

**FIGURE 1-2: SECURITY WINDOW**



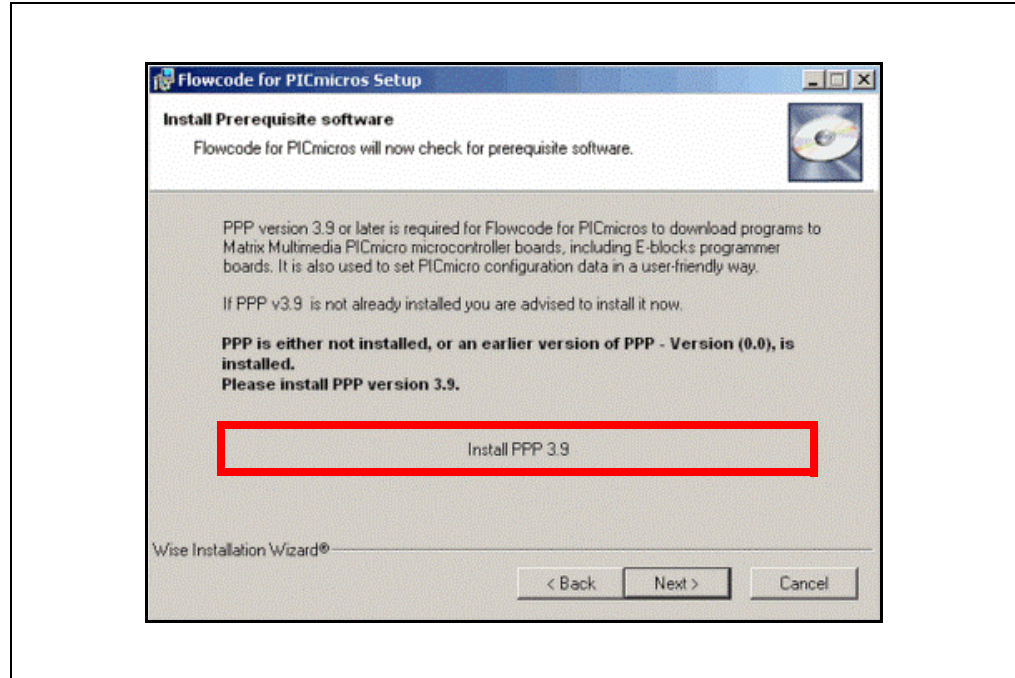
4. In the Welcome window click **Next>** (Figure 1-3).

**FIGURE 1-3: FLOWCODE WELCOME WINDOW**



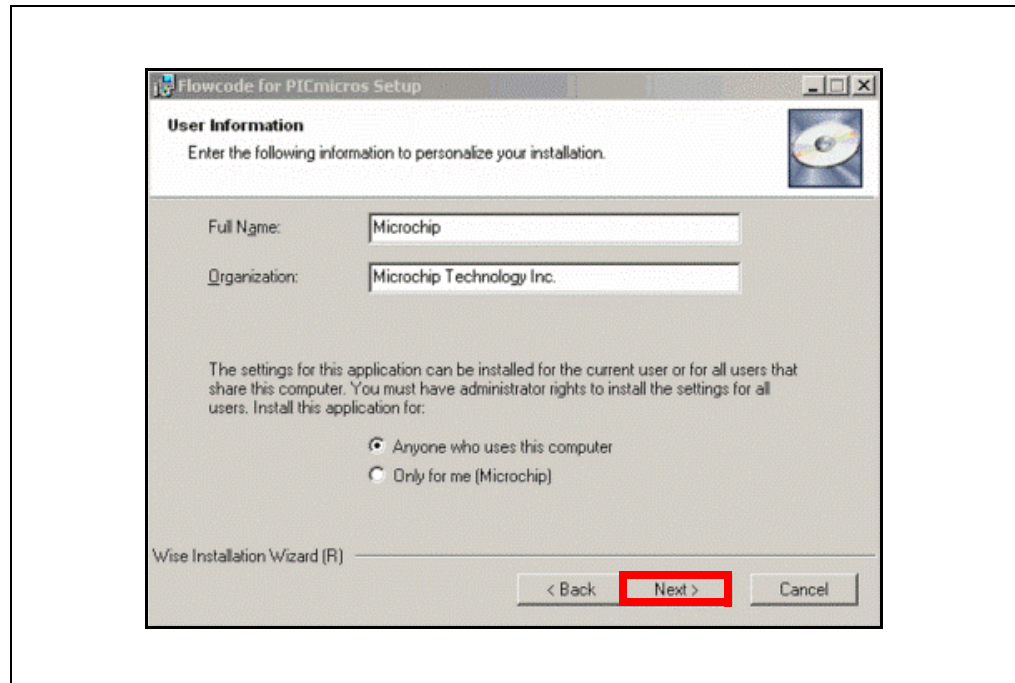
5. Next, the Install Prerequisite Software window opens providing the option to download the latest version of the PIC® MCU Parallel Programming (PPP) software. PPP provides a convenient way of selecting the Configuration data for PIC® devices while enabling programming via the PICkit™ 2 Programmer/Debugger directly from the Flowcode environment. Alternately, the PICkit 2 Programming software could be used. However, for convenience, PPP is recommended to be installed. Click the **Install PPP x.x** button to open a separate installer and follow the prompts (Figure 1-4).

**FIGURE 1-4: INSTALL PREREQUISITE SOFTWARE WINDOW**



6. Following installation of the PPP software, in the User Information window, complete as required and select **Next>** to continue (Figure 1-5).

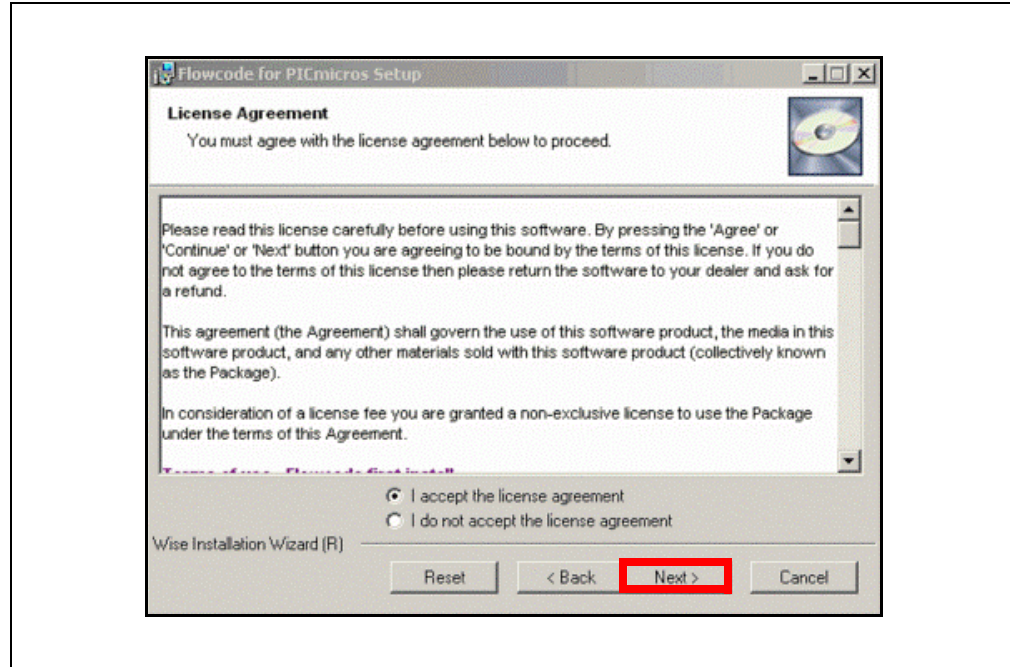
**FIGURE 1-5: USER INFORMATION WINDOW**



7. In the License Agreement window, ensure that the **I accept the license agreement** radio button is selected to continue with the installation and click **Next>** (Figure 1-6).

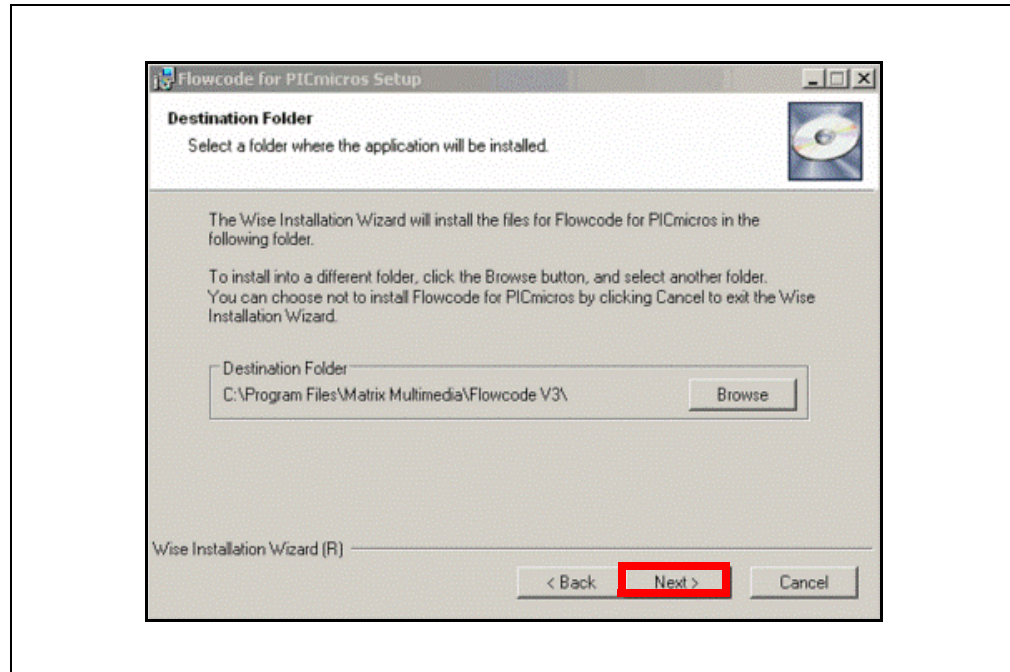


FIGURE 1-6: LICENSE AGREEMENT WINDOW



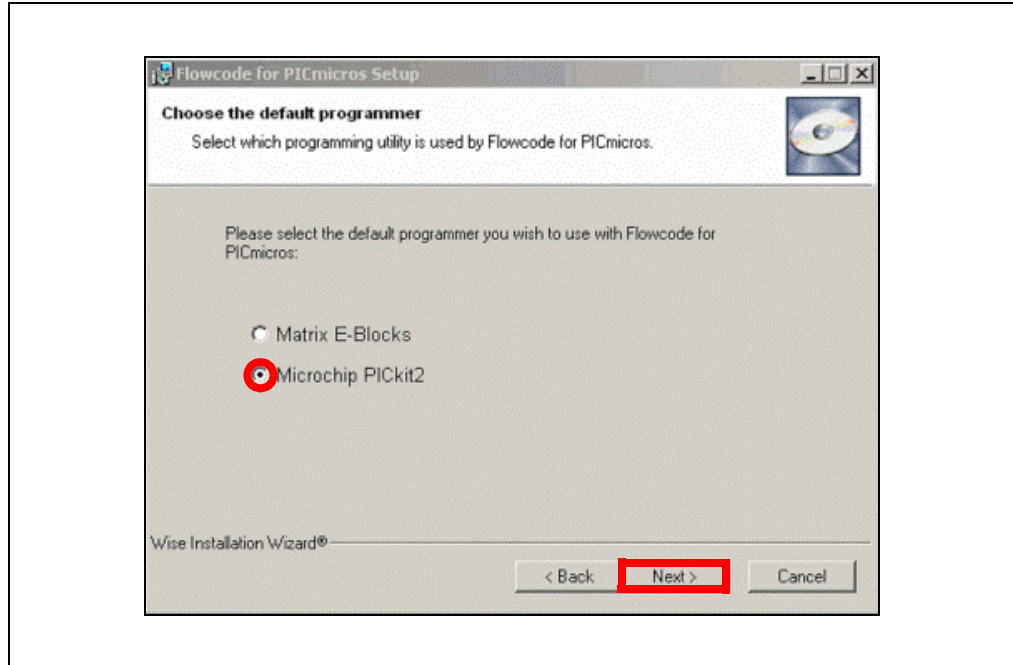
8. The Destination Folder window is used to provide a location to download the Flowcode software files. It is strongly recommended that the default destination is used. Click **Next>** to continue (Figure 1-7).

FIGURE 1-7: DESTINATION FOLDER WINDOW



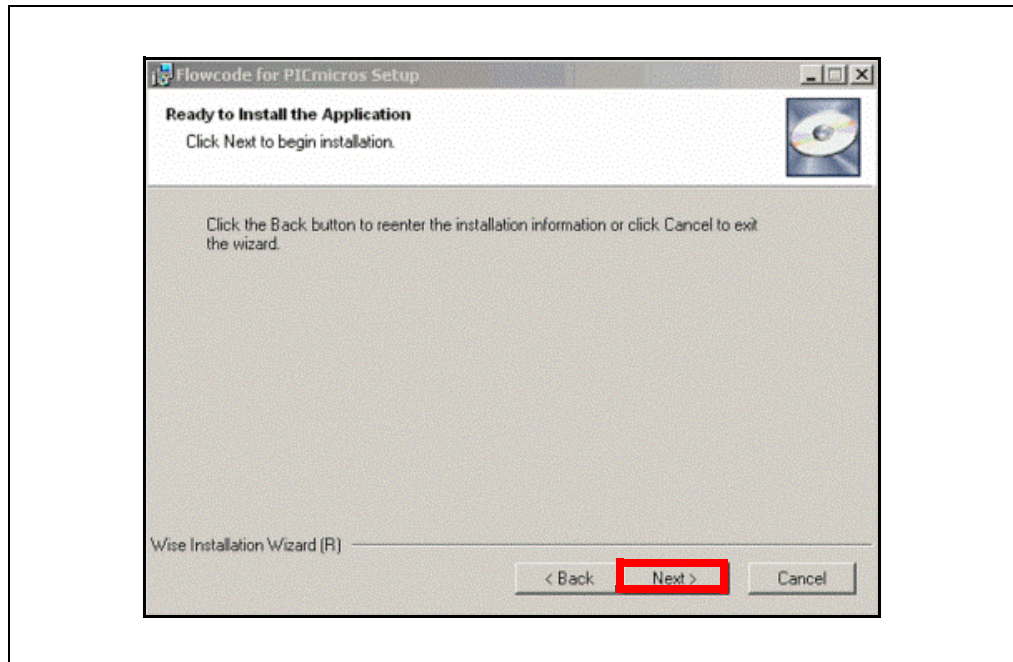
9. To download projects developed in the Flowcode environment to the PIC16F690 microcontroller populating the PICDEM Lab Development Board (socket U2), the provided PICKit 2 Programmer/Debugger will be used. In the Choose the Default Programmer window, be sure to select the **Microchip PICKit 2** radio button to select the PICKit 2 Programmer/Debugger as the default programmer and click **Next>** to continue (Figure 1-8).

**FIGURE 1-8: CHOOSE THE DEFAULT PROGRAMMER WINDOW**



10. Click **Next>** in the Ready to Install the Application window to continue with the installation (Figure 1-9).

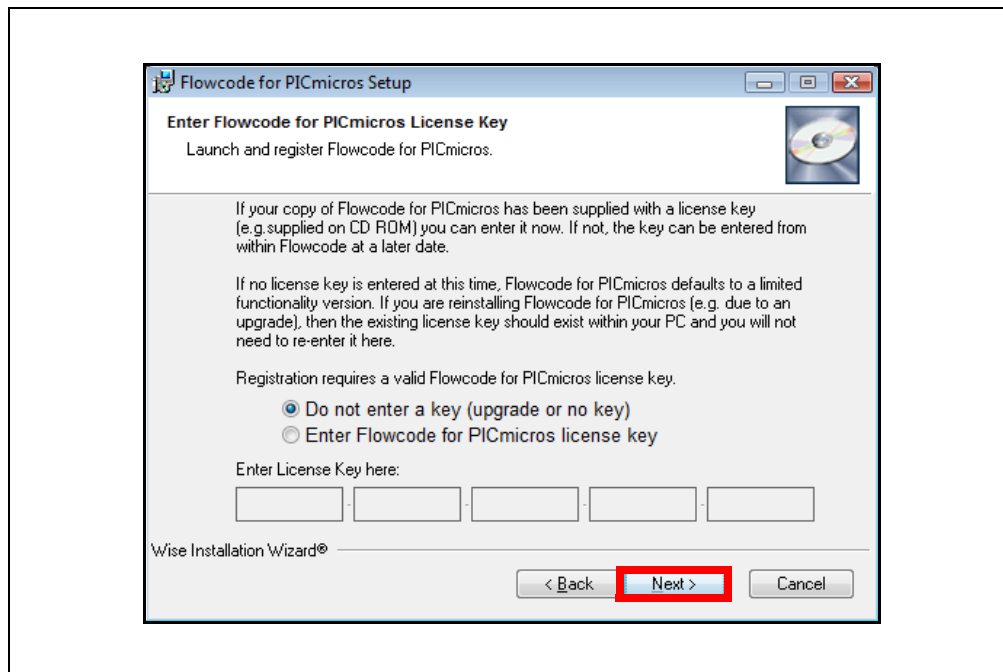
**FIGURE 1-9: READY TO INSTALL APPLICATION WINDOW**



The install process should begin. This may take a few minutes.

11. In the Enter Flowcode for PIC<sup>®</sup> MCU License Key window, select “Do not enter a key (upgrade or no key)” to use the free version of the software and click **Next>** to continue (Figure 1-10).

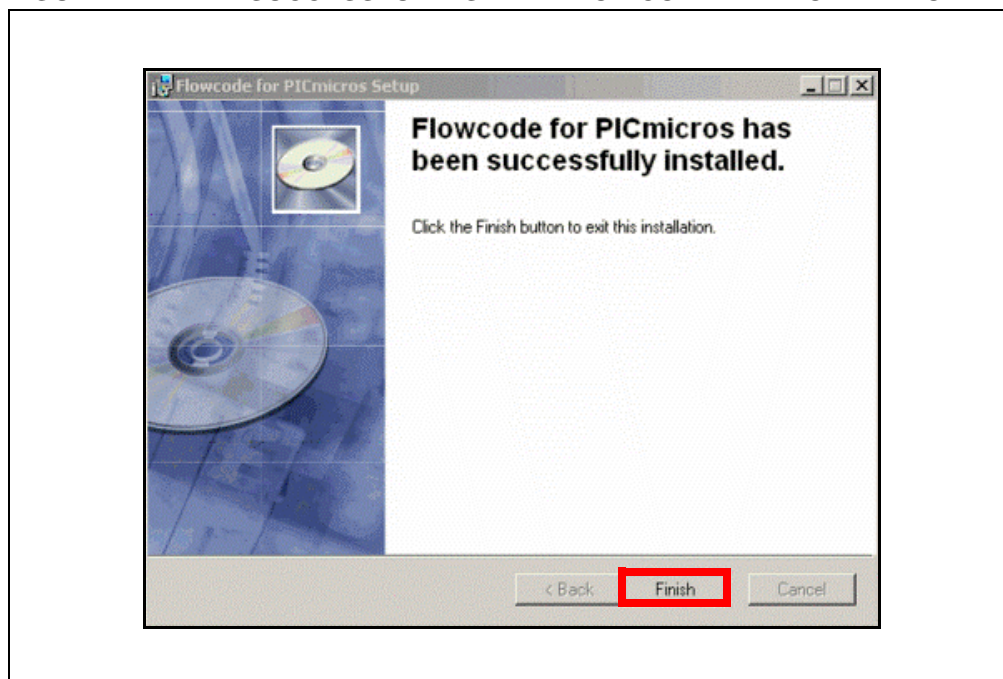
**FIGURE 1-10: ENTER FLOWCODE FOR PIC® MCU LICENSE KEY**



Press **Next>** to continue.

12. Following successful installation, a Confirmation window will open. Click **Finish** to exit the installer (Figure 1-11).

**FIGURE 1-11: SUCCESSFUL INSTALLATION CONFIRMATION WINDOW**



Flowcode V3 – free version is now installed and ready to use.

## 1.4 INSTALLING THE PICDEM™ LAB DEVELOPMENT KIT LABS (FLOWCODE VERSION)

The PICDEM Lab Development Kit homepage includes a download section with labs available for a number of software environments including Flowcode. To install the labs used in this user's guide, navigate to the downloads section and click on the `PICDEM_Lab_Flowcode.zip` link. Extract the contents of the .zip file to the `C:\` directory. These files provide a location to store any projects developed while completing the labs in this user's guide and solutions for each in a folder called `solutions`.

# PICDEM™ Lab Flowcode Companion Guide

---

NOTES:



---

---

## Chapter 2. General Purpose Input/Output Labs

---

---

### 2.1 INTRODUCTION

The following labs cover some of the fundamental features of the General Purpose Input/Output (GPIO) peripherals available on the PIC16F690. As the name implies, these peripherals are used for general purpose applications that can monitor and control other off-chip devices. Some PIC® devices have multiple GPIO peripherals on-chip including the PIC16F690 used in the following labs. Therefore, the PORTx naming convention is used. Available ports on the PIC16F690 are:

- PORTA
- PORTB
- PORTC

Reading through the data sheet highlights some of the unique characteristics associated with each port and the reader is encouraged to explore these in greater detail once comfortable with the labs in this user's guide. The labs will focus on two of the port peripherals: PORTC and PORTA. Labs will be naturally divided into two sections since these are General Purpose Input/Output peripherals:

- Output Labs
- Input Labs

Output labs will introduce the reader to concepts necessary to configuring these peripherals for output to off-chip devices using applicable registers by lighting 8 LEDs connected to the PORTC pins.

The Input labs will then add a push button interfacing to one of the PORTA pins to highlight concepts necessary for configuring these peripherals to receive information from off-chip devices. Finally, interrupts will be used to optimize the application.

### 2.2 GENERAL PURPOSE INPUT/OUTPUT LABS

- **Output Labs:**
  - Lab 1: Creating a Project in Flowcode to Light LEDs
  - Lab 2: Introducing the Software Control Loop to Flash LEDs (Delay Loop)
  - Lab 3: Using Macros to Rotate LEDs
- **Input Labs:**
  - Lab 4: Adding a Push Button Interface using a Component Macro
  - Lab 5: Adding an Interrupt and Importing Macros for a Push Button Interface

### 2.3 GPIO OUTPUT LABS

#### 2.3.1 Reference Resources

Free support documentation from [www.microchip.com](http://www.microchip.com):

- DS41262D – PIC16F690 data sheet
  - Section 2.2.2.2: Option Register
  - Section 2.2.2.3: Interrupt Control Register INTCON
  - Section 4: I/O Ports

On-line support for Flowcode Software:

- Forums: [www.matrixmultimedia.com/mmforums/](http://www.matrixmultimedia.com/mmforums/)
- Main web site including Frequently Asked Questions: [www.matrixmultimedia.com](http://www.matrixmultimedia.com)

## 2.3.2 Equipment Required for GPIO Output Labs

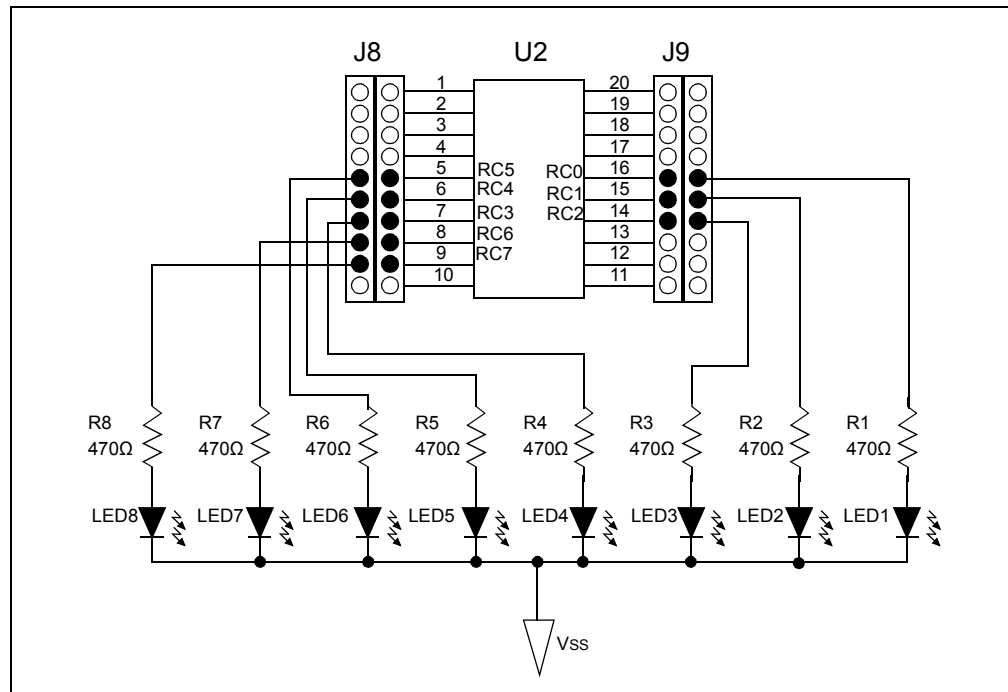
To complete the labs in this section, the following components are required:

1. 8 – Light Emitting Diodes
2. 8 – 470Ω resistors
3. PIC16F690 populating socket U2
4. Assorted jumper wires

## 2.3.3 PICDEM Lab Development Board Setup for GPIO Output Labs

The GPIO output labs will require that the PICDEM Lab Development Board be configured as shown in Figure 2-1, using the components listed in the previous section.

**FIGURE 2-1: PICDEM LAB SCHEMATIC FOR GPIO OUTPUT LABS**



Special care should be observed when connecting the LED jumper wires to the expansion headers surrounding the PIC16F690 as the PORTC pins are not in sequential order. The 470Ω resistors limit the current to each LED as per manufacturer's specifications. Furthermore, the electrical specifications for the PIC16F690 (see Section 17.0: "Electrical Specifications" in the PIC16F690 data sheet) specify a maximum output current by any I/O pin as 25 mA. Additionally, the maximum current sunk or sourced by all ports combined must not exceed 200 mA. The 470Ω resistors ensure that these specifications are met.

## 2.3.4 Lab 1: Creating a Project in Flowcode to Light LEDs

### 2.3.4.1 NEW REGISTERS USED IN THIS LAB

To configure the peripherals used in this lab, the following registers are used:

1. PORTC Register: PORTC (Register 4-11 in Section 4 of the PIC16F690 data sheet)
  - 8-bit bidirectional port

These registers are configured in the background automatically by the Flowcode Software and do not require any user manipulation:

2. PORTC Tri-State Register: TRISC (Register 4-12 in Section 4 of the PIC16F690 data sheet)
  - Configures corresponding bits in PORTC as either input or output
3. Analog Select Register High and Analog Select Register Low: ANSELH and ANSEL (Registers 4-4 and 4-3 in Section 4 of the PIC16F690 data sheet)
  - Configure associated pins for analog or digital input signals

### 2.3.4.2 OVERVIEW

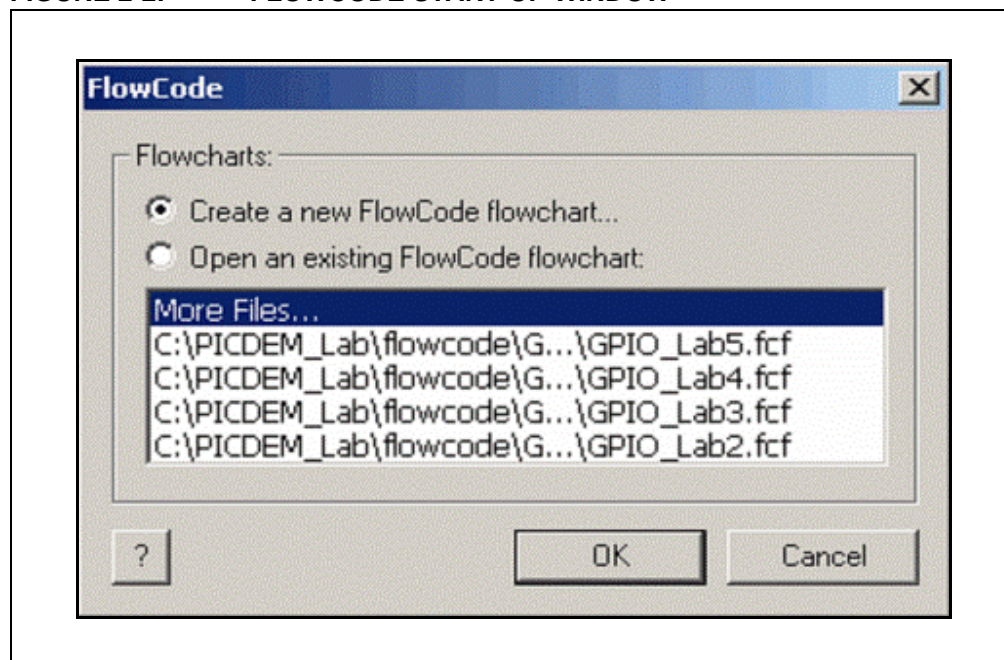
This first lab steps through the development of a Flowcode project to output data from the PORTC peripheral on the PIC16F690 to its associated pins. LEDs connected to PORTC pins will light when the associated pin is driven high (approx. VDD) or turn the LED OFF when driven low (approx. VSS). The Flowcode software will automatically configure the TRISC and ANSELH:ANSEL registers associated with PORTC to configure each bit as a digital output. The reader is encouraged to spend some time reviewing these registers to fully understand their functionality.

### 2.3.4.3 PROCEDURE

To complete this lab, the following steps are required:

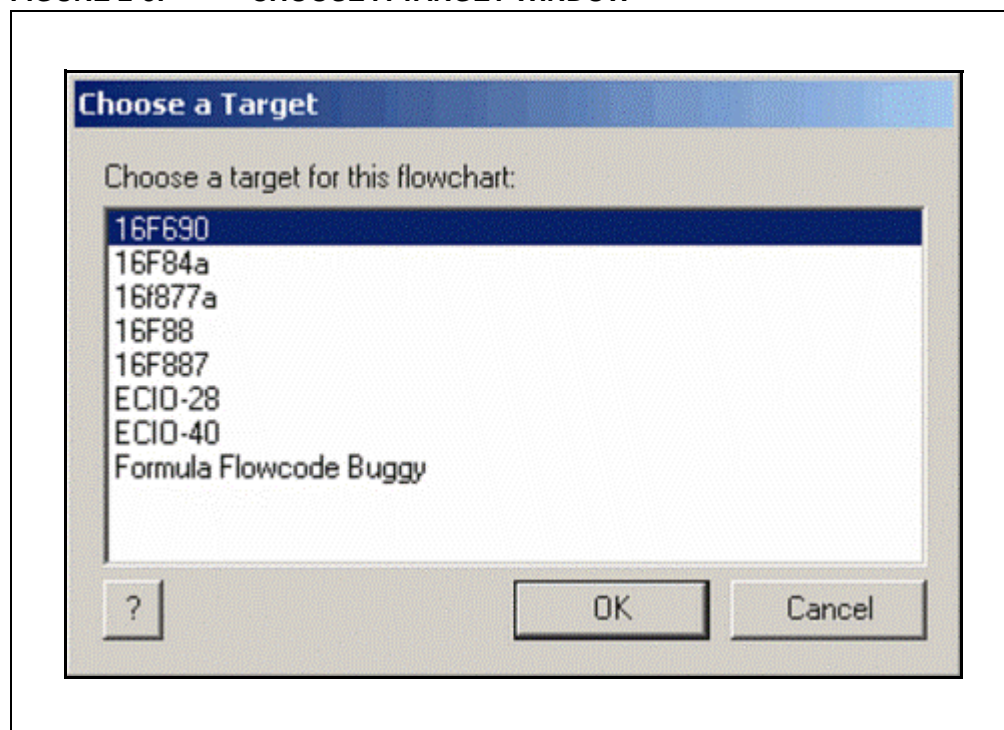
1. Open the Flowcode Software by selecting Start>All Programs>Matrix Multimedia>Flowcode Vx>Flowcode Vx
2. Click **OK** to continue.
3. In the Start-up window, ensure the **Create a new FlowCode Flowchart** radio button is selected and click **OK** (Figure 2-2).

FIGURE 2-2: FLOWCODE START-UP WINDOW



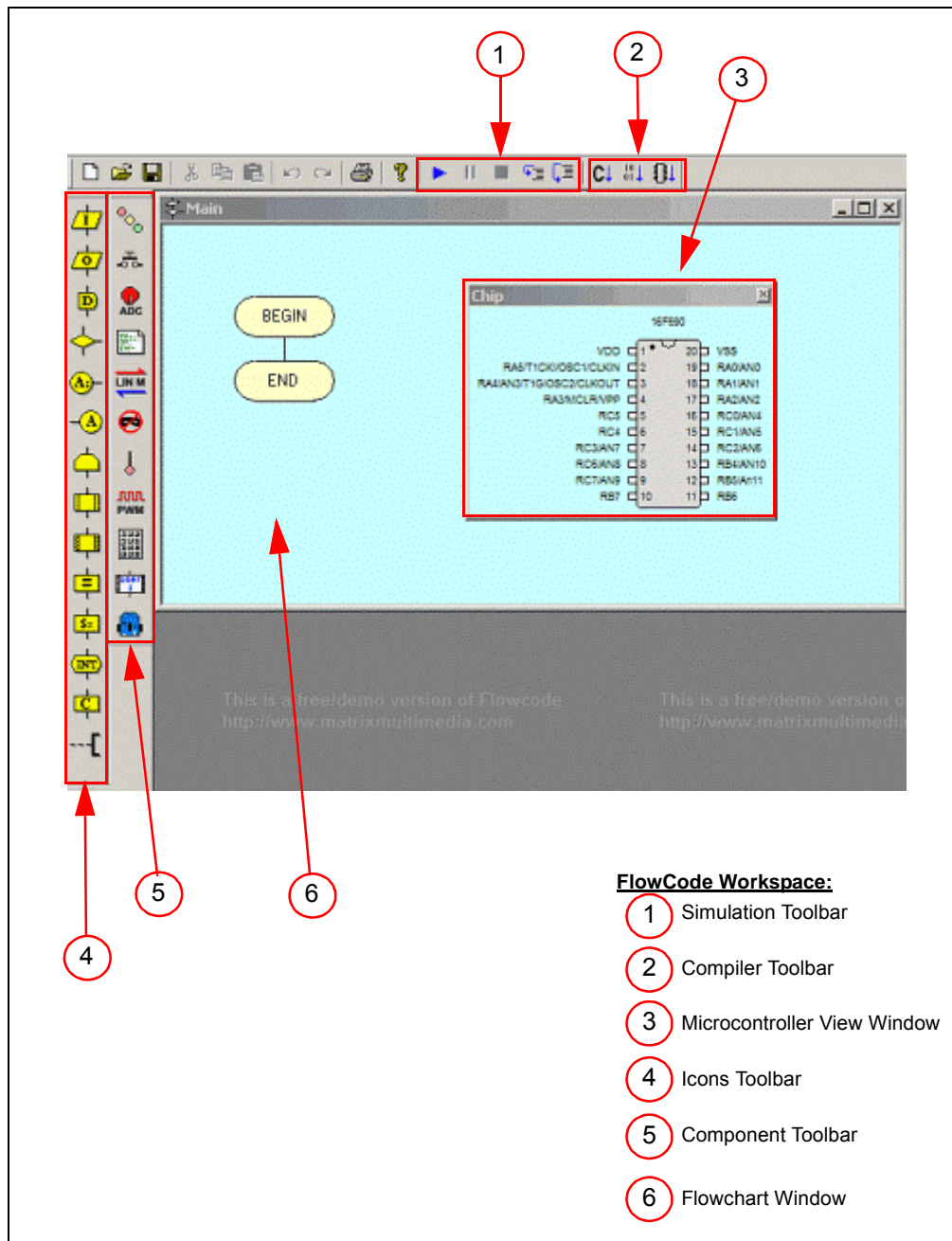
4. In the Choose a Target window, highlight the PIC16F690 microcontroller and click **OK** (Figure 2-3).

FIGURE 2-3: CHOOSE A TARGET WINDOW



5. The FlowCode workspace should now resemble Figure 2-4.

FIGURE 2-4: FLOWCODE WORKSPACE



**FlowCode Workspace:**

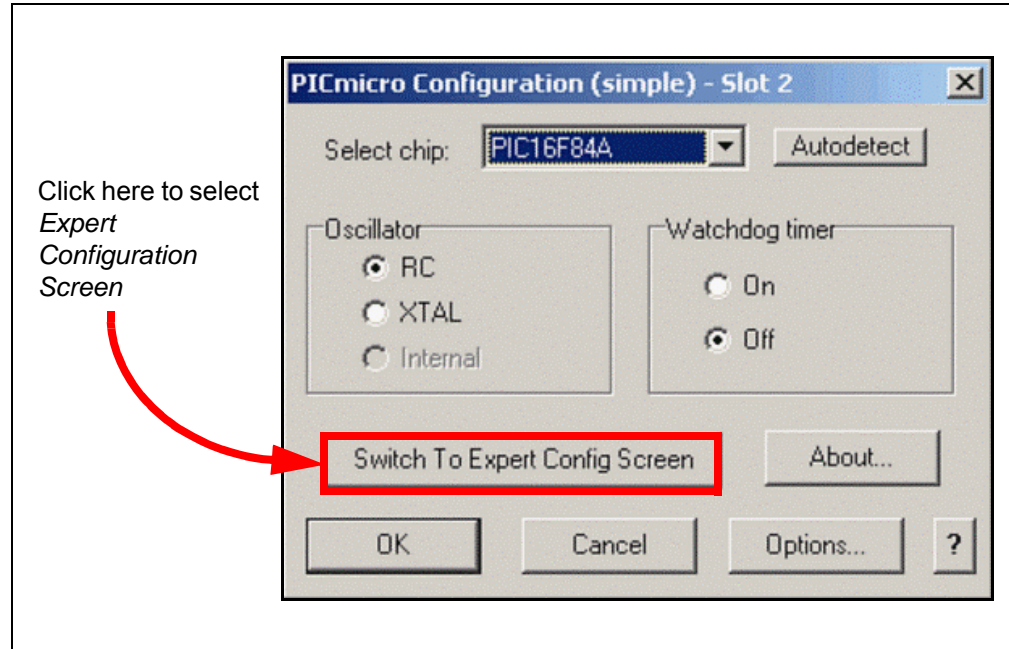
- 1 Simulation Toolbar
- 2 Compiler Toolbar
- 3 Microcontroller View Window
- 4 Icons Toolbar
- 5 Component Toolbar
- 6 Flowchart Window

**Note:** A more detailed overview of the workspace is provided in the FlowCode Help files: [Help>Contents](#).

6. Save the project to the  
 C:\PICDEM\_Lab\flowcode\GPIO\_Labs\GPIO\_Lab1 directory as  
 GPIO\_Lab1.fcf.

- Next, the PIC16F690 is configured using the PIC® MCU Configuration Tool to setup parameters, such as the oscillator used and other device functional characteristics desired (for more information on these configuration parameters, refer to the PIC16F690 data sheet, Section 14.1 “Configuration Bits”). Open the PIC MCU Configuration tool by selecting in the FlowCode menu Chip>Configure.... In the PIC® Configuration window, select “Switch to Expert Config Screen” (Figure 2-5).

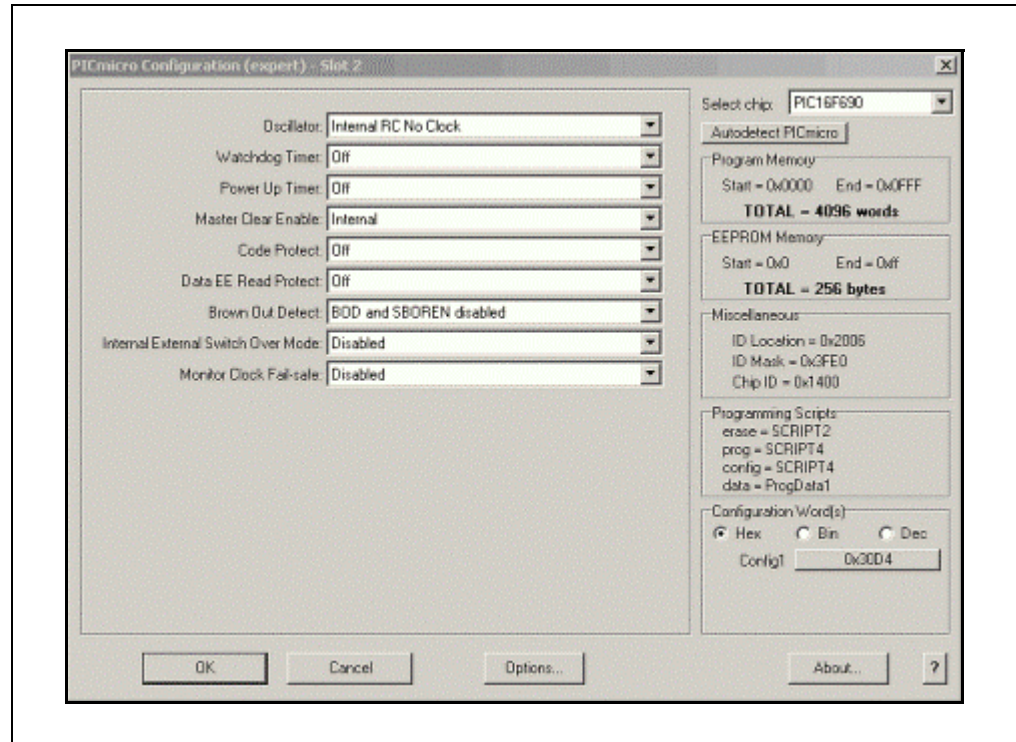
**FIGURE 2-5: PIC® CONFIGURATION WINDOW**



- In the PIC® Configuration (expert) window, ensure that the PIC16F690 is selected as the target device and use the drop-down menus to configure as shown in Figure 2-6.



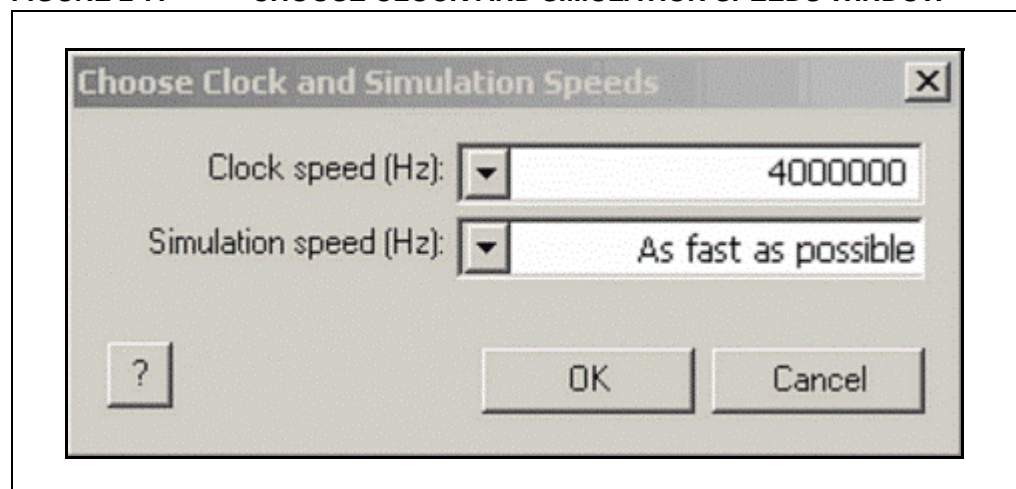
**FIGURE 2-6: PIC16F690 CONFIGURATION FOR LAB 1**



Click **OK** to exit the tool.

9. This application will use the PIC16F690 4 MHz internal oscillator. To configure the oscillator speed, select *Chip>Clock Speed...* in the FlowCode workspace. In the Choose Clock and Simulation Speeds window, select 4000000 (4 MHz) from the Clock speed (Hz) drop-down menu and ensure that the Simulation speed (Hz) drop-down menu indicates “As fast as possible” (Figure 2-7).

**FIGURE 2-7: CHOOSE CLOCK AND SIMULATION SPEEDS WINDOW**

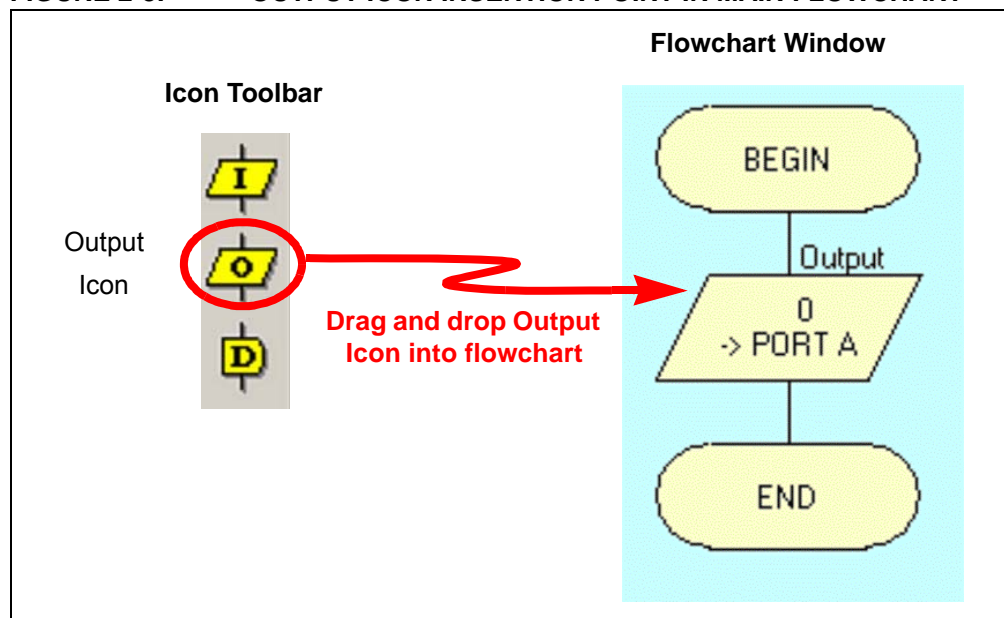


Click **OK**.

The PIC16F690 is now configured for the application. In the next series of steps, the flowchart for the application will be developed:

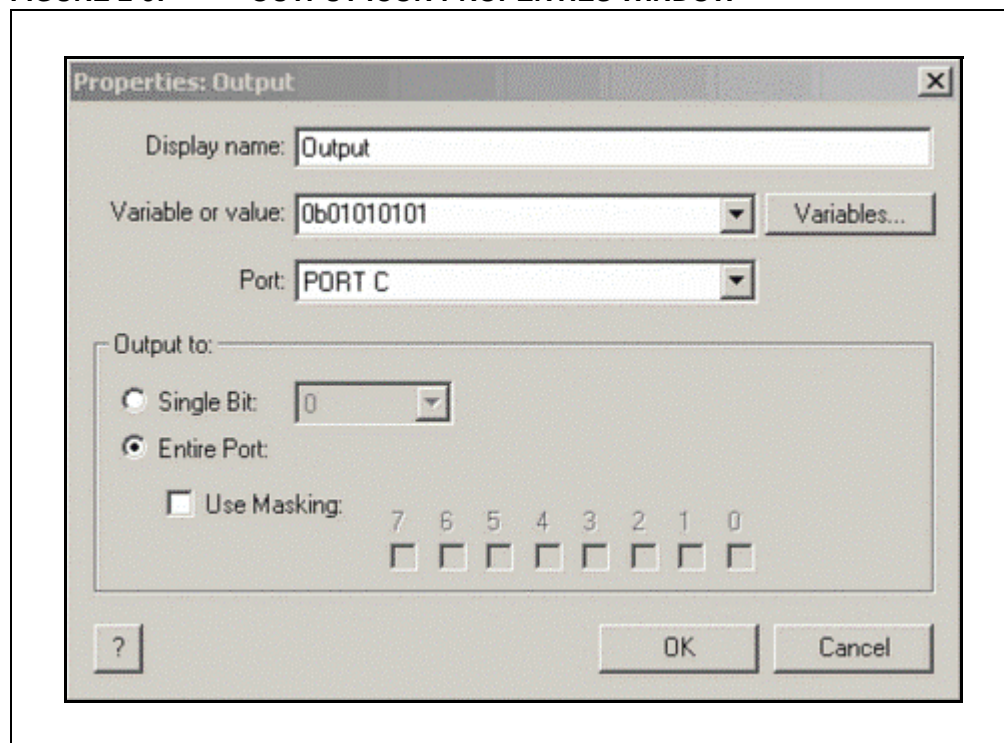
- To output values from the PORTC register to the associated pins on the PIC16F690, the **Output** icon is used. In the icon toolbar, click the mouse over the **Output** icon and drag to an insertion point between the BEGIN and END points in the Flowchart window (Figure 2-8).

**FIGURE 2-8: OUTPUT ICON INSERTION POINT IN MAIN FLOWCHART**



- In the Flowchart window, open the **Output** icon properties by either doubling clicking on the icon or by right clicking on the icon and selecting Properties (Figure 2-9).

**FIGURE 2-9: OUTPUT ICON PROPERTIES WINDOW**





# General Purpose Input/Output Labs

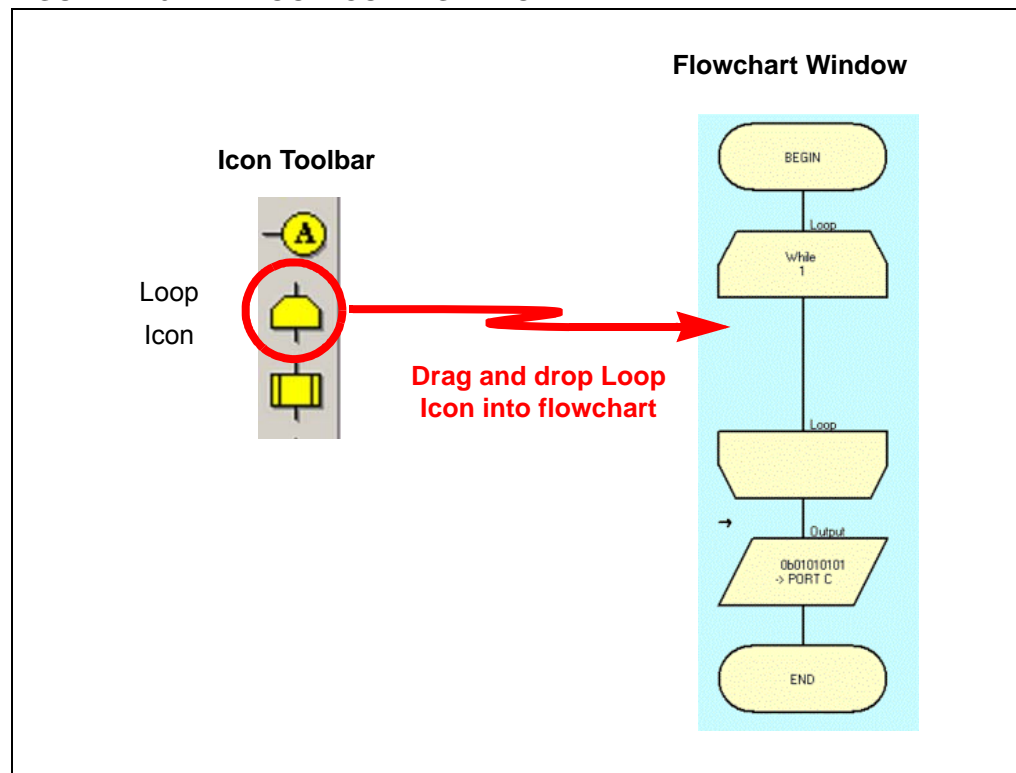
Configure the **Output** icon properties as shown in Figure 2-9 as follows:

- a) Ensure that the **Entire Port** radio button is selected. In this way, the value specified in the Variable or Value window will be output to the entire port. More information on these properties can be found in the Flowcode Software Help files.
- b) The Port drop-down menu will contain all of the available ports on the target device. Ensure that the PORTC is selected.
- c) The Variable or Value window is used to specify a value to be output to the selected PORT register and associated pins. Numeric values can be specified using the decimal, hexadecimal or binary radix. In this application, the binary radix is used by implementing an 8-bit binary value, since PORTC is an 8-bit register (e.g., 0bxxxxxxxx). Set the output value to 0b01010101.

Click **OK** to close the window.

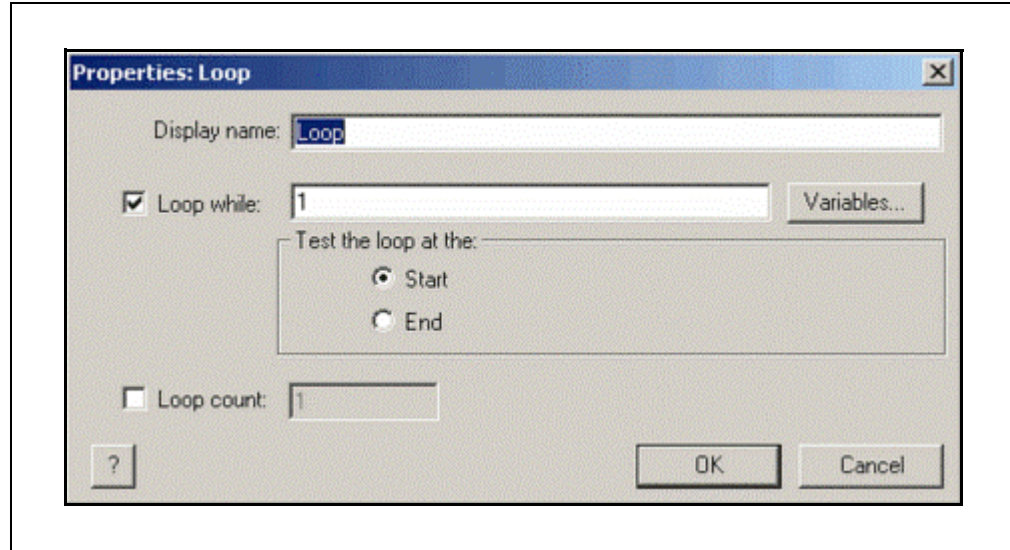
12. Most embedded applications will implement an Infinite loop of some kind that will repeatedly execute instructions while power is applied to Microcontroller. All labs in this user's guide will implement an Infinite loop. Select the **Loop** icon from the icon toolbar and drag into flowchart somewhere between the BEGIN and END points (Figure 2-10).

**FIGURE 2-10: LOOP ICON INSERTION**



**Loop** icons are used to repeat a task contained within it until specified conditions have been fulfilled or to perform the loop a set number of times. These parameters can be edited within the Properties: Loop window by double-clicking on the icon in the flowchart or by right clicking on the icon and selecting **Properties...** (Figure 2-11).

FIGURE 2-11: LOOP ICON PROPERTIES WINDOW

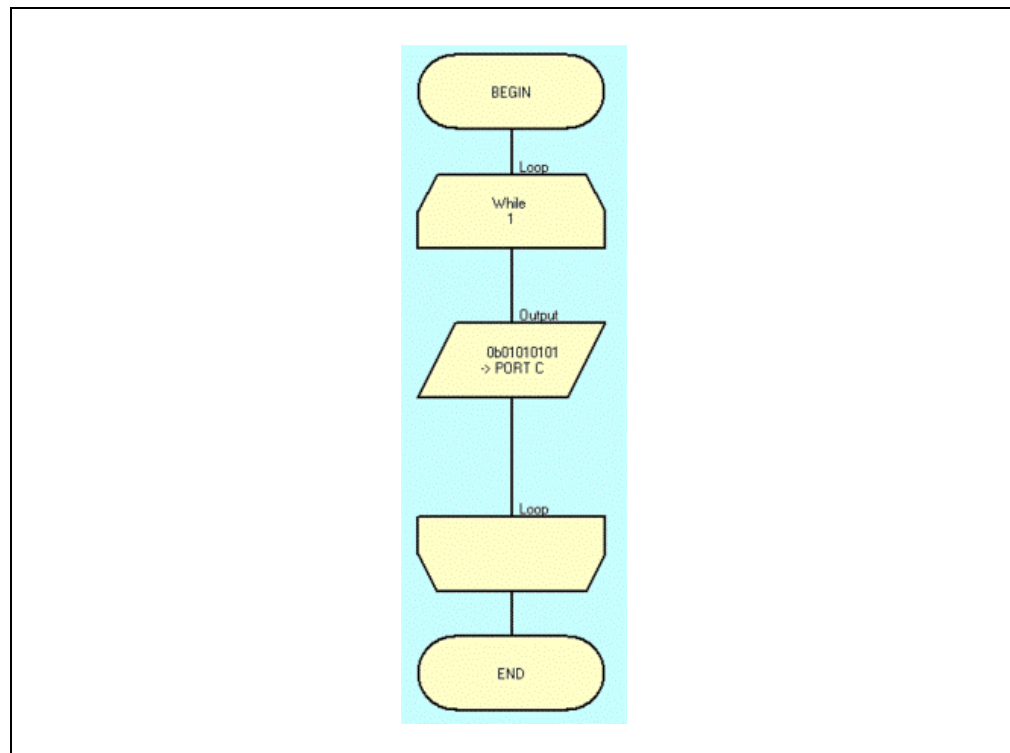


Ensure that the **Loop** icon properties resemble Figure 2-11. Any icon within the loop boundaries will continue to execute while the condition specified in the Loop While window is true. To implement the required Infinite loop, the default condition of '1' is used, meaning that the condition will always be true as long as there is power to the microcontroller. More information on the loop properties can be found in the Flowcode Software Help files.

Click **OK** to close the window.

13. Click the **Output** icon added earlier in the flowchart and drag/drop it between the start and finish of the While loop added in the previous step. The flowchart should now resemble Figure 2-12.

FIGURE 2-12: UPDATED FLOWCHART WITH OUTPUT ICON INSIDE OF LOOP

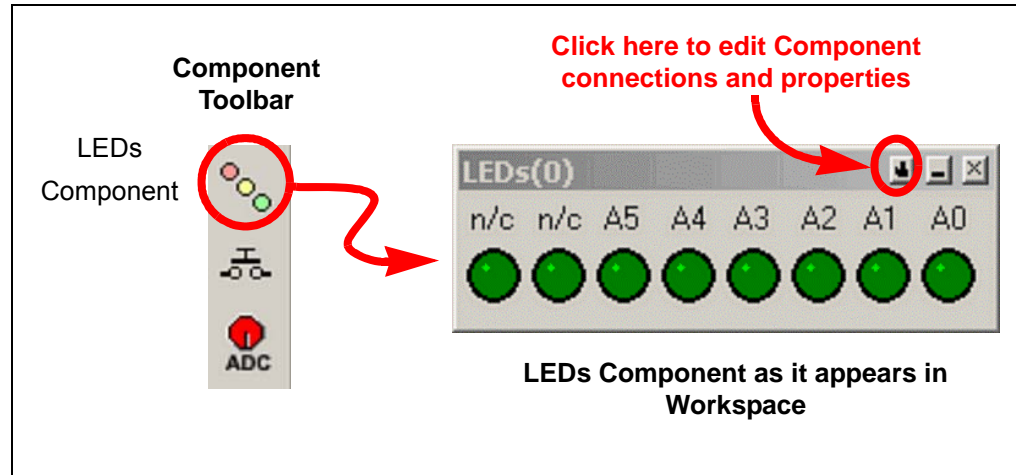


# General Purpose Input/Output Labs

This completes construction of the application flowchart. In the next steps, the project will be simulated to ensure functionality.

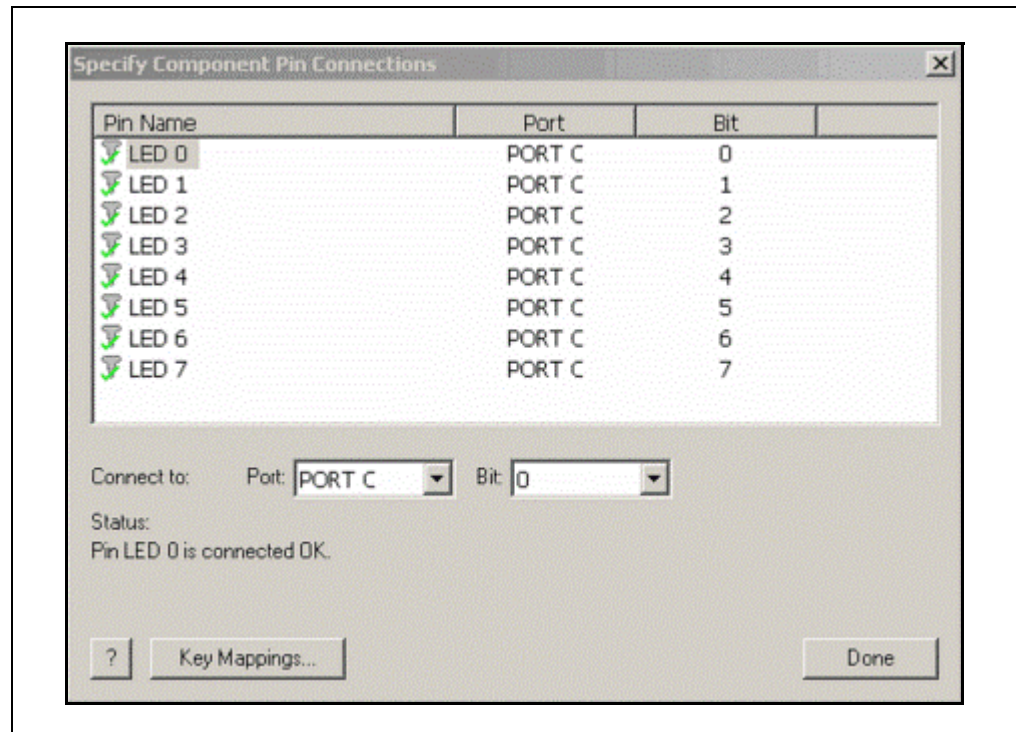
14. This application will be used to light LEDs connected to the associated PORTC pins on the PIC16F690. To simulate the effectiveness of the flowchart developed in the previous steps to accomplish this task, the Flowcode Software simulator will be used. In the **Components** toolbar, click on the LEDs component to have it appear in the Flowcode workspace (Figure 2-13).

**FIGURE 2-13: USING THE LEDES COMPONENT**



The LEDs component connects to the PORTA register by default. To simulate LEDs connected to PORTC pins, click on the small button on the component as shown in Figure 2-13 and select **Component Connections...** from the drop-down menu. The Specify Component Pin Connections window should now be open. Using the "Connect to:" drop-down menu, select PORTC as shown in Figure 2-14.

**FIGURE 2-14: SPECIFY COMPONENT PIN CONNECTIONS WINDOW**



# PICDEM™ Lab Flowcode Companion Guide

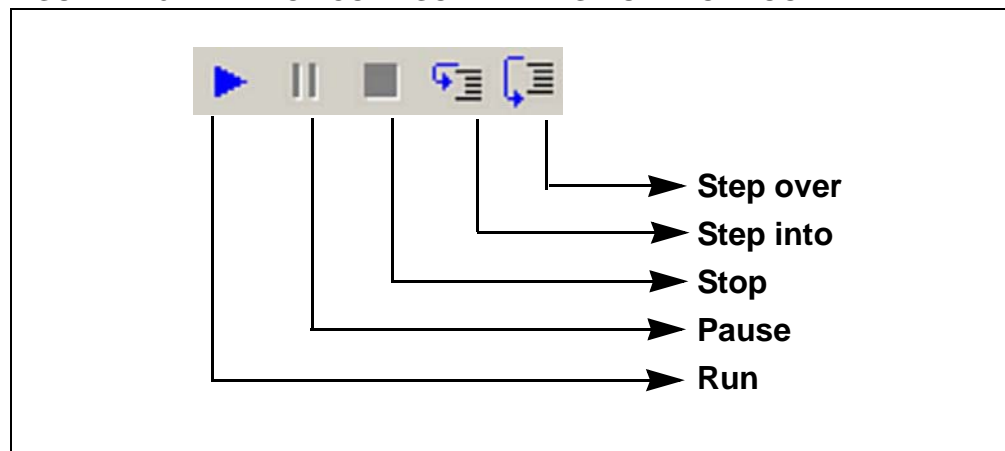
Each bit in the PORTC register should now be connected to one of the 8 LEDs of the LEDs component. Click **Done** to close the window.

More information on the LEDs component and configuration can be found in the Flowcode Software Help files.

The project is now ready to simulate.

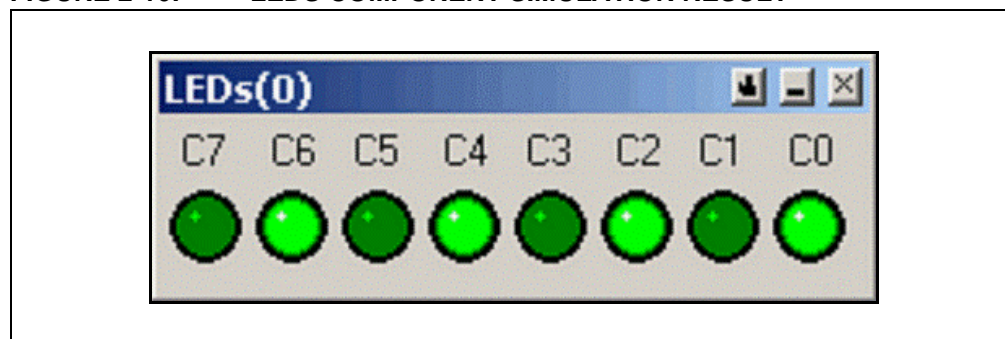
15. In the **Simulation Toolbar**, click the **Run** button to run the simulation or click the **Step Into** button to step through each block in the flowchart sequentially (Figure 2-15).

**FIGURE 2-15: FLOWCODE SOFTWARE SIMULATION TOOLBAR**




Regardless of the simulation method used, the LEDs component in the workspace should now light each LED as per the value loaded into the **Output** icon in Step 11.c, as shown in Figure 2-16.

**FIGURE 2-16: LEDS COMPONENT SIMULATION RESULT**



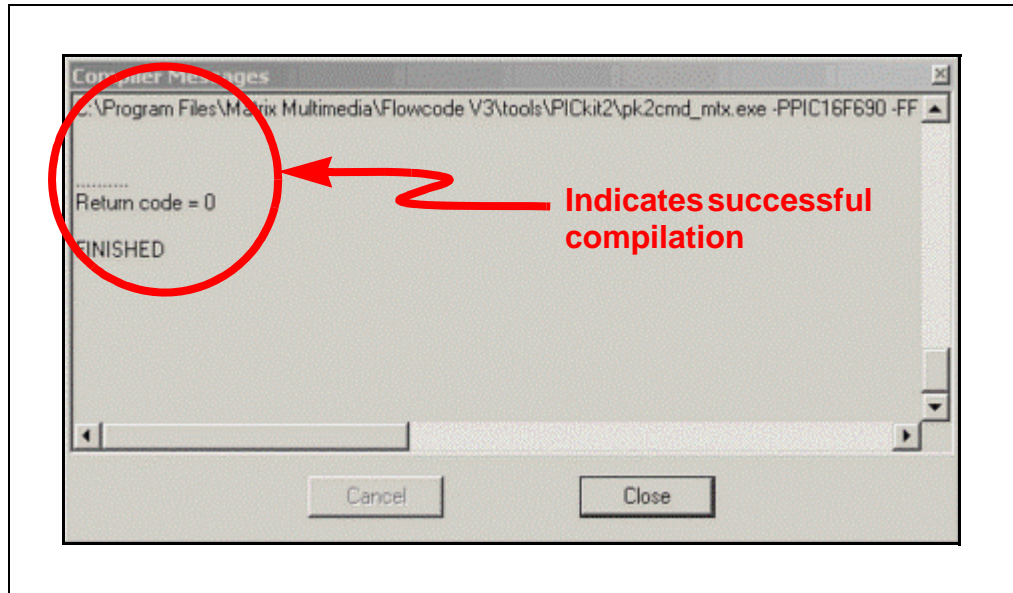
In the next steps, the project will be compiled and then downloaded to the PIC16F690 microcontroller on the PICDEM Lab Development Board.

16. Connect the PICkit™ 2 Programmer/Debugger to an available USB port on the PC using the supplied cable, then to the J6 ICSP1 for the PIC16F690 connector on the PICDEM Lab Development Board.
17. To begin programming, press the **Compile to Chip** button  in the compiler toolbar or select *Chip>Compile to Chip...* in the Flowcode workspace. The Compiler window should now open and begin compiling the project to check for any errors. Following a successful compilation, the programmer will launch beginning the download process to the PIC16F690. The red **Busy** LED on the PICkit 2 Programmer/Debugger will light briefly indicating that the PIC16F690 is being programmed. Once completed, the Compiler Messages window should indicate that the project compiled and was successfully downloaded to the

# General Purpose Input/Output Labs

PIC16F690, as shown in Figure 2-17. If any errors are indicated, the user should recheck the preceding steps.

**FIGURE 2-17: COMPILER MESSAGES WINDOW INDICATING SUCCESSFUL COMPILATION**

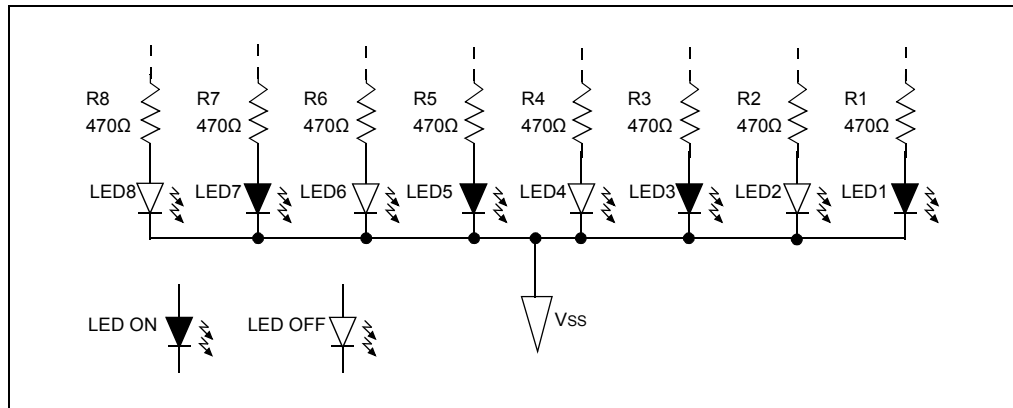


At this point, the PIC16F690 is programmed. The PICkit 2 Programmer/Debugger will power the device while connected to the USB port on the PC. Alternately, disconnect the PICkit 2 Programmer/Debugger from the J6 ICSP1 connector on the PICDEM Lab Development Board and use either a 9V battery or 9V supply.

## 2.3.4.4 TESTING THE APPLICATION

Once programmed and powered, the LEDs connected to the individual PORTC pins should now resemble the output shown in Figure 2-18.

**FIGURE 2-18: LAB 1 LED OUTPUT**



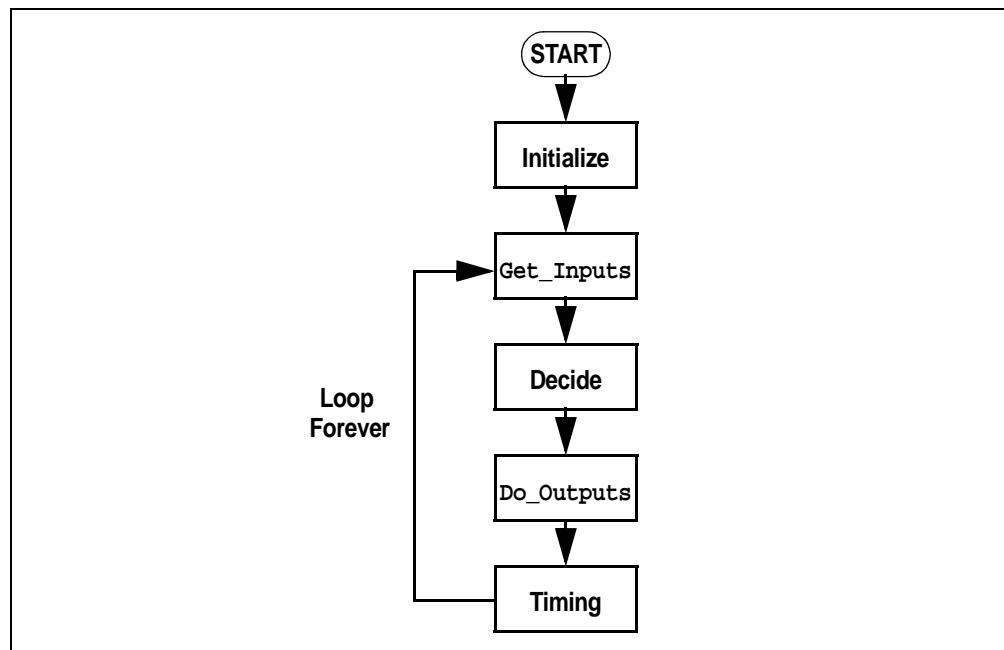


## 2.3.5 Lab 2: Introducing the Software Control Loop to Flash LEDs (Delay Loop)

### 2.3.5.1 OVERVIEW

This lab expands on the previous by flashing the LEDs connected to the PORTC pins ON/OFF in 1-second intervals. Also, the concept of the Software Control loop to maintain an organized and logical flow to program execution is introduced. The remainder of the labs in this user's guide will divide the embedded application into functional blocks, as shown in Figure 2-19.

**FIGURE 2-19: THE SOFTWARE CONTROL LOOP**



Referring to Figure 2-19, the `Initialize` functional block contains all the firmware that initializes peripherals and variables used in the application. Note that this block is called only once during execution (it is outside of the Infinite loop). Next, the `Get_Inputs` block is called. Here all inputs are retrieved whether from off-chip devices via the PIC16F690 I/O pins or from internal sources such as registers. Once all inputs have been obtained, the `Decide` block contains firmware that will make any decisions based off of these inputs. Once all input information has been analyzed, the `Do_Outputs` block outputs information either to off-chip devices via the I/O pins or to internal destinations such as registers based off the decisions made in the previous block. Finally, the `Timing` block sets the rate for the entire Software Control loop.

In this lab, not all of these functional blocks are required to flash the LEDs connected to the PORTC pin ON/OFF. There are no inputs into the application eliminating the need for the `Get_Inputs` block. The remaining blocks will be organized as follows:

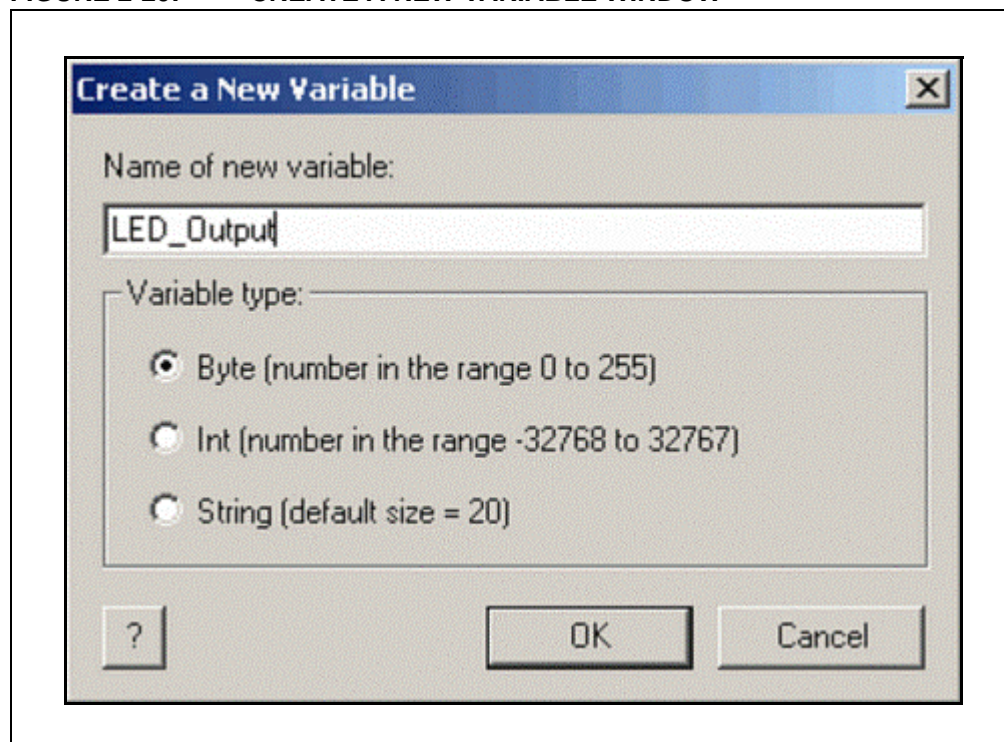
- a) **Initialize:** This functional block will initialize an 8-bit variable called `LED_Output` that will contain the value to be output to PORTC. Using a variable, rather than simply assigning a value to PORTC, allows the firmware to manipulate the data in the variable without outputting data to the I/O pins until it is finally assigned to

- the PORT in the `Do_Outputs` block.
- b) **Decide:** This functional block will perform an XOR operation on the data contained within the `LED_Output` variable each time through the Control loop. When a value is XOR'd with itself, the result is '0' (i.e., 1 XOR'd with 1 = 0, 0 XOR'd with 0 = 0). When a value is XOR'd with a value different than itself, the result is '1' (i.e., 1 XOR'd with 0 = 1). Therefore, each time through the loop, `LED_Output` data bits will toggle from 1-to-0 or 0-to-1, depending on its current value. Remembering that when the value on the I/O pin is '0', the LED will turn off, when the value is '1' the LED will turn on.
  - c) **Do\_Outputs:** This block assigns the current value in the `LED_Output` variable to the PORTC register which in turn lights the LEDs connected to its associated pins.
  - d) **Timing:** This block implements a delay to flash the LEDs connected to PORTC I/O pins ON/OFF in 1-second intervals. As configured, the PIC16F690 executes 1 million instructions per second. At this rate, the Software loop execution needs to be slowed down so that the LED flashing is visible to the eye. This is accomplished using a **Delay** icon within the main Flowcode Flowchart.

## 2.3.5.2 PROCEDURE

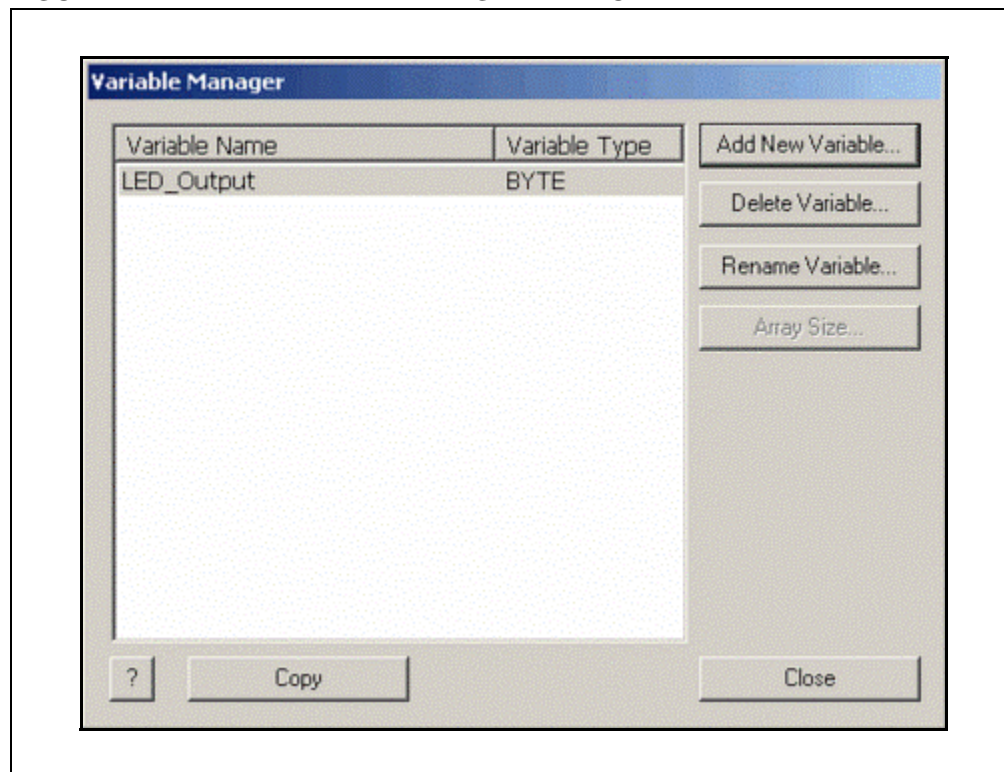
1. Create a new project in Flowcode using steps 1-9 from the previous lab saving the project as `GPIO_Lab2.fcf` in the `C:\PICDEM_Lab\flowcode\GPIO_Labs\GPIO_Lab1` directory.
2. First, the `LED_Output` variable is created. In the Flowcode Workspace, select *Edit>Variables...*. In the Variable Manager window, select **Add New Variable...** The Create a New Variable window should now be open. In the Name of New Variable: window enter `LED_Output`. Ensure that the **Byte (number in the range 0 to 255)** radio button is selected in the "Variable type:" section. Note that a number of variable types can be selected. In this application, the `LED_Output` variable will be assigned to the 8-bit PORTC register. Therefore, the Byte variable type will be used. The Create a New Variable window should now resemble Figure 2-20.

FIGURE 2-20: CREATE A NEW VARIABLE WINDOW



Click **OK** to continue. The Variable Manager window should now display the newly created variable as shown in Figure 2-21.

FIGURE 2-21: VARIABLE MANAGER WINDOW



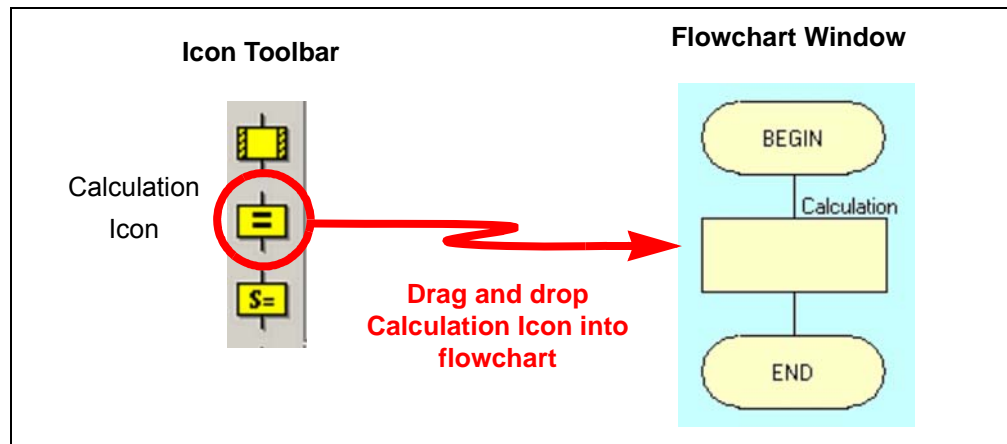
Click **Close**.



# General Purpose Input/Output Labs

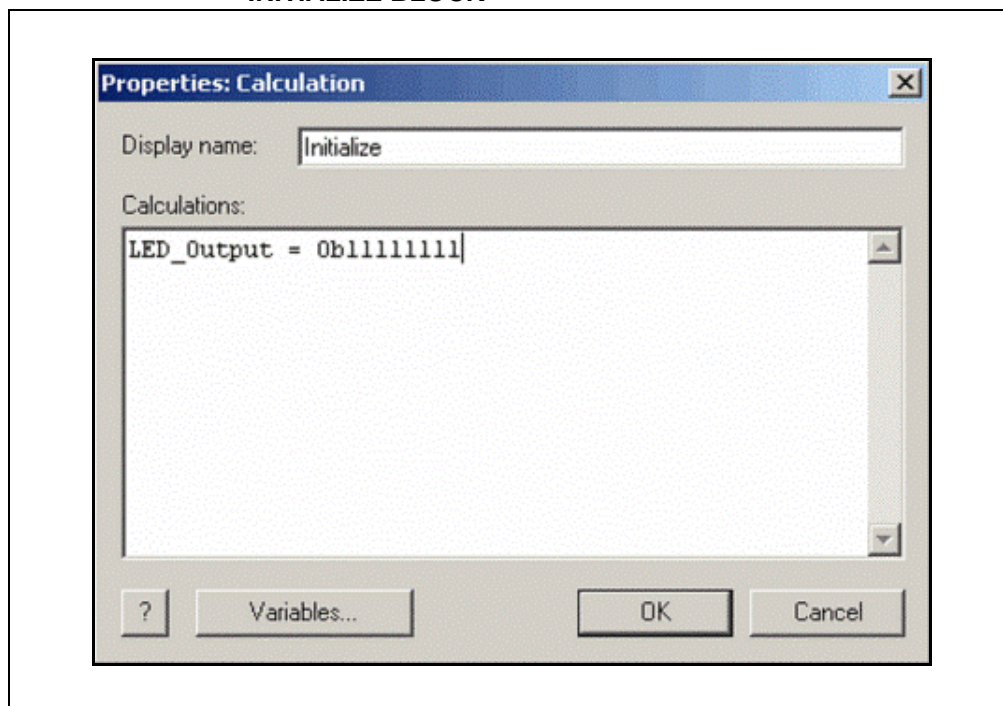
- Next, the Software Control loop is constructed. First, the `Initialize` block will be created to initialize the `LED_Output` variable to a known value. It is good programming practice to always initialize variables to a known value. To accomplish this, the Flowcode **Calculation** icon is used. Drag/drop the **Calculation** icon from the icon toolbar into the workspace between the `BEGIN` and `START` blocks of the flowchart (Figure 2-22).

FIGURE 2-22: ADDING THE CALCULATION ICON



- Open the **Calculation** icon properties by double-clicking on the icon or by right-clicking on the icon and selecting **Properties...** In the Properties: Calculation window, select the **Variables...** button to open the Variable Manager window. In the Variable Manager window, highlight the `LED_Output` variable created earlier and click on the **Use Variable** button on the right side of the window to add it to the Properties: Calculation window. Next, change the "Display name:" in the Properties: Calculation window to `Initialize`. Add the calculation code to assign the initial value of `0b11111111` (all `LED_Output` data bits high), as shown in Figure 2-23.

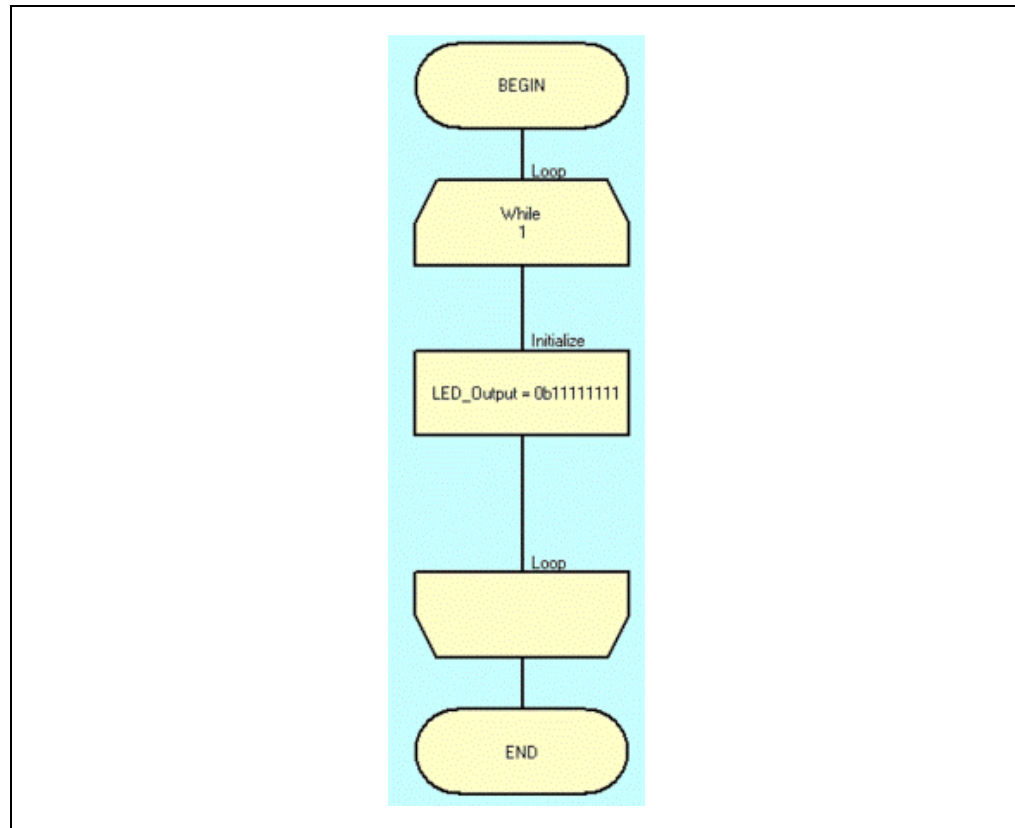
FIGURE 2-23: CONFIGURING THE CALCULATION PROPERTIES FOR THE INITIALIZE BLOCK



Click **OK** to close the window.

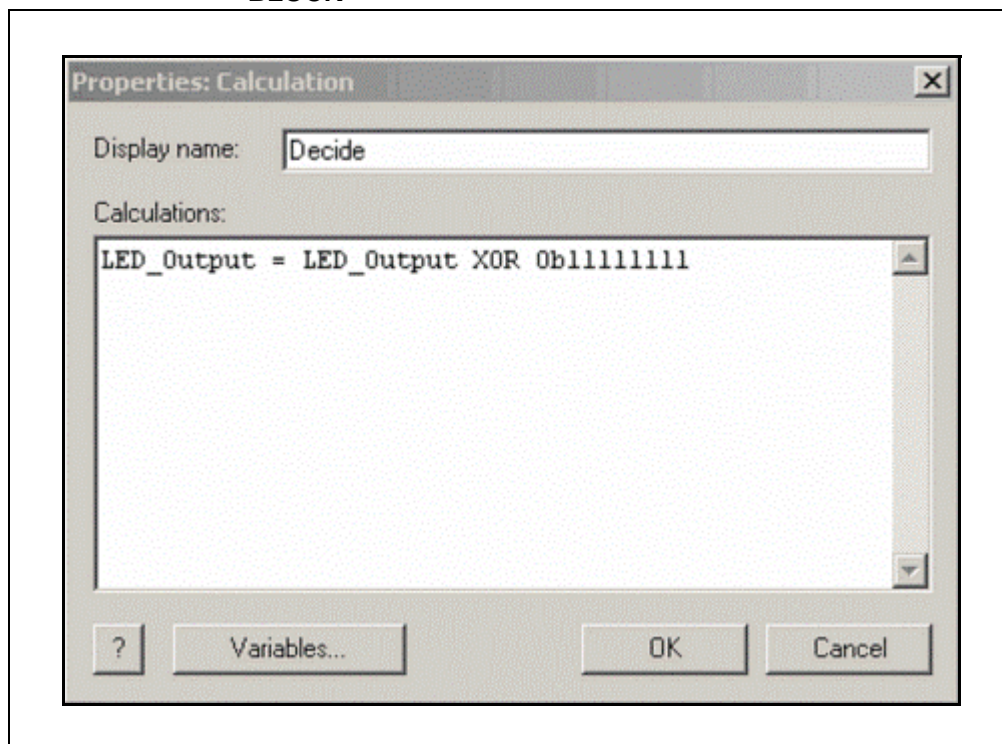
5. Immediately under the `Initialize` block, drag/drop a **Loop** icon to create the Infinite loop, as was done in step 12 of the previous lab. The flowchart should now resemble Figure 2-24.

FIGURE 2-24: FLOWCHART WITH INITIALIZE BLOCK AND INFINITE WHILE LOOP



- Next, the `Decide` functional block that will perform the XOR operation on the `LED_Output` data bits will be added. Again, the **Calculation** icon is used. Drag/drop a new **Calculation** icon into the workspace as was done in step 3 above, only this time within the `While` loop. Open the Properties: Calculation window, change the "Display name:" to `Decide`, add the `LED_Output` variable as per step 4 and add the XOR operation calculation code as shown in Figure 2-25.

FIGURE 2-25: PROPERTIES: CALCULATION FOR DECIDE FUNCTIONAL BLOCK

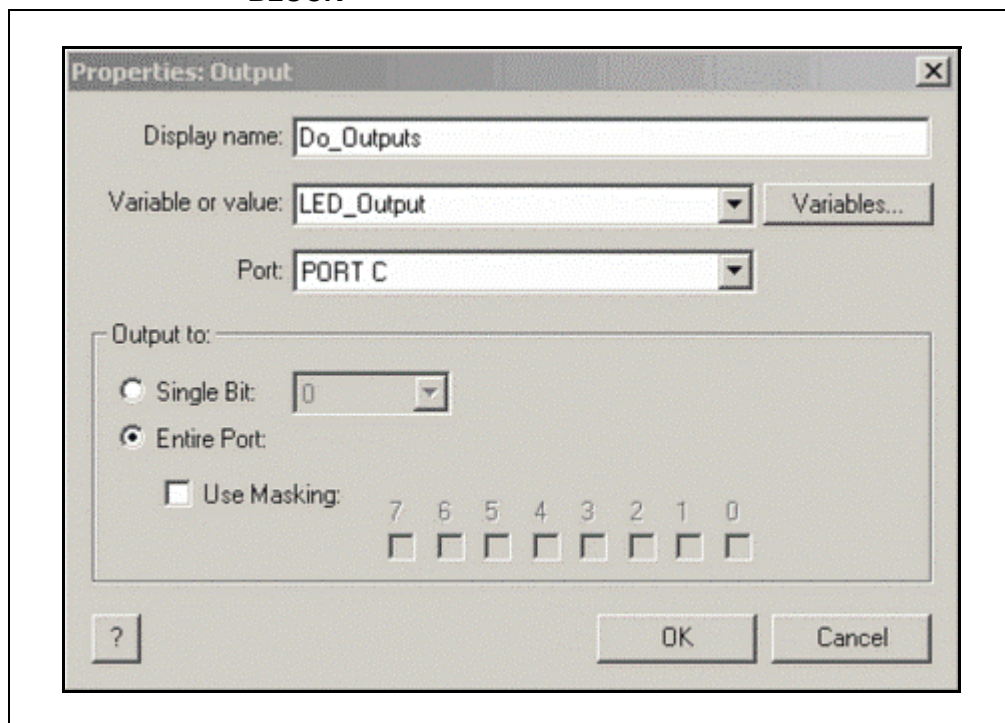


The code in Figure 2-25 states “**make** LED\_Output **equal to** LED\_Output **data XOR’d with 0b11111111**”.

Click **OK** to close the window.

7. Next, the Do\_Outputs functional block is added using the **Output** icon from the icon toolbar. Drag/drop an **Output** icon into the flowchart immediately after the Decide block created in the previous step. Open the Properties: Output window. Change the “Display name:” to Do\_Outputs. Ensure that the PORTC is selected in the “Port” drop-down menu and that the **Entire Port** radio button is enable. Use the “Variable or value” drop menu to select the LED\_Output variable. The Properties: Output window should now resemble Figure 2-26.

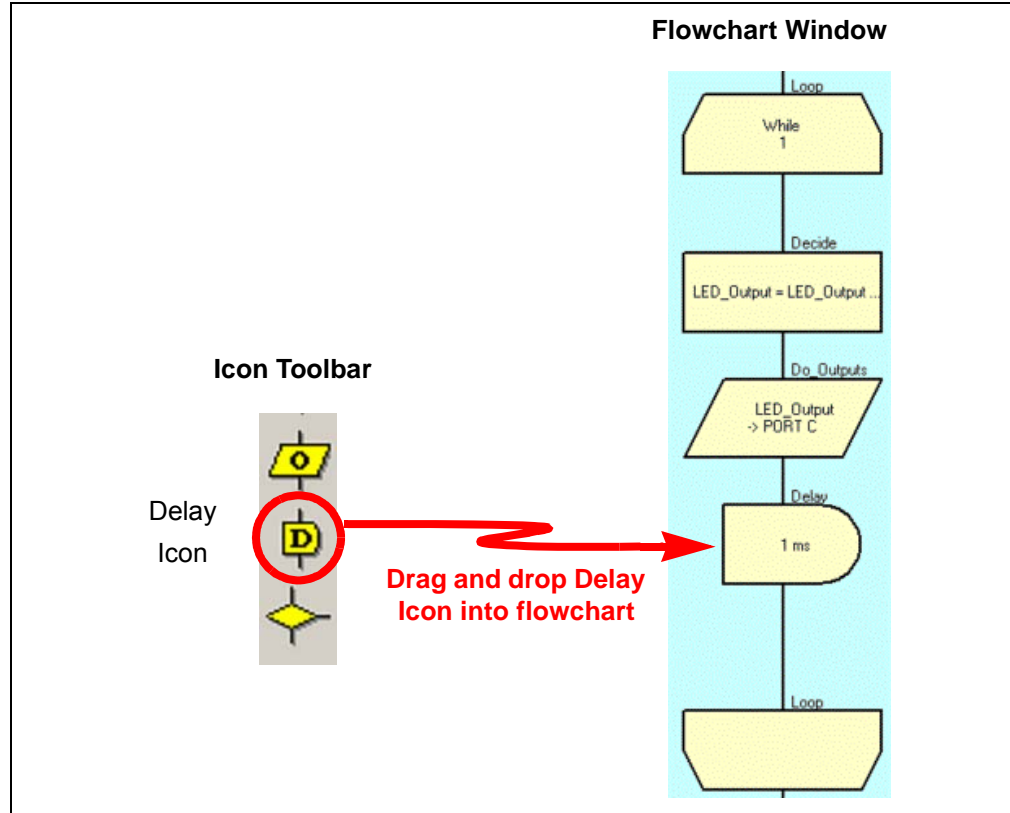
FIGURE 2-26: PROPERTIES: OUTPUT FOR DO\_OUTPUTS FUNCTIONAL BLOCK



Click **OK** to close the window.

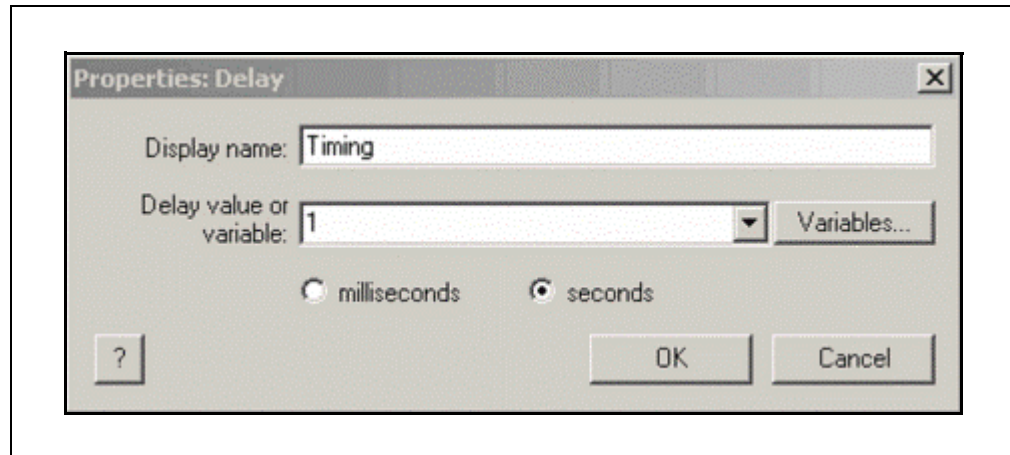
8. Finally, the `Timing` functional block is added to the flowchart to slow down the Software Control loop to execute in 1-second intervals. To accomplish this, the **Delay** icon is used. Drag/drop the **Delay** icon from the icon toolbar into the flowchart immediately after the `Do_Outputs` functional block (Figure 2-27).

FIGURE 2-27: ADDING THE DELAY ICON



9. Open the **Properties: Delay** by double-clicking on the **Delay** icon or by right-clicking on the icon and selecting **Properties...** In the **Properties: Delay** window, change the "Display name:" to **Timing**, add a value of '1' to the **Delay Value or Variable** window and ensure that the **Seconds** radio button is selected (Figure 2-28).

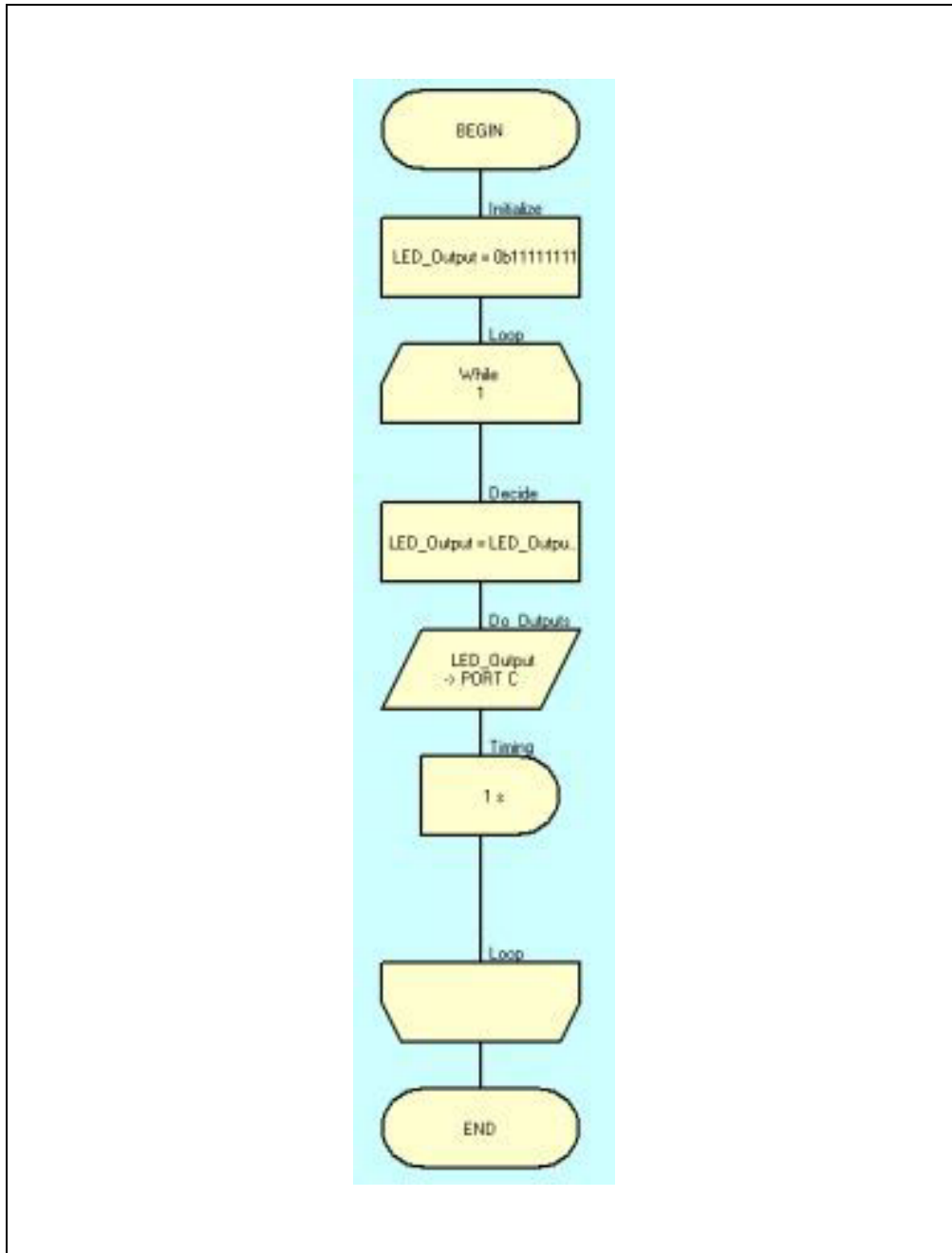
FIGURE 2-28: PROPERTIES: DELAY FOR TIMING FUNCTIONAL BLOCK



Click **OK** to close the window.

The project flowchart should now resemble Figure 2-29.

FIGURE 2-29: GPIO\_LAB 2 FLOWCHART



Compile the project to HEX and download to the PIC16F690 as per steps 16 through 20 of Lab 1. There should be no errors.

### 2.3.5.3 TESTING THE APPLICATION

The LEDs connected to the individual PORTC pins should now all flash ON/OFF in 1-second intervals.

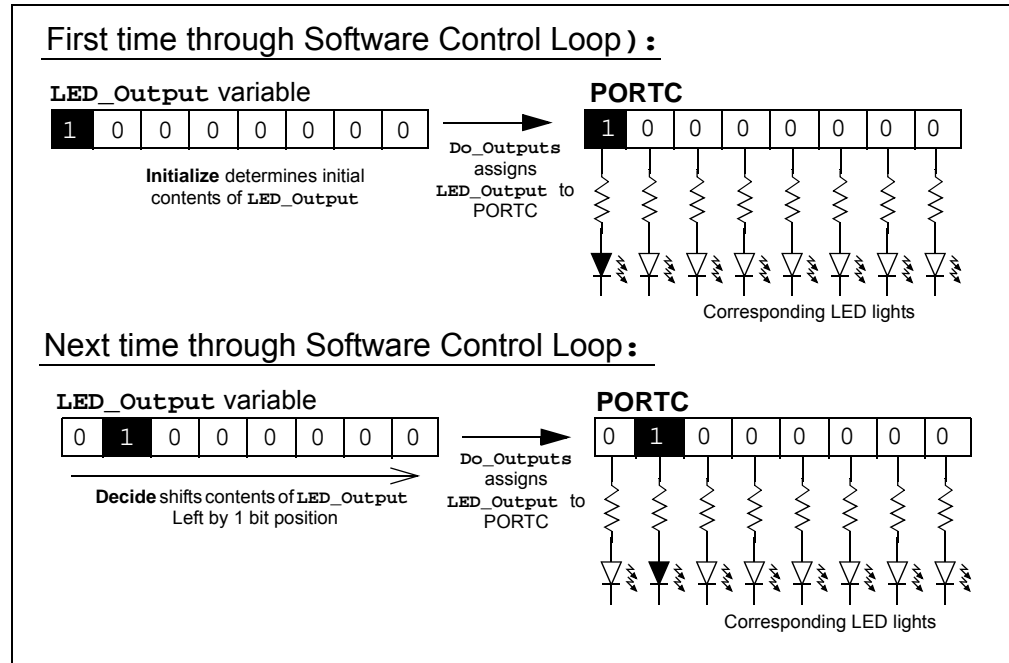


## 2.3.6 Lab 3: Using Macros to Rotate LEDs

### 2.3.6.1 OVERVIEW

This lab introduces a macro called by the `Decide` functional block that will shift a high bit in the `LED_Output` variable from left-to-right each time through the Software Control loop before being assigned to the `PORTC` register for output to the LEDs in the `Do_Outputs` functional block (Figure 2-30).

**FIGURE 2-30: RESULTS OF DO\_OUTPUT( )**



Macros are sections of code that can be used and reused in projects. Macros allow complex tasks to be handled by code blocks and can be imported and exported.

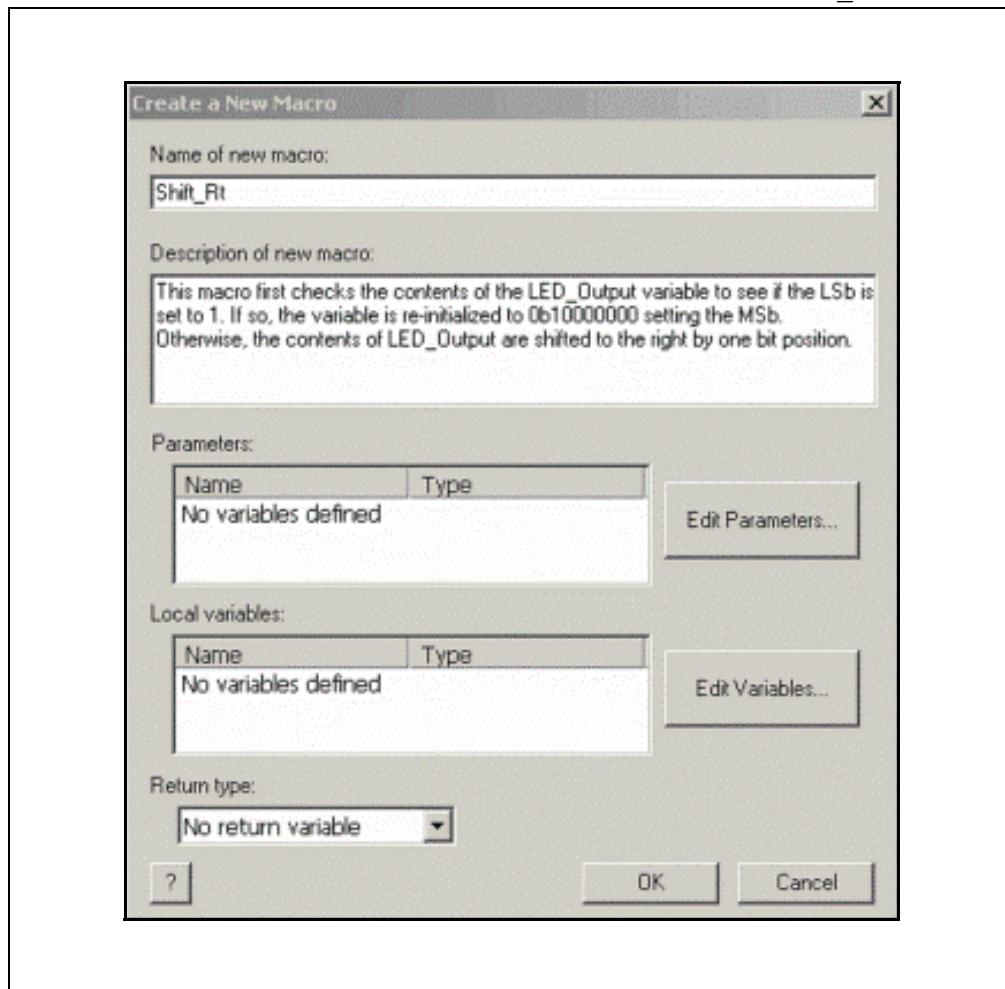
The macro called by the `Decide` functional block will first check the current value in `LED_Output` to see if the Least Significant bit (LSb), bit 0, is high. If so, this means that on the next shift to the right, the contents of the `LED_Output` variable will be all '0's. To avoid this condition, the macro re-initializes `LED_Output` to set the Most Significant bit (MSb), bit 7, high. Otherwise, if any other condition exists, the contents of the `LED_Output` variable are shifted by 1 bit position to the right.

### 2.3.6.2 PROCEDURE

1. Create a new project in Flowcode using steps 1 through 9 from `GPIO_Lab1` saving the project as `GPIO_Lab3.fcf` in the `C:\PICDEM_Lab\flowcode\GPIO_Labs\GPIO_Lab3` directory.
2. Construct the Software Control loop flowchart using steps 2 through 9 developed in the previous lab omitting the `Decide` functional block (step 6). In the `Initialize` block, initialize the `LED_Output` variable so that the MSb is high and the rest of the bits are low (i.e., `0b10000000`).
3. Next, the `Decide` functional block is constructed using the flowcode macro feature. Create a new macro by selecting Macro>New... The Create a New Macro window should open. In the "Name of new macro" box enter `Shift_Rt`. The "Description of new macro:" box describes what this macro does. Enter the description shown in Figure 2-31 or a description that clearly defines how this macro will behave and click **OK** to close the window.



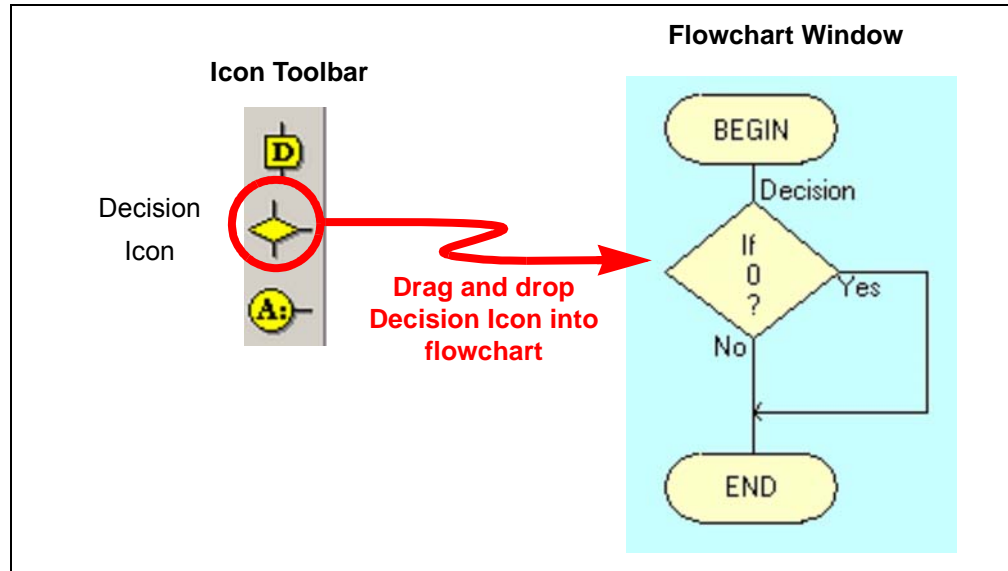
FIGURE 2-31: CREATE A NEW MACRO WINDOW FOR SHIFT\_RT



A new Flowchart window for the `Shift_Rt` macro should open displaying the description of the macro at the top of the window. The description of the macro can be edited by double-clicking on the text. More information on macros can be found in the Flow-code Help files.

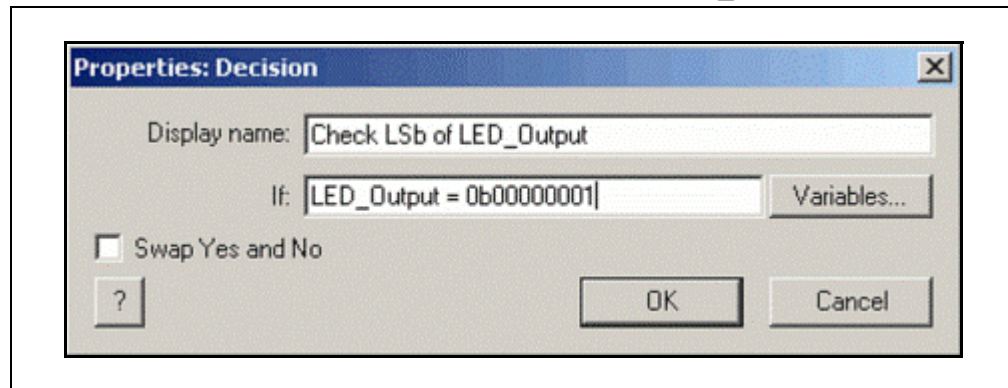
4. Next, the `Shift_Rt` macro's flowchart will be constructed. To check the contents of the `LED_Output` variable for the condition when the LSB is set, a **Decision** icon is used. Drag/drop a **Decision** icon from the icon toolbar into the macro flowchart between the BEGIN and END blocks (Figure 2-32).

FIGURE 2-32: ADDING A DECISION ICON



5. Open the **Decision** icon properties by double-clicking on the icon. Select the **Variables...** button next to the "If:" box. In the Variable Manager window, highlight the `LED_Output` variable and select **Use Variable** in the Properties: Decision window, change the contents in the If: window, as shown in Figure 2-33, to check if the LSb of `LED_Output` is high. Note that the "Display name:" has been changed to describe the action of the icon. This is good practice to ensure readability of the flowchart especially if someone other than the original programmer must view the code.

FIGURE 2-33: PROPERTIES: DECISION FOR `SHIFT_RT` MACRO

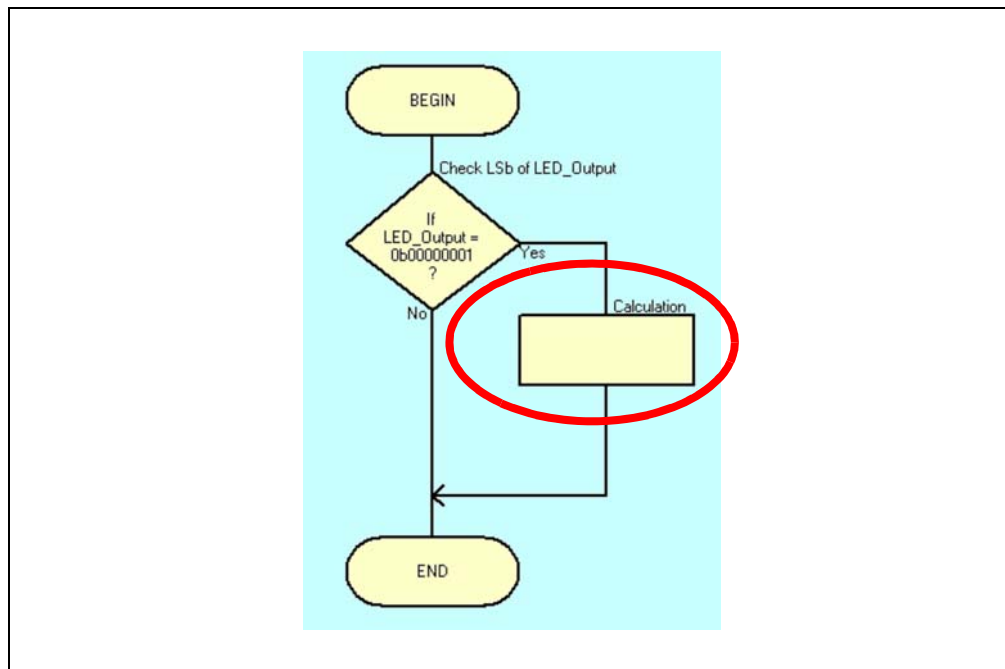


Click **OK** to close the window.

6. If the condition exists that the LSb of the `LED_Output` variable is set, the macro must then re-initialize the data to set the MSb. To accomplish this, drag/drop a **Calculation** icon onto the "Yes" branch of the **Decision** icon, as shown in Figure 2-34.

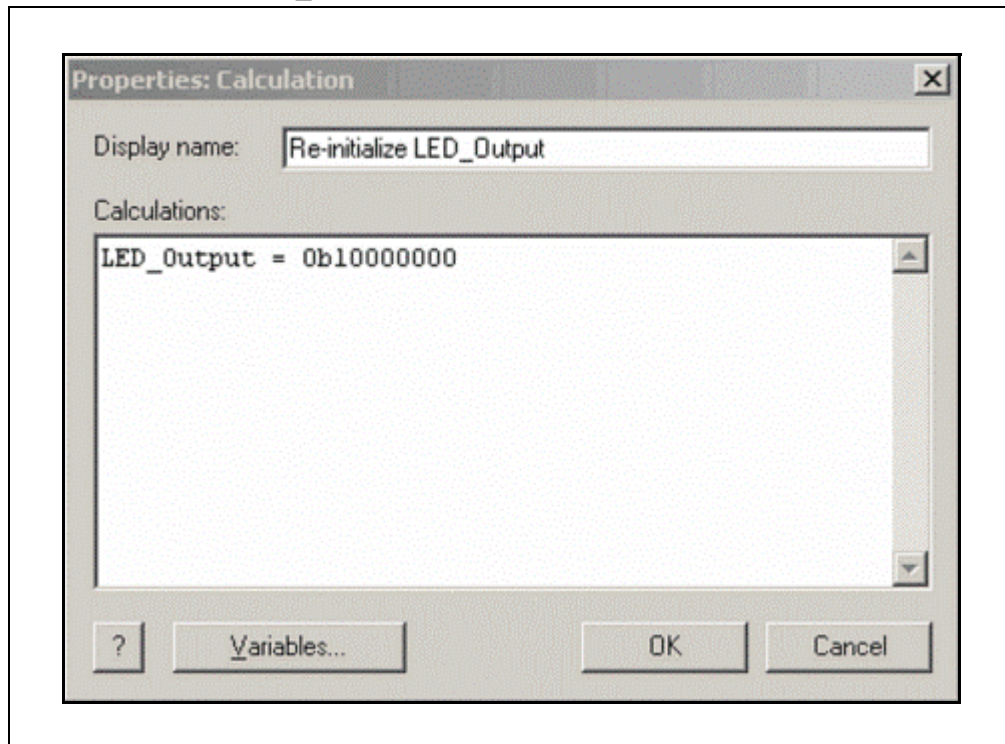
# General Purpose Input/Output Labs

**FIGURE 2-34: ADDING A CALCULATION ICON ONTO THE YES BRANCH OF THE DECISION ICON**



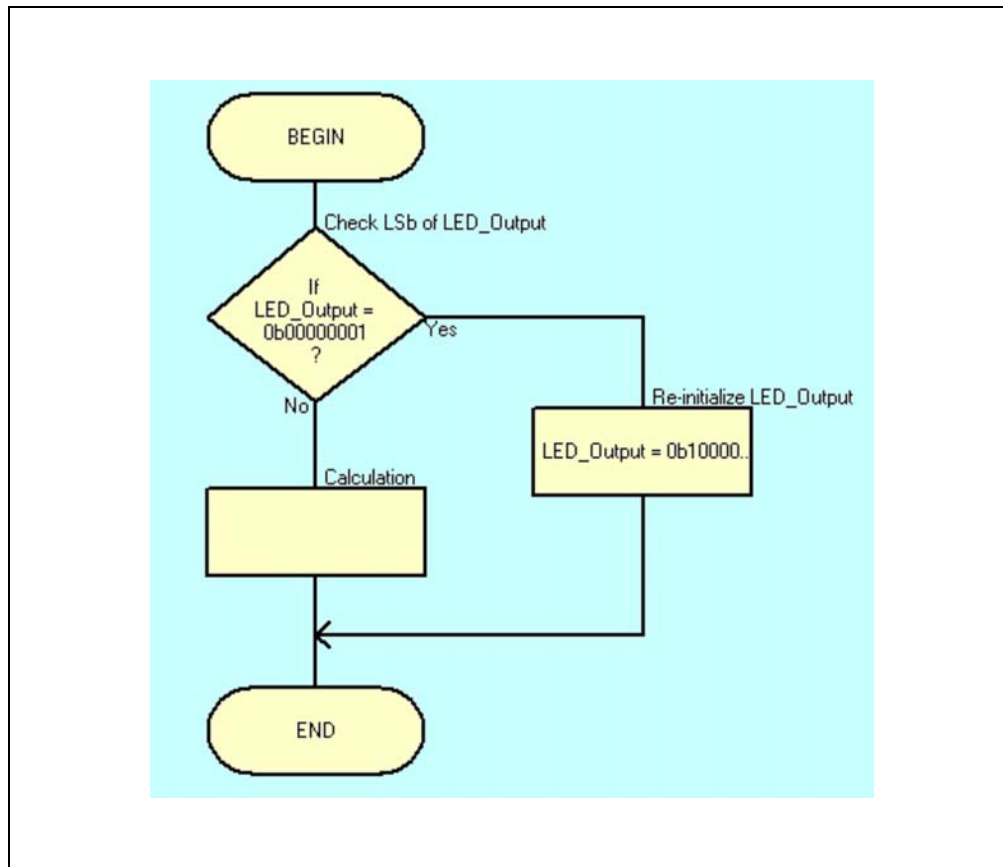
7. Open the Properties: Calculation window and configure, as shown in Figure 2-35.

**FIGURE 2-35: CONFIGURING THE CALCULATION ICON TO RE-INITIALIZE LED\_OUTPUT DATA**



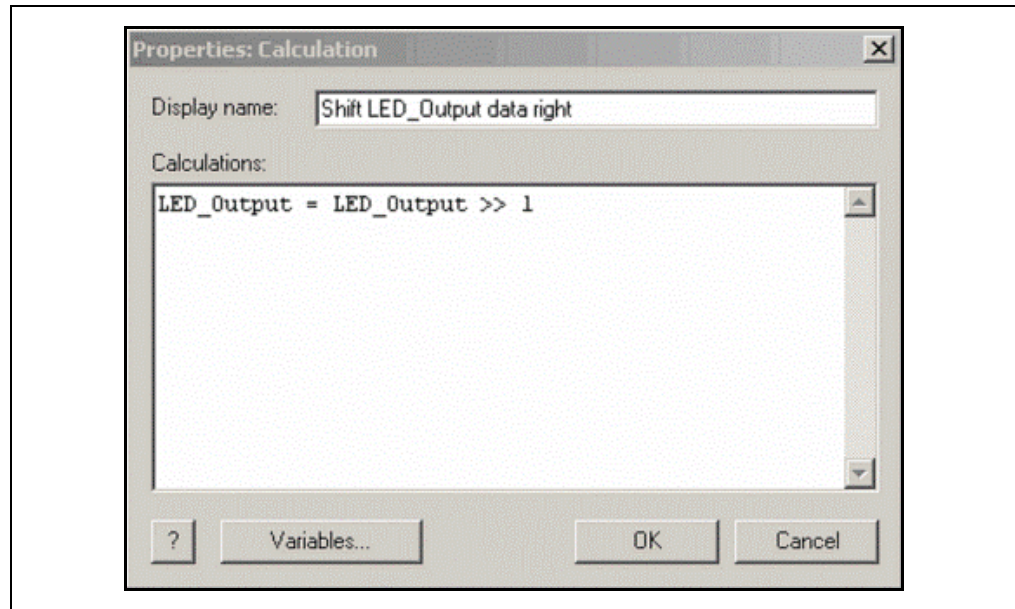
8. If the condition in which the LSb of `LED_Output` is not met, the macro then simply shifts all data one bit to the right. Drag/drop another **Calculation** icon onto the “No” branch of the **Decision** icon before the intersection with the arrowhead from the icon added in the previous steps (Figure 2-36).

**FIGURE 2-36: ADDING CALCULATION ICON TO THE NO BRANCH OF DECISION ICON**



9. Open the Properties: Calculation window and configure, as shown in Figure 2-37.

**FIGURE 2-37: CONFIGURING CALCULATION ICON TO SHIFT LED\_OUTPUT DATA RIGHT BY 1 BIT**



Note the right shift operator ">>". This code essentially says: **"Make LED\_Output equal to LED\_Output data shifted to the right by 1 bit position"**.

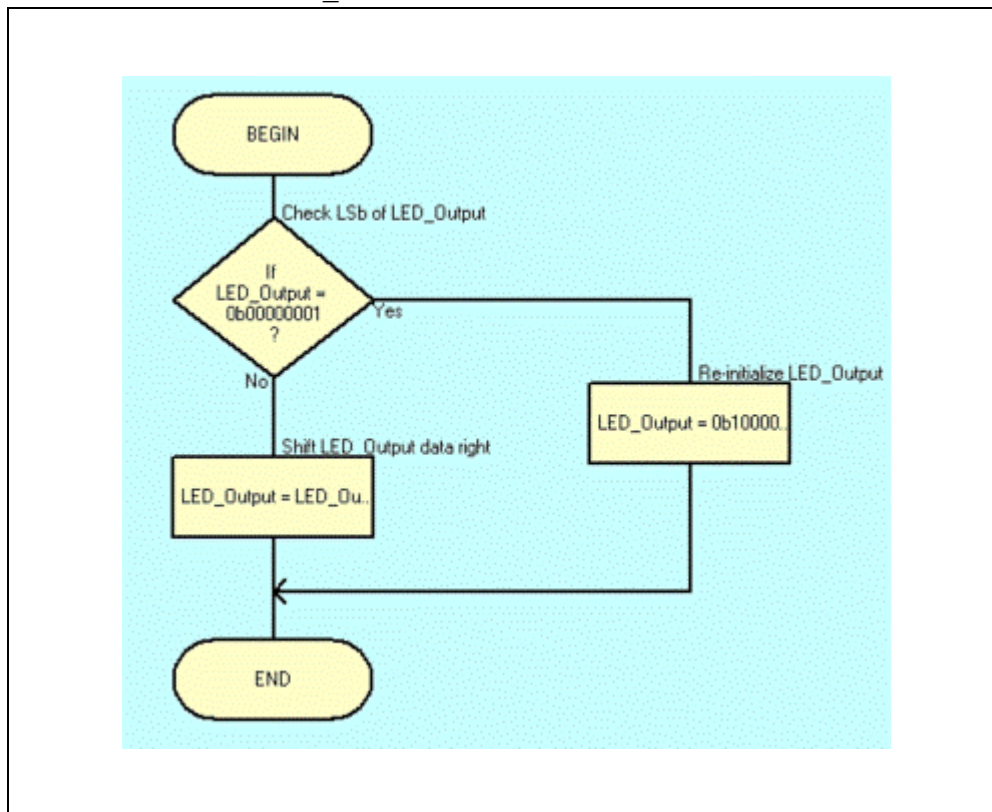
To shift the contents right by 2, the code would change to:

```
LED_Output = LED_Output >> 2
```

Alternately, to shift the contents of LED\_Output to the left, the left shift operator "<<" is used.

The Shift\_Rt macro is now completed. The flowchart should resemble Figure 2-38.

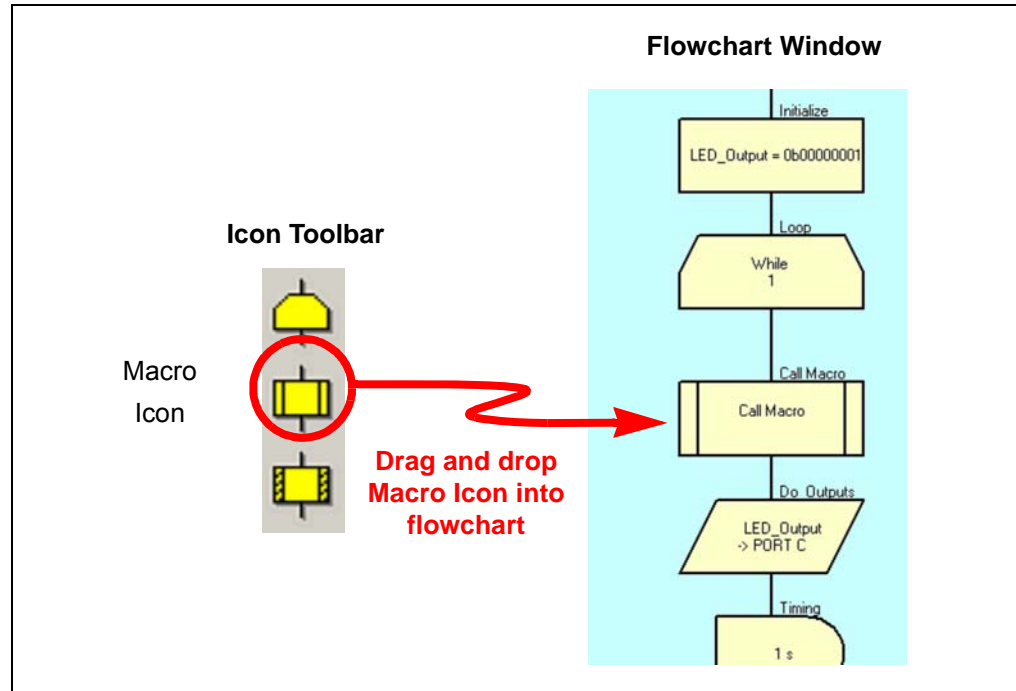
FIGURE 2-38: SHIFT\_RT MACRO FLOWCHART



10. Now that the macro is constructed, it will need to be called from the main flowchart. Return to the Main flowchart by clicking on it. To call the `Shift_Rt` macro, drag/drop a **Macro** icon from the icon toolbar and inset it into the main flowchart immediately before the `Do_Outputs` functional block (Figure 2-39).

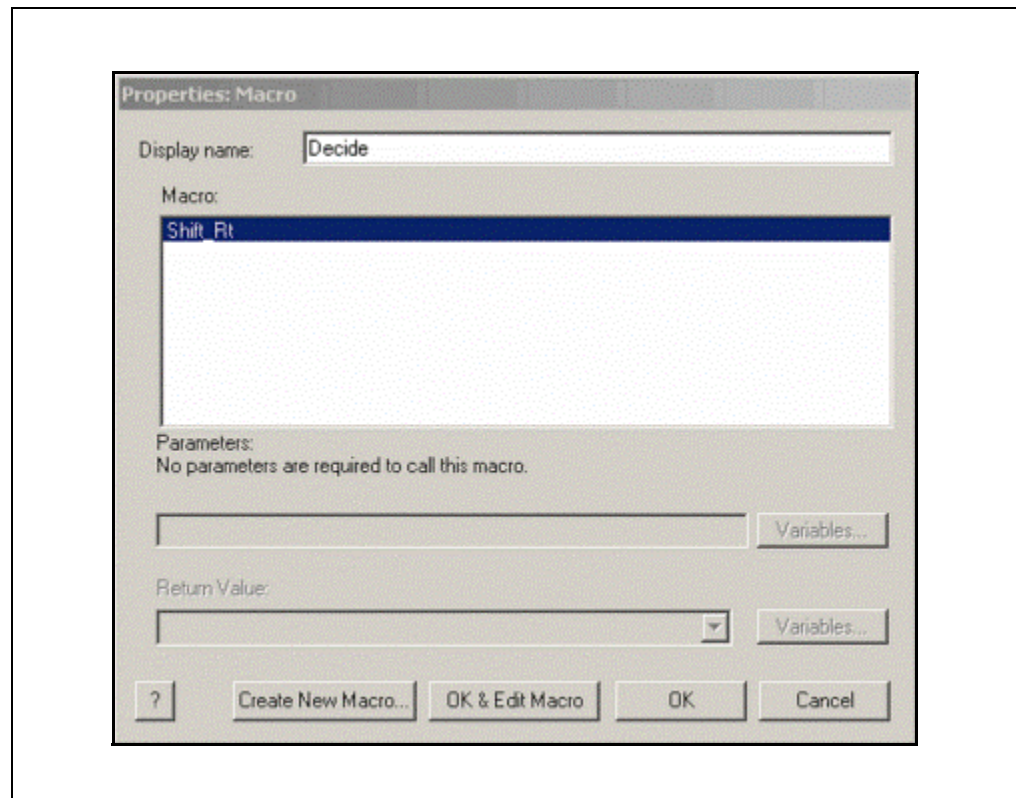


FIGURE 2-39: ADDING THE MACRO ICON



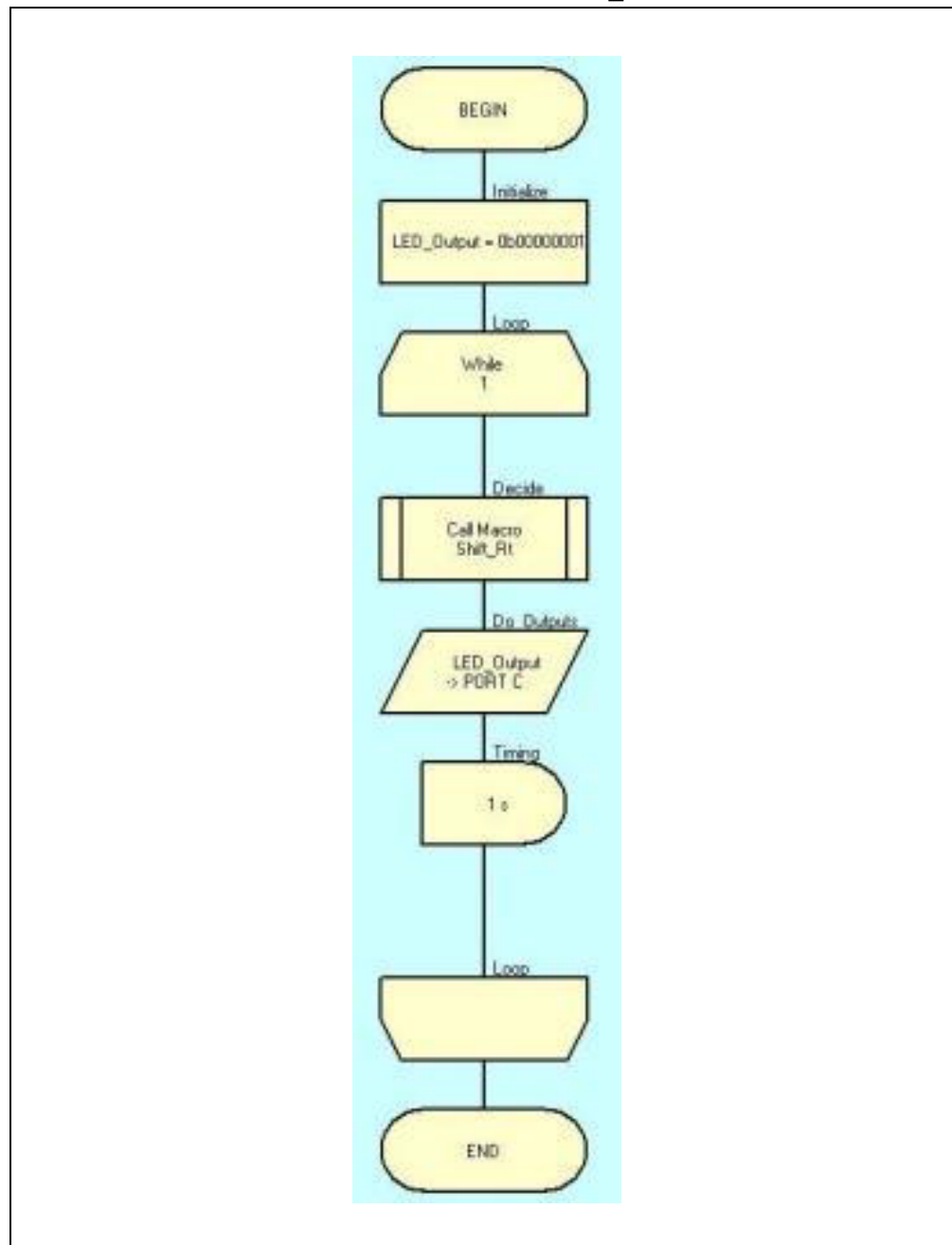
11. Open the Properties: Macro. All macros available for use by this project are displayed in the Macro: window. Select the `Shift_Rt` macro by clicking on it to highlight. Next, change the Display name: to `Decide` and click **OK** to close the window (Figure 2-40).

FIGURE 2-40: CONFIGURING THE MACRO ICON TO CALL THE `SHIFT_RT` MACRO



12. The final flowchart should now resemble Figure 2-41.

**FIGURE 2-41: FINAL FLOWCHART FOR GPIO\_LAB3**



Compile the project to chip. There should be no errors.

### 2.3.6.3 TESTING THE APPLICATION

The LEDs connected to the individual PORTC pins should now flash ON/OFF sequentially from left-to-right in 1-second intervals.

## 2.4 GPIO INPUT LABS

### 2.4.1 Reference Documentation

Free support documentation from [www.microchip.com](http://www.microchip.com):

- DS41262D – PIC16F690 data sheet
  - Section 2.2.2.2: Option Register
  - Section 2.2.2.3: Interrupt Control Register INTCON
  - Section 4: I/O Ports

On-line support for Flowcode Software:

- Forums: [www.matrixmultimedia.com/mmforums/](http://www.matrixmultimedia.com/mmforums/)
- Main web site including Frequently Asked Questions: [www.matrixmultimedia.com](http://www.matrixmultimedia.com)

### 2.4.2 Equipment Required for GPIO Input Labs

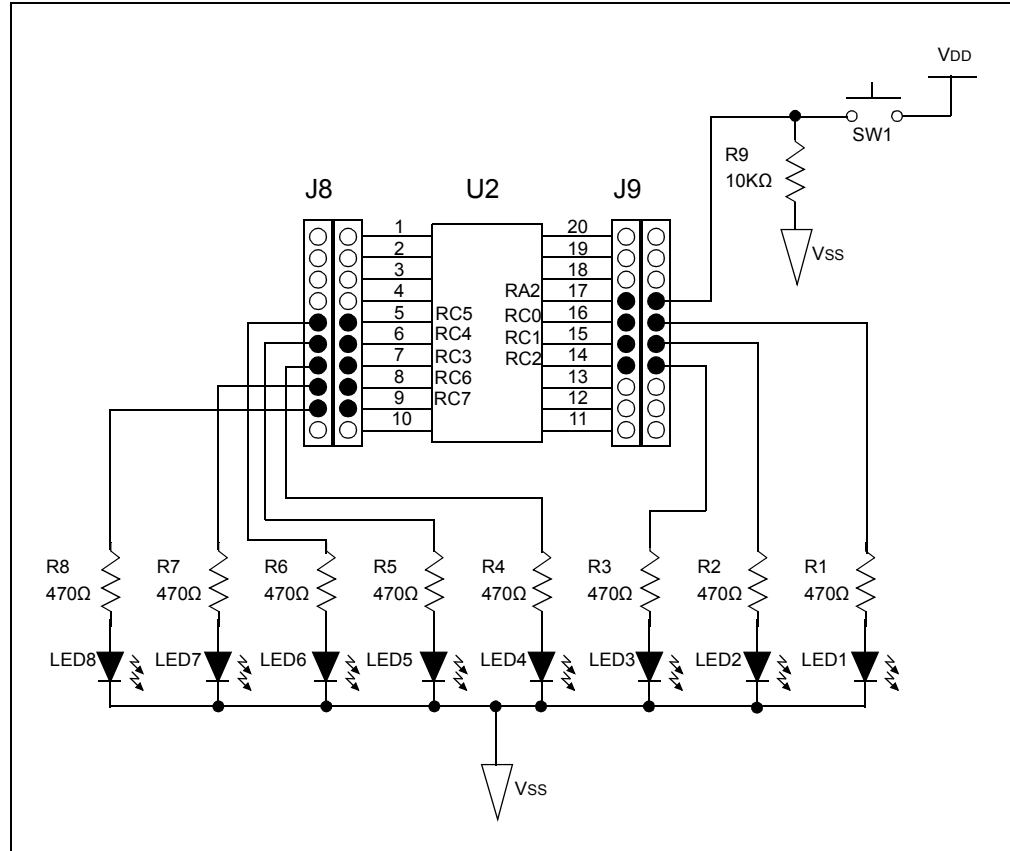
To complete the labs in this section, the following components are required:

1. 1 – Push button
2. 8 – Light Emitting Diodes
3. 1 – 10 K $\Omega$
4. 8 – 470 $\Omega$  resistors
5. PIC16F690 populating socket U2
6. Assorted jumper wires

### 2.4.3 PICDEM Lab Development Board Setup for GPIO Input Labs

The GPIO input labs will require that the PICDEM Lab Development Board be configured as shown in Figure 2-42, using the components listed in the previous section.

**FIGURE 2-42: PICDEM LAB SCHEMATIC FOR GPIO INPUT LABS**



The only change from the previous section is the inclusion of a push button connected to RA2 with associated pull-down resistor.

## 2.4.4 Lab 4: Adding a Push Button Interface using Component Macros

### 2.4.4.1 NEW REGISTERS USED IN THIS LAB

To configure the peripherals used in this lab, the following registers are used:

1. PORTA Register: PORTA (Register 4-1 in Section 4 of the PIC16F690 data sheet)
  - 8-bit bidirectional port

These registers are configured in the background automatically by the Flowcode Software and do not require any user manipulation.

2. PORTA Tri-State Register: TRISA (Register 4-2 in Section 4 of the PIC16F690 data sheet)
  - Configures corresponding bits in PORTC as either input or output
3. Analog Select Register High and Analog Select Register Low: ANSELH and ANSEL (Registers 4-4 and 4-3 in Section 4 of the PIC16F690 data sheet)
  - Configure associated pins for analog or digital input signals

### 2.4.4.2 OVERVIEW

This lab expands upon previous labs by adding a push button interface to change the direction of the sequential shift in the PORTC register.

Mechanical switches play an important and extensive role in practically every computer, microprocessor and microcontroller application. Mechanical switches are inexpensive, simple and reliable. However, switches can be very noisy electrically. The apparent noise is caused by the closing and opening action that seldom results in a clean electrical transition. The connection makes and breaks several, perhaps even hundreds, of times before the final switch state settles. The problem is known as switch bounce. This results in a period of time where the state of the I/O pin connected to the push button will change states high/low numerous times before this bouncing subsides. Some of the intermittent activity is due to the switch contacts actually bouncing off each other. Also, switch contacts are not perfectly smooth. As the contacts move against each other, the imperfections and impurities on the surfaces cause the electrical connection to be interrupted. The result is switch bounce. The consequences of uncorrected switch bounce can range from being just annoying to catastrophic. The classic solution involved filtering, such as through a resistor-capacitor circuit, or through resettable shift registers. These methods are still effective, but they involve additional cost in material, installation and board real estate. Debouncing in software eliminates these additional costs.

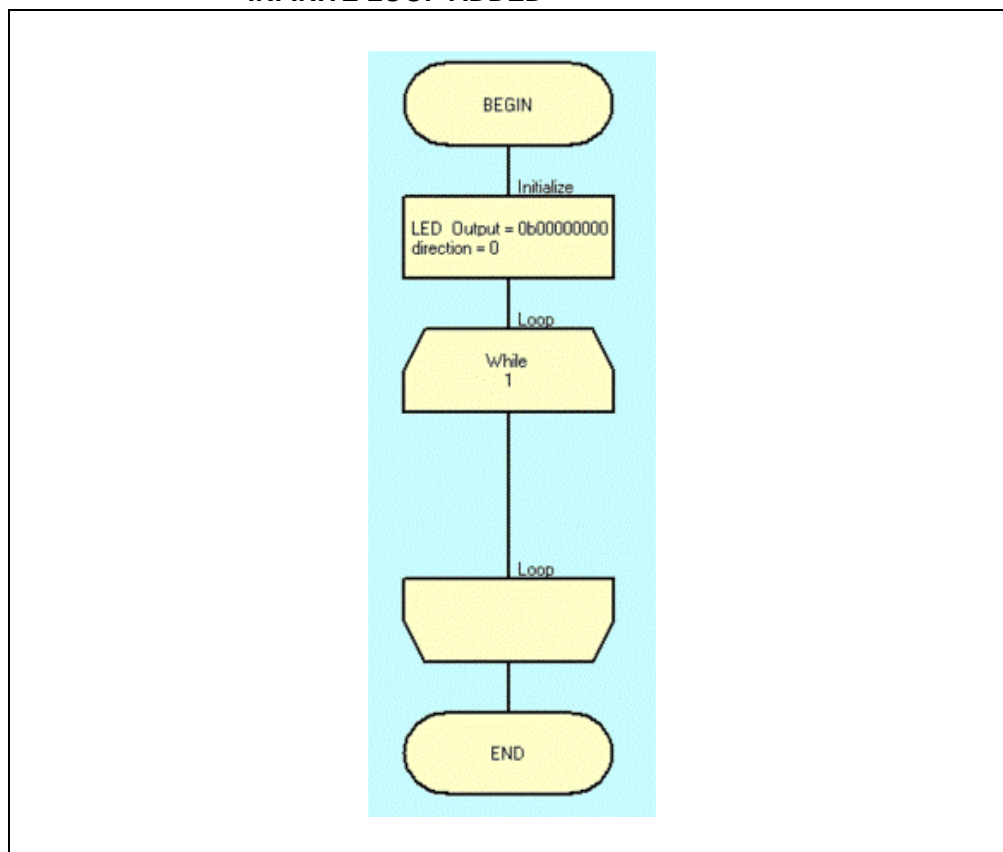
One of the simplest ways to debounce a switch is to sample the switch until the signal is stable or continue to sample the signal until no more bounces are detected. How long to continue sampling requires some investigation. However, 5mS is usually adequate, while still reacting fast enough that the user won't notice it.

As an example, in this application the pin connected to the push button is pulled low using a 10 K $\Omega$  resistor tied to Vss (refer to Figure 2-42). The pull-down resistor ensures that the pin state will remain low whenever the push button is released, eliminating state transitions due to noise. Once the push button is pressed, the pin is shorted to VDD resulting in a high state. The firmware then implements a 5 mS delay to give any bouncing at the push button to settle. After the delay, the firmware once again checks the pin state. If the state is still high then a push button press is confirmed and the firmware reacts accordingly. Otherwise, the push button is deemed "Not Pressed".

#### 2.4.4.3 PROCEDURE

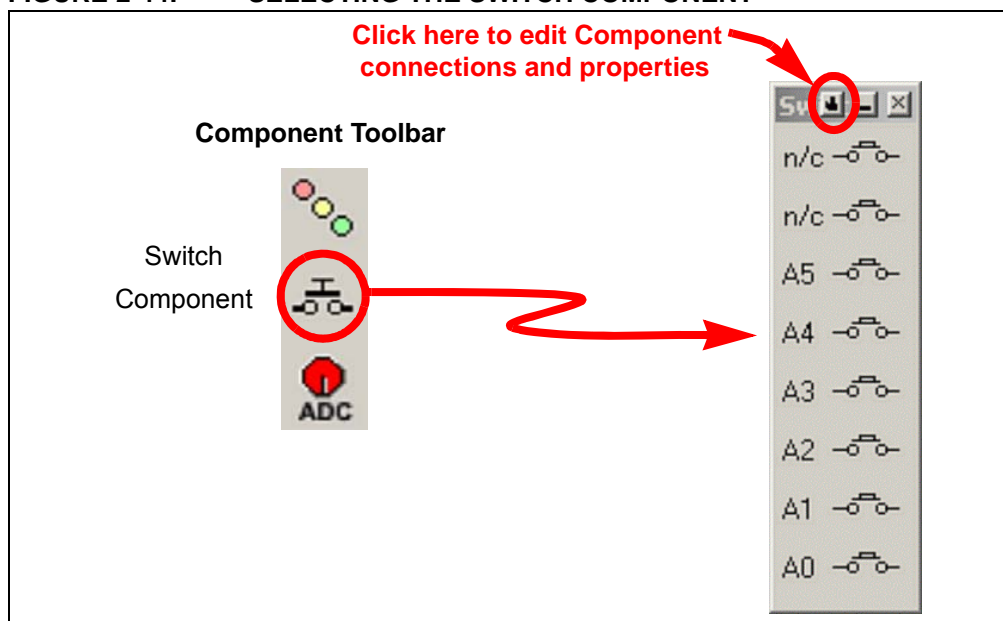
1. Create a new project in Flowcode using steps 1 through 9 from GPIO\_Lab1 saving the project as `GPIO_Lab4.fcf` in the `C:\PICDEM_Lab\flowcode\GPIO_Labs\GPIO_Lab4` directory.
2. Create the `Initialize` block using the **Calculation** component and create/initialize the `LED_Output` variable as per the previous labs initializing data to all 0's (`0b00000000`). Create/initialize an additional 8-bit variable named `direction` to '0' as well.
3. Immediately following the `Initialize` block, add the loop component to implement the Infinite loop discussed in the previous labs. The flowchart should now resemble Figure 2-43.

**FIGURE 2-43: GPIO\_LAB4 FLOWCHART WITH INITIALIZE BLOCK AND INFINITE LOOP ADDED**



4. This application will use the PORTA <RA2> I/O pin as the push button interface. The switch component will be used to configure the interface. Select the switch component from the component toolbar by clicking on it (Figure 2-44).

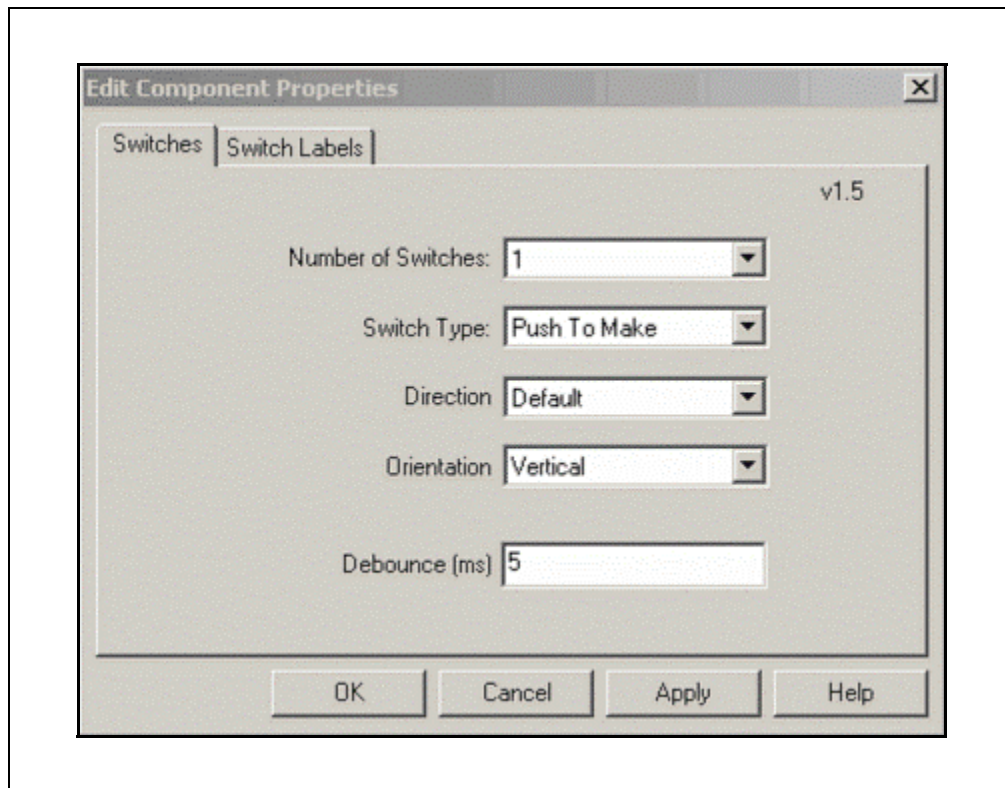
**FIGURE 2-44: SELECTING THE SWITCH COMPONENT**





- The switch component automatically defaults to connect to the PORTA register pins. However, this application will only make use of a single switch. Therefore, the switch component will need to be modified. Click on the small button on the component, as shown in Figure 2-44, and select **Properties...** from the drop-down menu. In the Edit Component Properties window, change the “Number of Switches:” to one using the drop-down menu. Flowcode features the ability to set-up a “Debounce(ms)” for any push buttons/switches used in the application, thereby minimizing the amount of code the user needs to create. As per the previous discussion, set the “Debounce(ms)” to 5mS (see Figure 2-45).

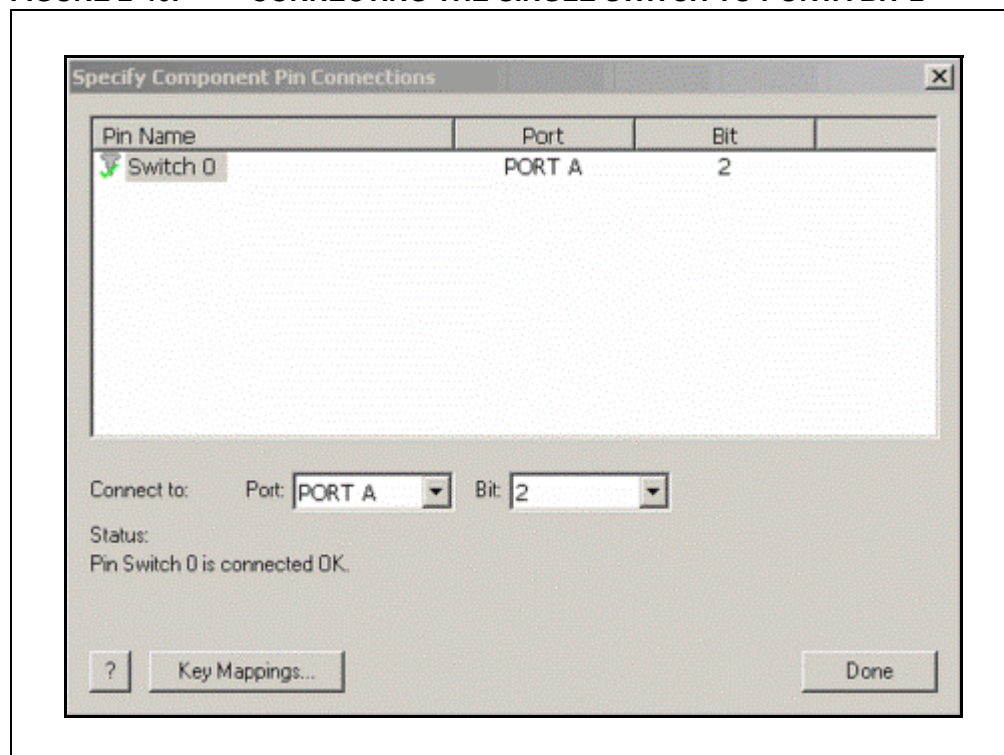
**FIGURE 2-45: CONFIGURING THE NUMBER OF SWITCHES AND DEBOUNCE TIME**



Click **Apply** and then **OK** to close the window.

- The switch component should now only show one switch. This switch is connected to PORTA<RA0>. To configure the switch to be connected to the RA2 pin, once again select the properties drop-down menu as per step 5, only this time selecting **Component Connections...** In the Specify Component Pin Connections window, select **Bit: 2** from the appropriate drop-down menu. The window should now resemble Figure 2-46.

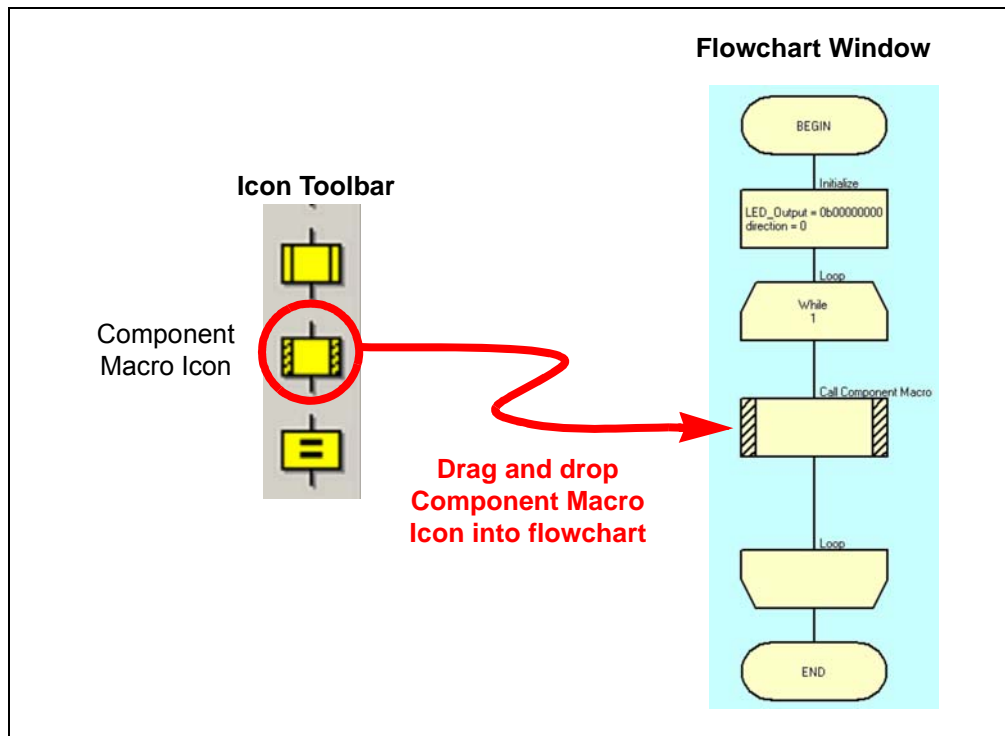
FIGURE 2-46: CONNECTING THE SINGLE SWITCH TO PORTA BIT 2



Click **Done** to close the window.

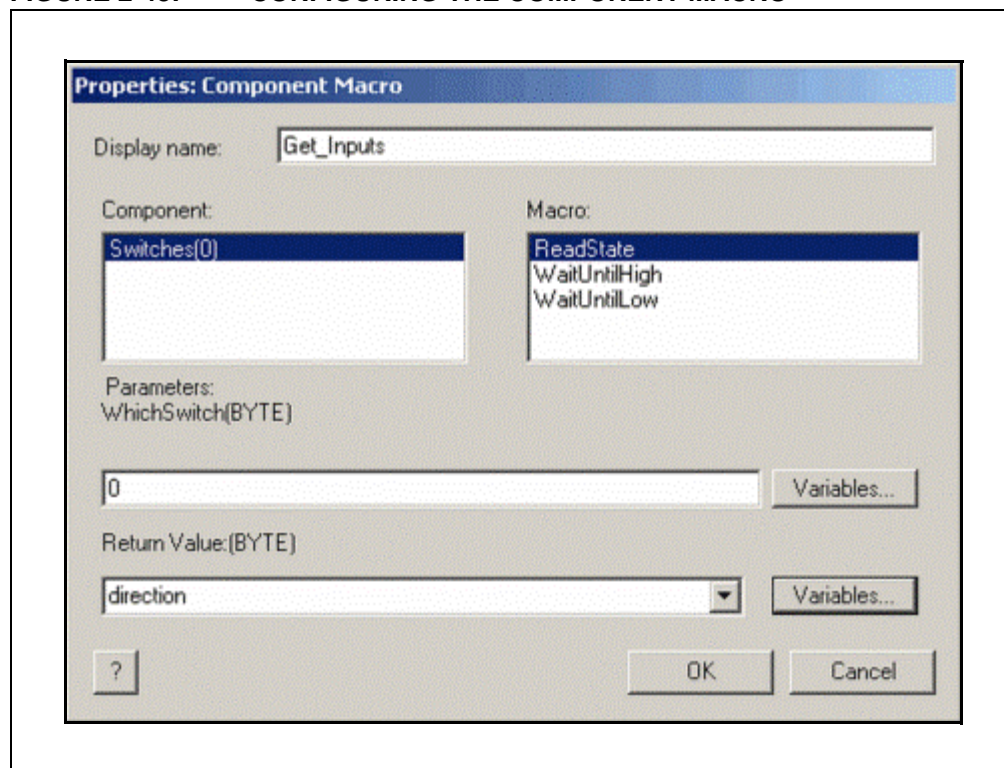
7. Next, the `Get_Inputs` functional block will be added to obtain the current state of the switch (i.e., either high for pressed or low for unpressed). Flowcode has a variety of pre-defined macros available for components such as the switch component added in the previous steps. To access these pre-defined macros, the **Component Macro** icon is used. Drag/drop the **Component Macro** icon from the icon toolbar to the flowchart within the While loop (see Figure 2-47).

FIGURE 2-47: ADDING A COMPONENT MACRO



8. Double-click on the component macro to open the Properties: Component Macro window. Change the "Display name:" to `Get_Inputs`. In the Component window, select **Switches(0)** to select the switch and then `ReadState` in the Macro: window. The `ReadState` macro reads the state of a single switch, specified by the BYTE parameter `WhichSwitch`. The macro returns a BYTE value of '0' if the switch is open or '1' if the switch is closed. In the Parameters: window specify '0' as the switch to be read, and in the Return Value:(BYTE) window copy the `direction` variable initialized in step 2 of this lab. The window should now resemble Figure 2-48.

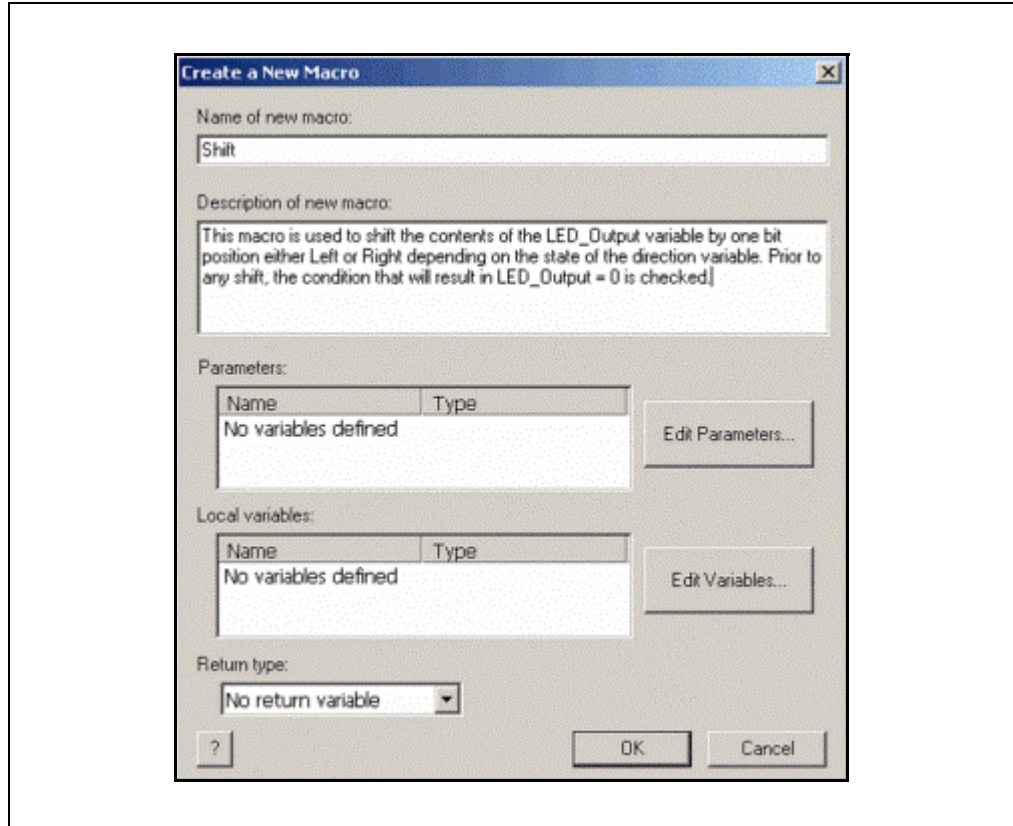
FIGURE 2-48: CONFIGURING THE COMPONENT MACRO



Click **OK** to close the window.

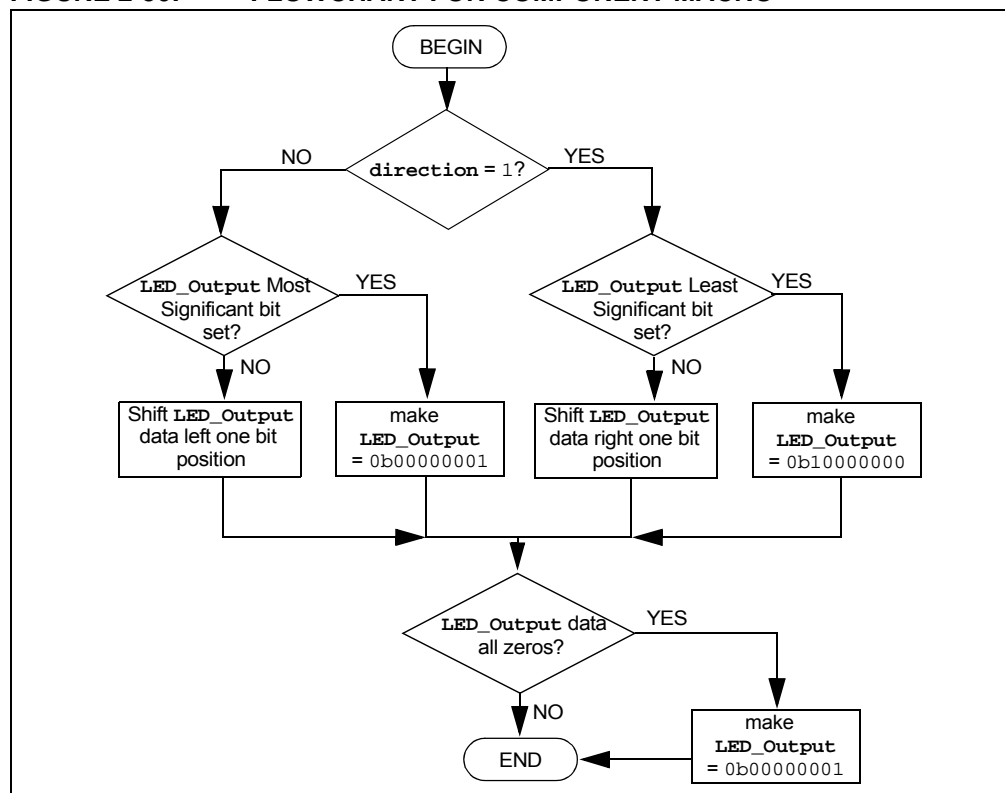
9. Similar to the previous lab, the Decide functional block will be constructed using a macro. Create a new macro as was done in step 3 of the previous lab. Name the macro *Shift* and provide a "Description of new macro:" as shown in Figure 2-49 or some other description equally as revealing as to the purpose of the macro.

FIGURE 2-49: CREATE A NEW MACRO WINDOW FOR LAB 4

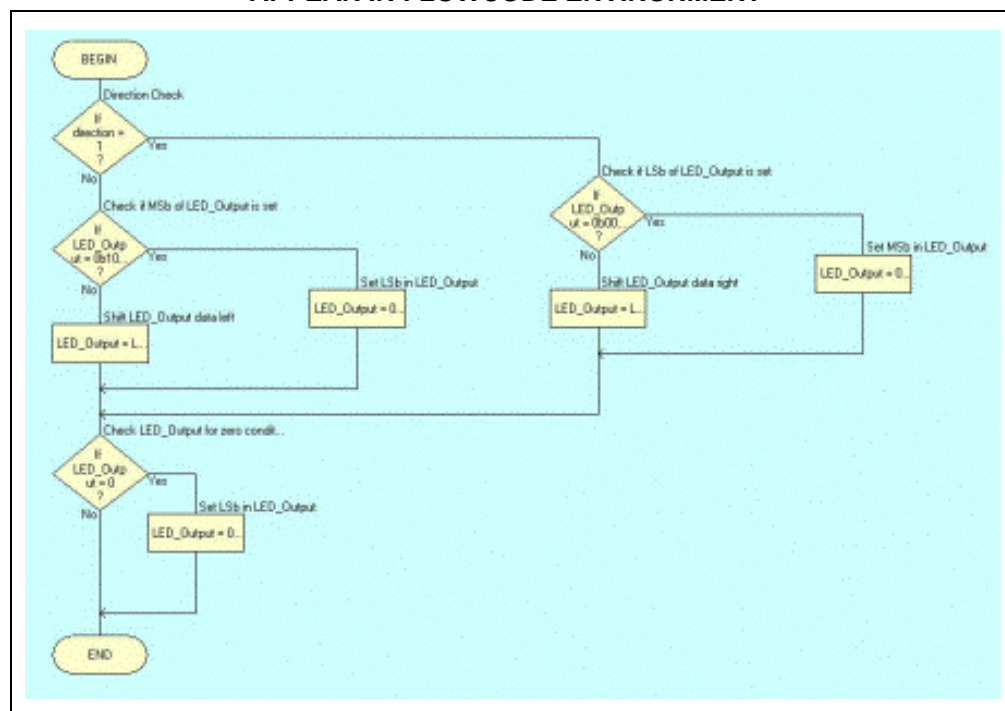


10. This macro will first check if the `direction` variable is high indicating a push button press. If so, the data in `LED_Output` will be shifted left by one bit position. If `direction` is low the data in `LED_Output` is shifted right by one bit position. As in the previous lab, the condition in which a shift (left or right) will fill the `LED_Output` variable with zeros is first checked. Finally, the last check in the macro will test for the condition that the `LED_Output` is all zeros (as would be the case the first time through the loop) and if so, re-initializes the `LED_Output` variable to `'0b00000001'`. The complete flowchart is shown in Figure 2-50 for clarity and the actual flowchart as should be seen in the Flowcode workspace shown in Figure 2-51. The reader should refer to the previous lab sections on how to add each icon in the component macro.

**FIGURE 2-50: FLOWCHART FOR COMPONENT MACRO**



**FIGURE 2-51: FLOWCHART FOR COMPONENT MACRO AS IT SHOULD APPEAR IN FLOWCODE ENVIRONMENT**

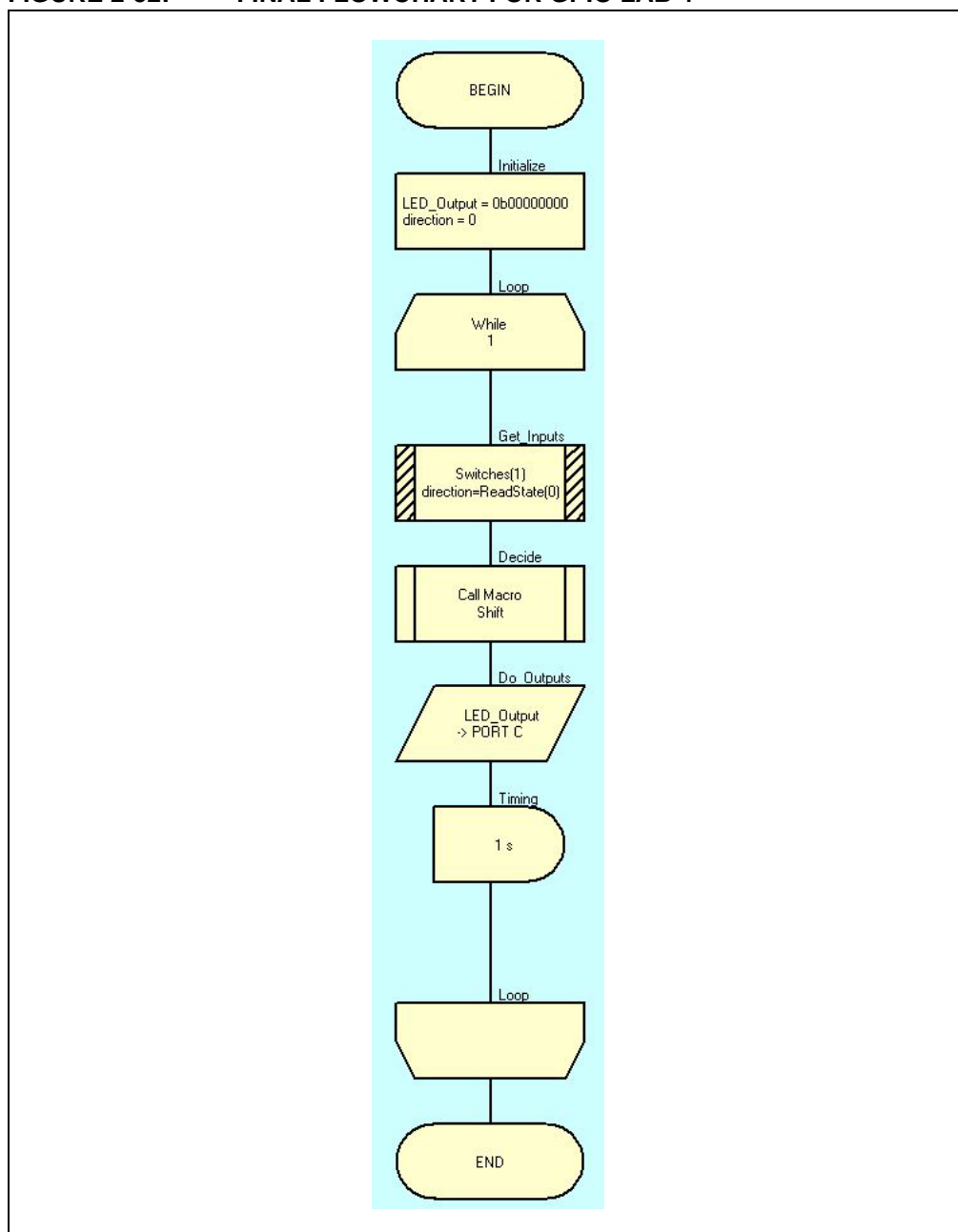




# General Purpose Input/Output Labs

11. Return to the main flowchart and add a **Macro** icon just beneath the `Get_Inputs` functional block. Change the display name to `Decide` and call the `Shift` macro created in the previous step (see step 10 in the previous lab as a reference).
12. Add a new **Output** icon beneath the `Decide` functional block added in the previous step changing the display name to `Do_Outputs` and assign the contents of `LED_Output` to `PORTC`.
13. Finally, add a **Delay** icon to the flowchart just beneath the `Do_Outputs` functional block changing the display name to `Timing` and configuring the properties to perform a 1-second delay.
14. The main flowchart should now resemble Figure 2-52.

**FIGURE 2-52: FINAL FLOWCHART FOR GPIO LAB 4**



Compile the project to chip. There should be no errors.

## 2.4.4.4 TESTING THE APPLICATION

The LEDs connected to the individual PORTC pins should flash ON/OFF sequentially from right-to-left in 1-second intervals while the push button is released. Pressing the push button will change the ON/OFF flashing to left-to-right.

## 2.4.5 Lab 5: Adding an Interrupt and Importing Macros for a Push Button Interface

### 2.4.5.1 NEW REGISTERS USED IN THIS LAB

To configure the peripherals used in this lab, the following registers are used:

1. Interrupt Control Register: INTCON (Register 2-3 in Section 2 of the PIC16F690 data sheet)
  - This register contains various enable and flag bits for a variety of peripheral interrupts including:
    - TMR0 overflow
    - PORTA interrupt-on-change and external RA2/INT pin interrupt.

<b>Note:</b> Flowcode will automatically configure these bits in the background so that the user does not have to.
--

### 2.4.5.2 OVERVIEW

This lab expands upon the previous lab by adding a push button interrupt that will handle the modification to the direction variable that determines the direction of the sequential shift of the flashing LEDs connected to the PORTC pins. The `Shift` macro created in the previous lab will also be used by importing the macro into the current Flowcode project. In the previous lab, the `PORTA<RA2>` was checked each time through the Software Control loop for a change in voltage level signifying a push button press. This is a method known as “Polling”. Polling a bit is a common method to check the status of a register bit. However, checking a bit in this manner can only be done when the Software Control loop execution reaches that point in firmware. Sometimes this may not be sufficient to the application which brings up the use of interrupts.

Interrupts provide a means of signalling the processor that a task must be performed immediately. Imagine an emergency shut-off switch on a piece of machinery. In this scenario, the machinery must respond without delay when the switch is activated. To accomplish this task, assuming a microcontroller is integrated into the application, an interrupt is used. When the Central Processing Unit (CPU) on the microcontroller receives notification of an interrupt, it immediately stops whatever it is doing and “services” the interrupt. Servicing the interrupt involves executing user defined code that has been associated with a particular interrupt in firmware. Using the emergency shut-off switch example, the CPU may currently be executing code necessary in the operation of the machinery. When the emergency shut-off switch is pressed, the CPU receives the interrupt notification and immediately stops its current task to execute the interrupt specific code that will stop the machinery.

Typically, each peripheral on a PIC® device will have an interrupt associated with it that is triggered due to a specific event. The GPIO peripherals on the PIC16F690 have a number of interrupts that can be triggered via a number of events such as a change in the voltage level on a particular port pin as would be the case if a push button connected to that pin were pressed. Other events that trigger interrupts associated with different peripherals on the PIC16F690 include a change on the output of the comparator peripheral or the completion of a conversion using the Analog-to-Digital Converter (ADC), to name but a few.

# General Purpose Input/Output Labs

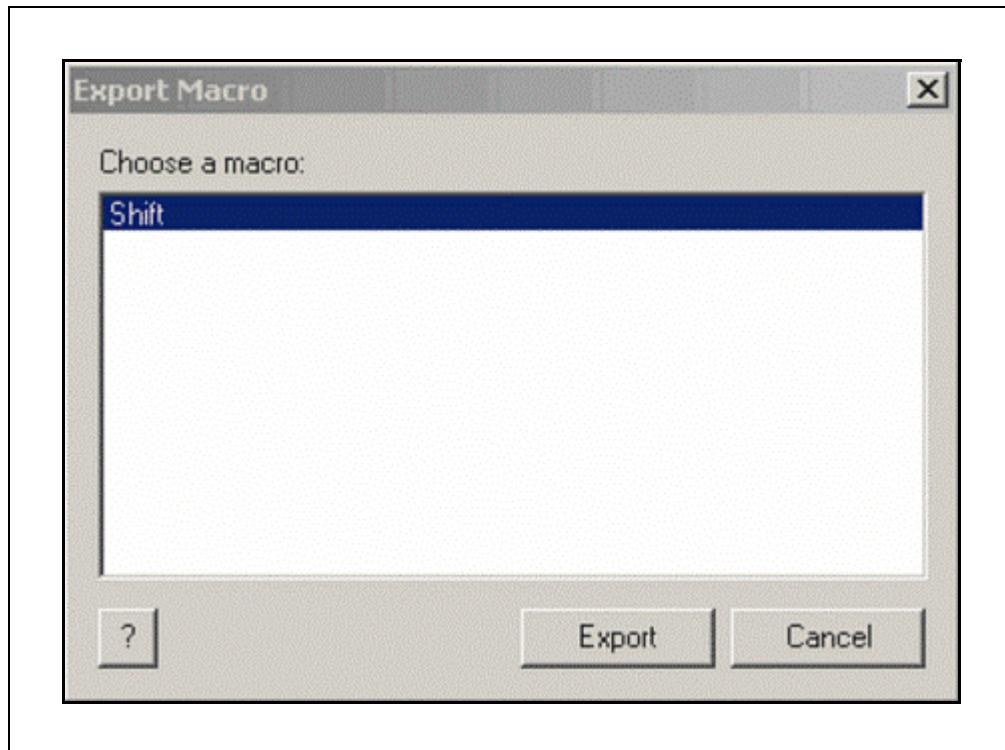
So the question may arise, why not use interrupts all the time? The answer to this is cost. In order to have an interrupt for a particular event usually requires a specific peripheral. The more peripherals on a microcontroller, the higher the cost of the device. Many times simply checking an event each time through the Software Control loop may be sufficient for the application at hand minimizing pin-count and ultimately the costs associated.

This lab will incorporate an interrupt on the PORTA RA2 pin interfaced the push button used in the circuit from the previous lab. Referring to the PIC16F690 pin diagram in the PIC16F690 data sheet, the RA2 pin shares functionality with a feature designated as INT. The INT designation refers to the external edge-triggered interrupt capability on this particular pin. This interrupt can be triggered by either a high-to-low or low-to-high transition of the voltage level on the RA2 pin. When the particular event occurs, and the external interrupt is configured, the CPU will then execute a macro that will be created within Flowcode specific to that interrupt. This interrupt macro will contain the code to toggle the `direction` variable used in the previous lab by the `Decide` functional block to determine the direction of the sequential flashing of the LEDs connected to the PORTC pins.

## 2.4.5.3 PROCEDURE

1. The first step will be to export the `Shift` macro from the previous lab so that it can be used here. Ensure that the `GPIO_Lab4.fcf` project is open from the previous lab. To export the `Shift` macro select **Macro>Export** to open the Export Macro dialog. Highlight the `Shift` macro and click **Export** (see Figure 2-53).

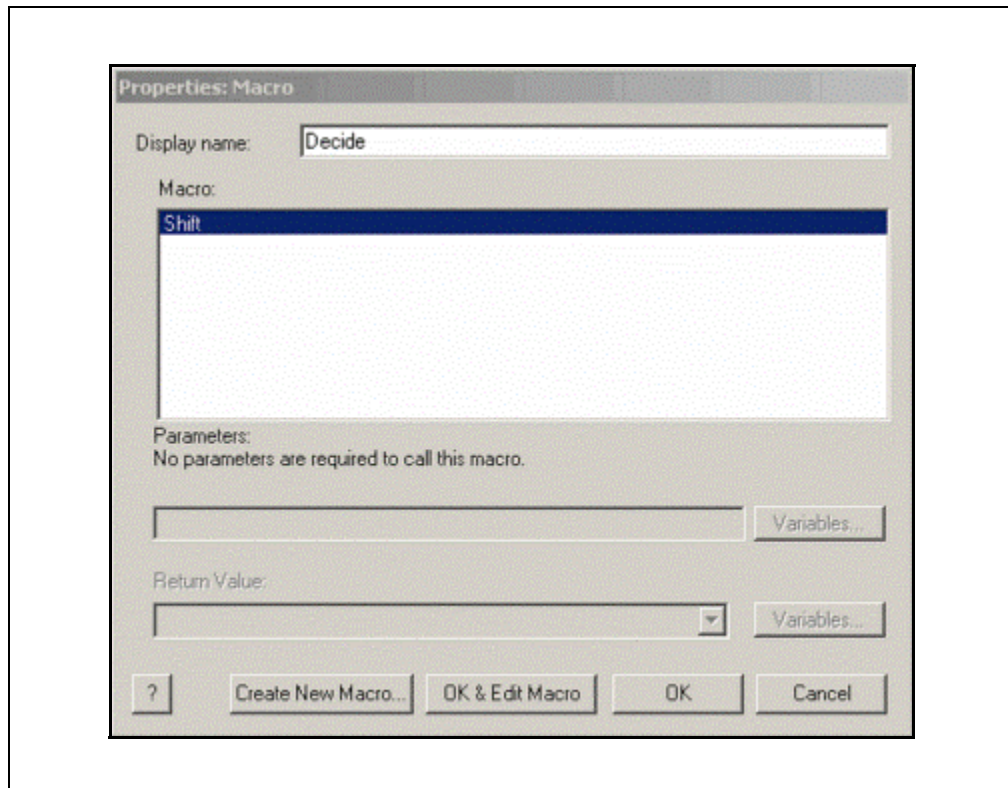
**FIGURE 2-53: EXPORT MACRO WINDOW**



2. The user will be prompted to select a directory into which to save the `Shift` macro. Navigate to the `C:\PICDEM_Lab\flow-code\GPIO_Labs\GPIO_Lab5` directory and save the macro there.

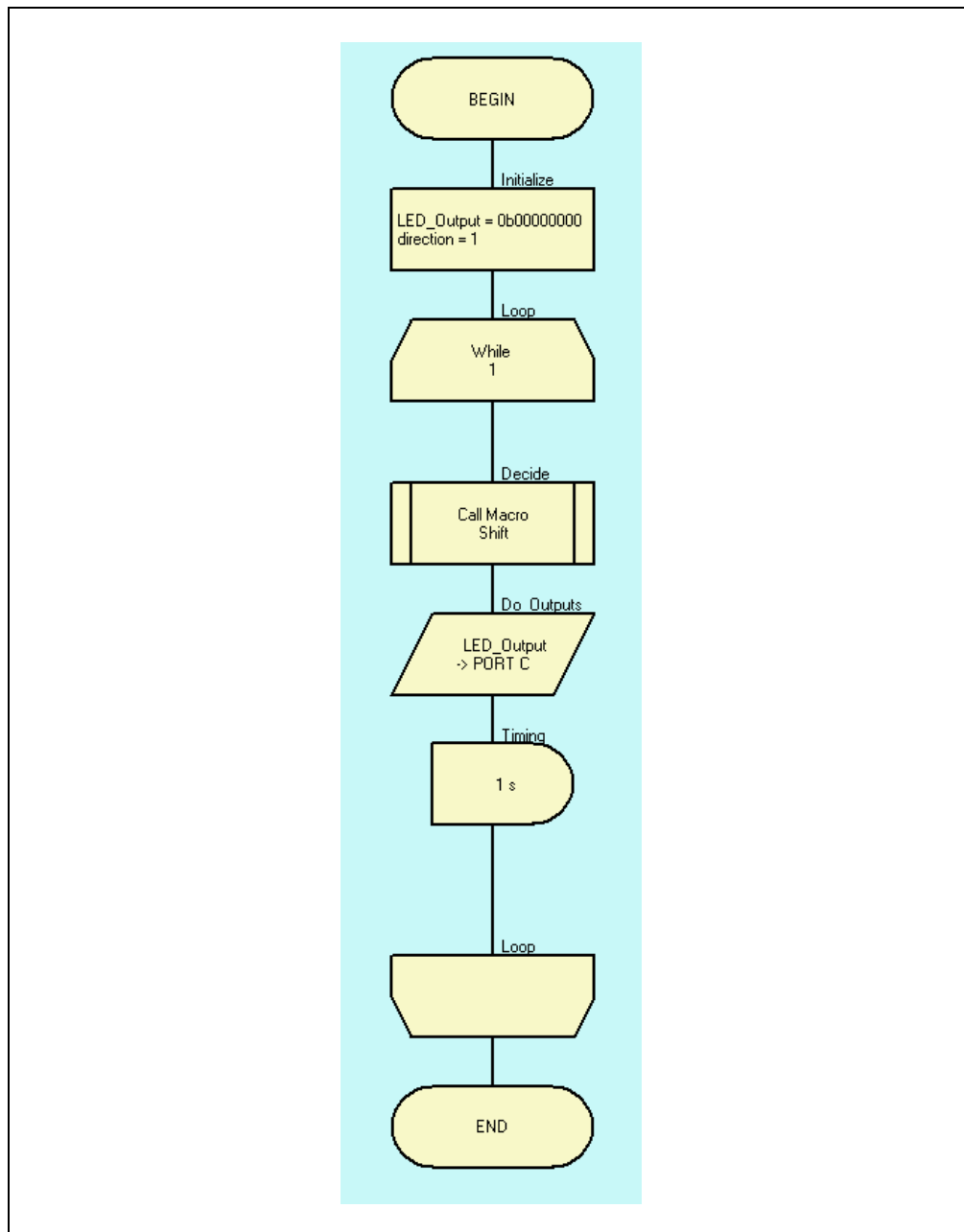
3. Close the `GPIO_Lab4.fcf` project and create a new project in Flowcode using steps 1 through 9 from `GPIO_Lab1` saving the project as `GPIO_Lab5.fcf` in the `C:\PICDEM_Lab\flowcode\GPIO_Labs\GPIO_Lab5` directory.
4. Create the `Initialize` block using the **Calculation** component and create/initialize both the `LED_Output` and `direction` variables to 0 as per the previous lab.
5. Immediately following the **Initialize** block, add the loop component to implement the Infinite loop discussed in the previous labs.
6. Next, within the `While` loop, add a **Macro** icon, rename the macro to `Decide` and call the `Shift` macro that was exported from Lab 4 earlier by highlighting it in the Macro window and clicking **OK** (see Figure 2-54).

**FIGURE 2-54: CONFIGURING THE MACRO PROPERTIES TO CALL THE SHIFT MACRO CREATED IN LAB 4**



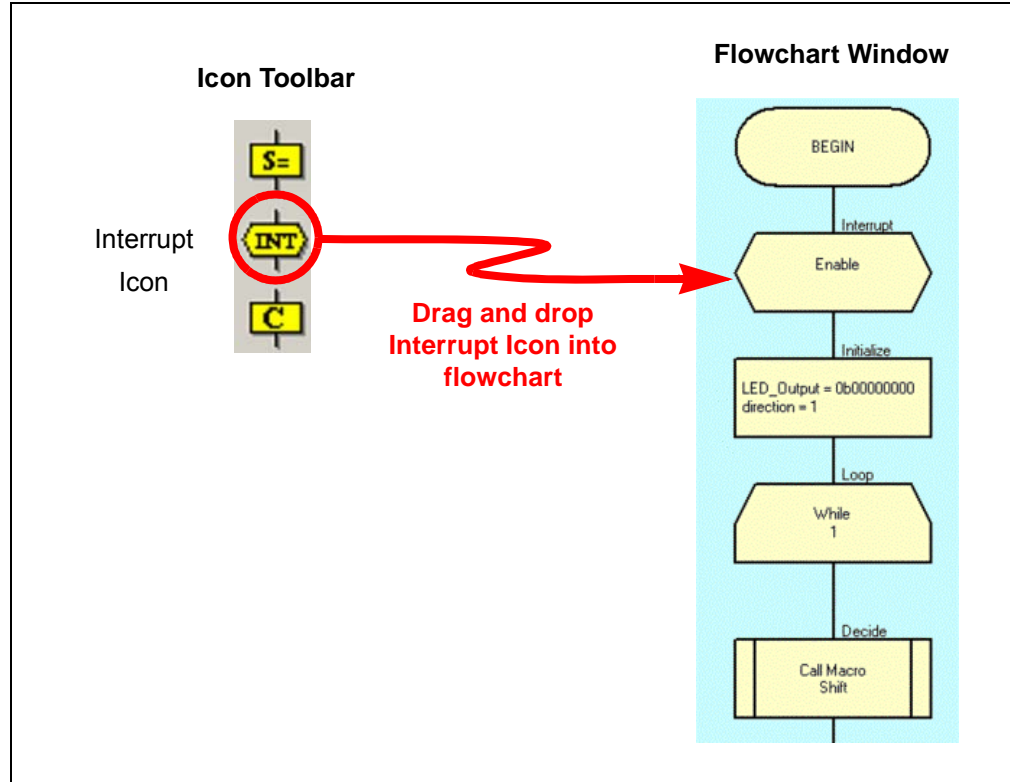
7. Next, immediately beneath the `Decide` functional block, add the `Do_Outputs` functional block using the **Output** icon assigning the `LED_Output` variable to `PORTC` as was done in the previous lab.
8. Add the `Timing` functional block below the `Do_Outputs` functional block using the **Delay** icon and configure for 1 second.
9. The flowchart should now resemble Figure 2-55.

FIGURE 2-55: FLOWCHART UP TO THIS POINT



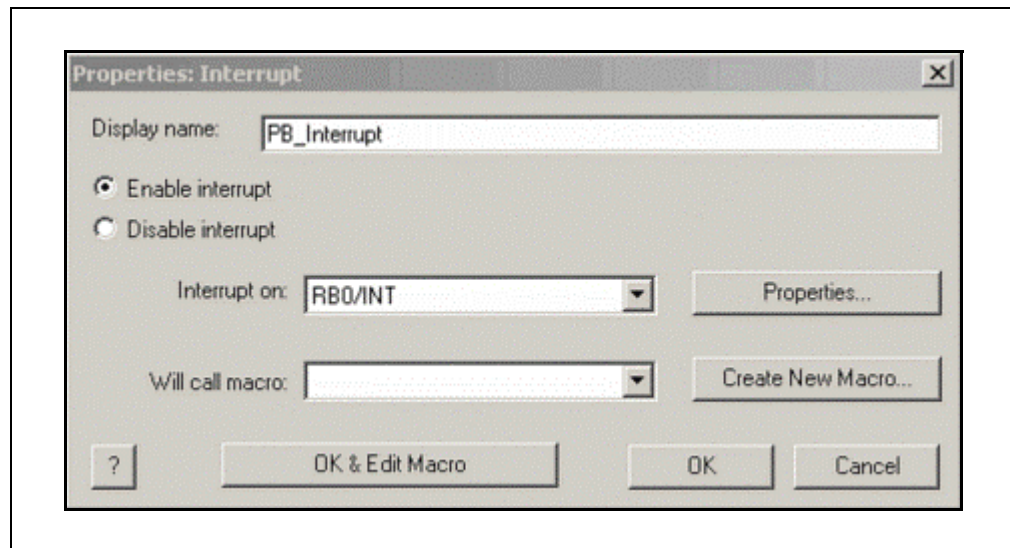
10. This application will again use the PORTA <RA2> I/O pin as the push button interface. Referring to steps 4 through 6 in the previous lab, connect the switch to RA2 with a 5mS debounce.
11. The `Get_Inputs` functional block will not be required in this lab as the push button will trigger an interrupt when pressed, thereby eliminating the need to poll the RA2 pin. To create the interrupt, the **Interrupt** icon will be used. Drag/drop an **Interrupt** icon to the very beginning of the main flowchart just beneath the BEGIN point (see Figure 2-56).

FIGURE 2-56: ADDING THE INTERRUPT ICON



12. Double-click on the **Interrupt** icon to open the Properties: Interrupt window. Change the display name to `PB_Interrupt` and select **RB0/INT** from the “Interrupt on:” drop-down menu (see Figure 2-57).

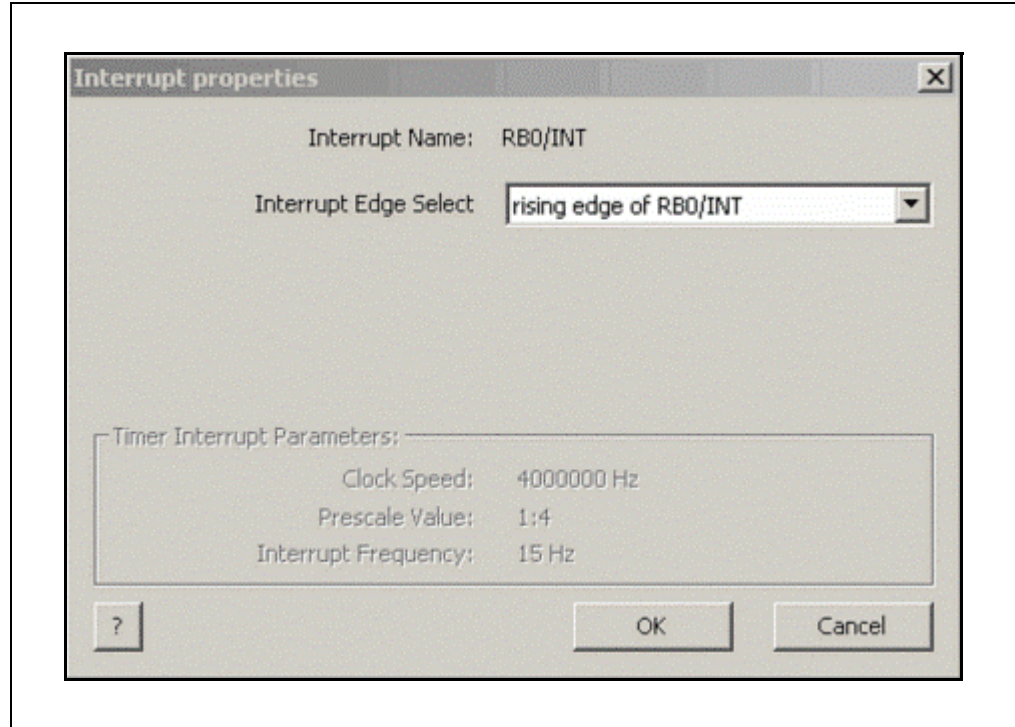
FIGURE 2-57: CONFIGURING AN INTERRUPT FOR INT



13. Click on the **Properties...** button next to the “Interrupt on:” drop-down menu. In the Interrupt Properties window, select “**rising edge of RB0/INT**” in the “Interrupt Edge Select” drop-down menu (see Figure 2-58).



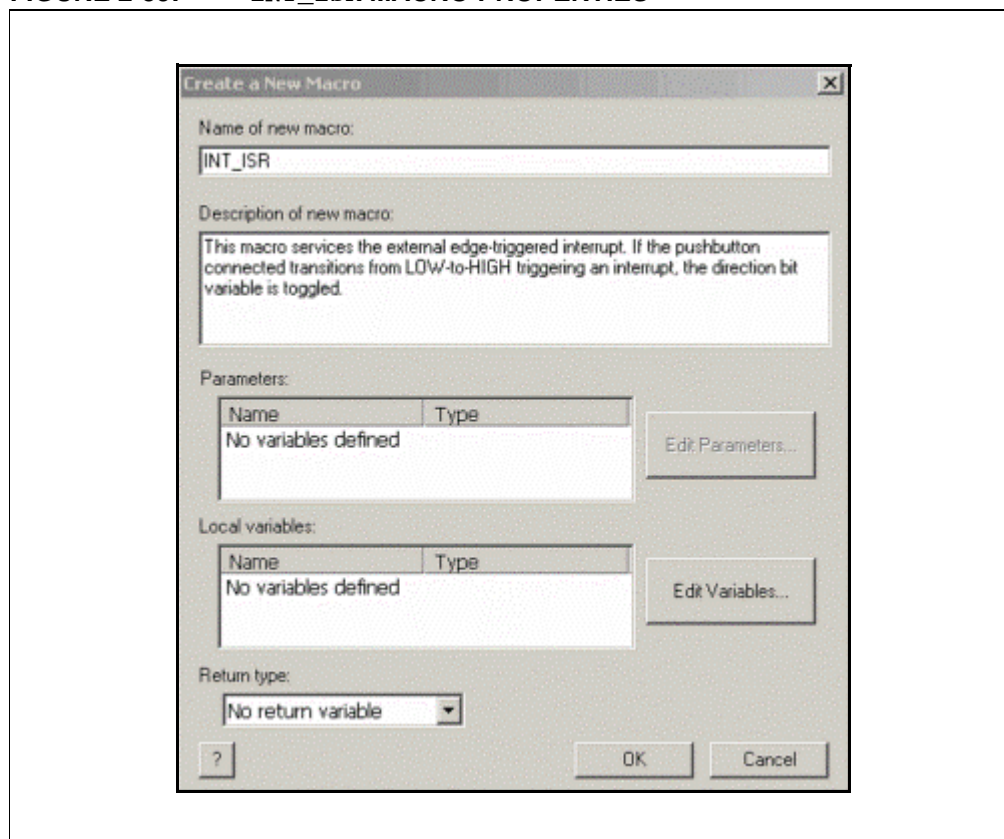
FIGURE 2-58: CONFIGURING THE INT INTERRUPT PROPERTIES



Click **OK** in the Interrupt Properties window to return to the Properties: Interrupt window.

- Next, a macro will be created that will be called whenever the **INT** interrupt is triggered. This will be the Interrupt Service Routine or ISR. Click the **Create New Macro...** button. The Create a New Macro window will now open. Rename the macro as `INT_ISR` and provide a description in the Description of New Macro: window similar to that shown in Figure 2-59.

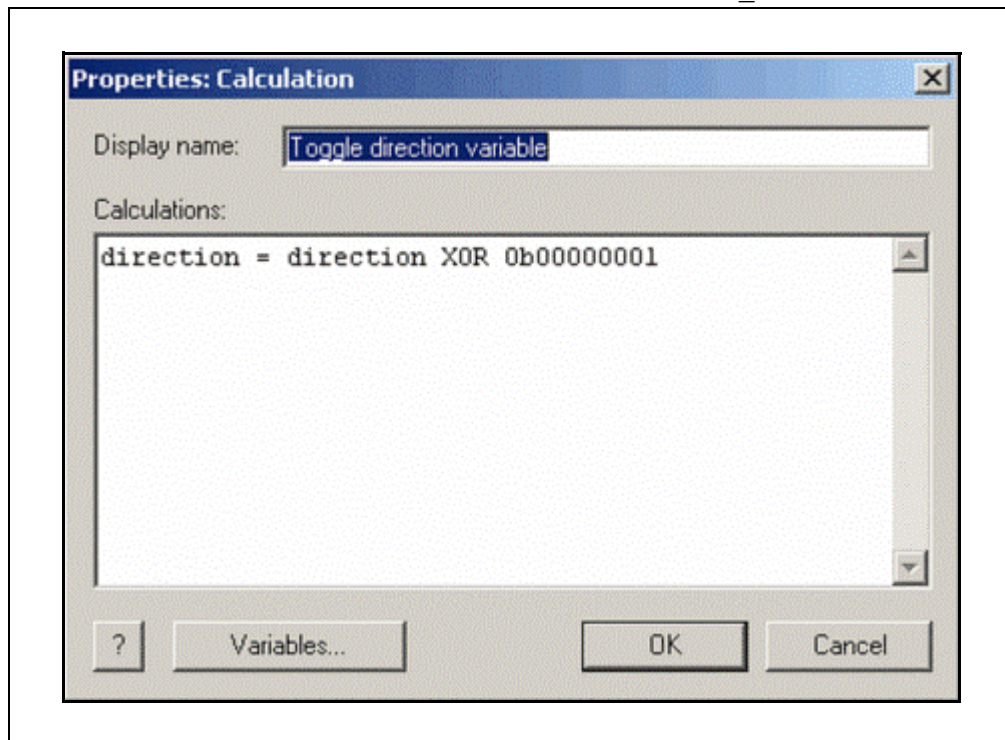
FIGURE 2-59: INT\_ISR MACRO PROPERTIES



Click **OK** to return to the Properties: Interrupt window.

15. In the Properties: Interrupt window, click on the **OK & Edit Macro** button to open a new flowchart workspace for the `INT_ISR` macro.
16. In the new `INT_ISR` workspace, add a **Calculation** icon between the BEGIN and END points. Double-click on the **Calculation** icon to open the Properties window. Change the "Display name:" to "Toggle direction variable". In the Calculations: window add the text that will toggle the `direction` variable from 0-to-1 or 1-to-0 each time the interrupt is triggered using the XOR command as was done in previous labs (see Figure 2-60).

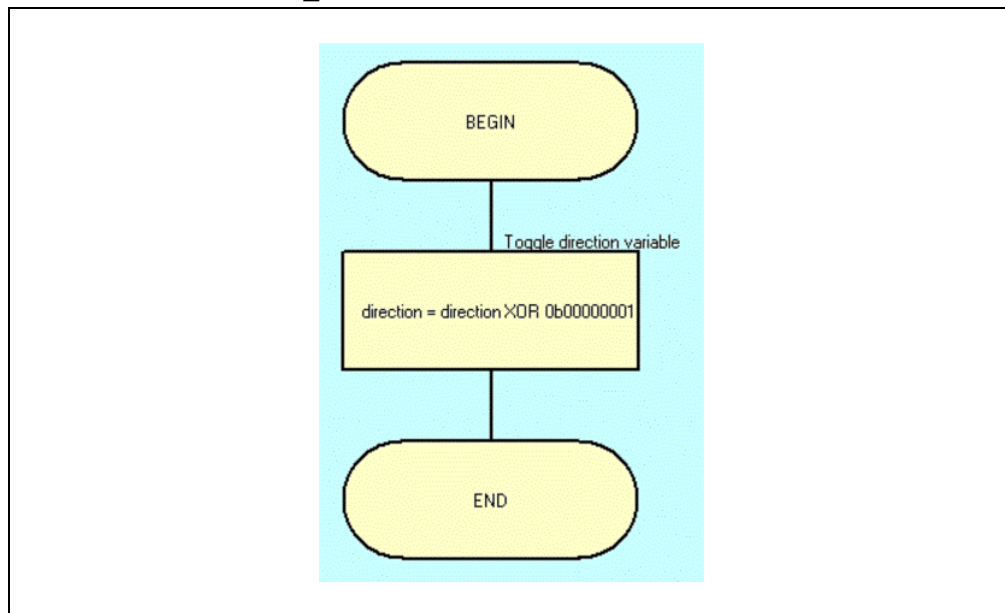
FIGURE 2-60: CALCULATION PROPERTIES FOR INT\_ISR



Click **OK** to continue.

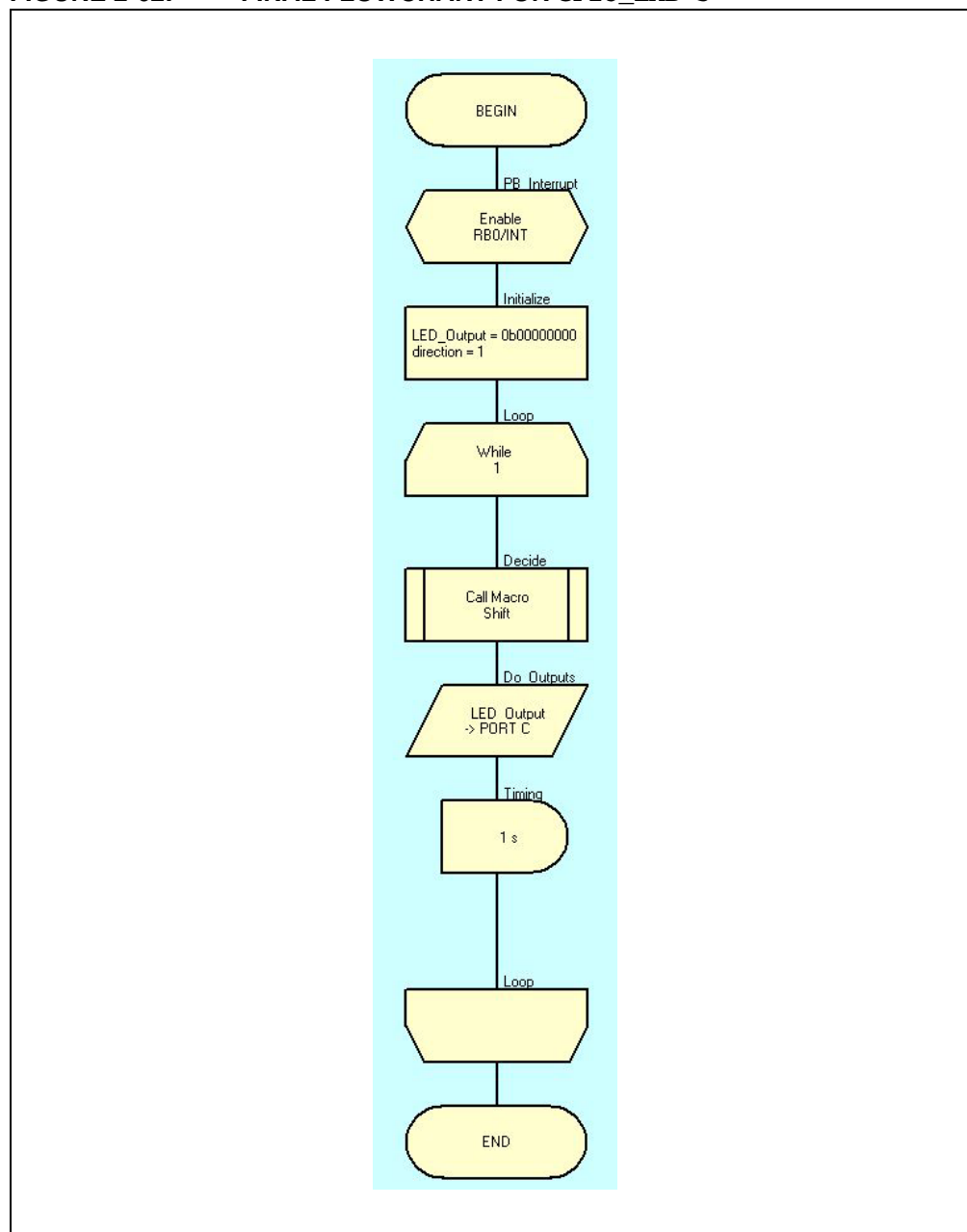
17. The INT\_ISR flowchart should now resemble Figure 2-61.

FIGURE 2-61: INT\_ISR FLOWCHART



18. Return to the main Flowchart window. The main flowchart should now resemble Figure 2-62.

FIGURE 2-62: FINAL FLOWCHART FOR GPIO\_LAB 5



19. Compile the project to chip. There should be no errors.

#### 2.4.5.4 TESTING THE APPLICATION

The LEDs connected to the individual PORTC pins should flash ON/OFF sequentially from right-to-left in 1-second intervals while the push button is released. Pressing the push button will change the ON/OFF flashing to left-to-right. This should behave the same as the previous lab. The user may notice that the LEDs are now more responsive to the push button press thanks to the addition of an interrupt.

---

---

## Chapter 3. Comparator Peripheral

---

---

### 3.1 INTRODUCTION

The following labs cover some of the fundamental features of the Comparator 1 peripheral found on the PIC16F690, including some unique applications. The PIC16F690 has two comparators, Comparator 1 and Comparator 2. These peripherals are very useful mixed signal building blocks, as they provide analog functionality independent of program execution. This means that, once initialized, no further firmware is required for a functioning application. The labs in this section will demonstrate this functionality, then introduce intelligence in the form of additional firmware to implement a Higher Resolution Sensor Measurement application.

### 3.2 COMPARATOR LABS

#### 3.2.1 Reference Documentation

All Documentation is available for download, free of charge, from [www.microchip.com](http://www.microchip.com):

- DS41262 – PIC16F690 data sheet:
  - Section 4.0: I/O Ports
  - Section 8.0: Comparator Module

#### 3.2.2 Comparator Labs

The labs that will be implemented in this chapter are:

- Lab 1: Simple Comparator
- Lab 2: Using the Internal Comparator Voltage Reference
- Lab 3: Higher Resolution Sensor Readings Using a Single Comparator

#### 3.2.3 Equipment Required

To complete the labs in this section, the following components are required:

1. 1 – 1 K $\Omega$  resistor
2. 2 – 10 K $\Omega$  resistors
3. 4 – 470 $\Omega$  resistor
4. 1 – 100 K $\Omega$  potentiometer
5. 1 – 100 K $\Omega$  NTC Thermistor
6. 4 – Light Emitting Diodes
7. 1 – 1N4148 Diodes
8. 1 – 1  $\mu$ F Capacitor
9. PIC16F690 populating socket U2
10. Assorted jumper wires

## 3.2.4 Lab 1: Simple Compare

### 3.2.4.1 NEW REGISTERS USED IN THIS LAB

To configure the peripherals used in this lab, the following registers are used:

1. The Analog Select Register: ANSEL (Register 4-3 in Section 4 of the PIC16F690 data sheet).
  - As mentioned in the previous chapter, this register is used to configure pins shared with analog peripherals (such as the comparator) as either digital (clearing the individual bit to 0) or analog (setting the individual bit to '1').
2. PORTA Tri-State Register: TRISA (Register 4-2 in Section 4 of the PIC16F690 data sheet).
  - This register configures the individual pins associated with the PORTA register as either input (setting the associated bit to 1) or output (clearing the associated bit to '0').

**Note:** In the previous labs, these registers were configured automatically in the Flowcode environment using various icons. This lab will show the reader how to configure these registers using the **C Code** icon.

3. Comparator C1 Control Register 0: CM1CON0 (Register 8-1 in Section 8 of the PIC16F690 data sheet).
  - This register is the main control register for the Comparator 1 peripheral. This register allows the user to configure Comparator 1 to do the following:
    - Turn-on or enable the Comparator 1 peripheral.
    - Enable the Comparator 1 output so that it is available on the RA2/C1OUT pin or internal only.
    - Invert the output of Comparator 1.
    - Select either the RA0/C1IN+ pin or the internal Comparator Voltage Reference (more on this in the next lab) as the inverting reference input to Comparator 1.
    - Select one of 4 channels as the C1VIN- inverting reference input to Comparator 1.

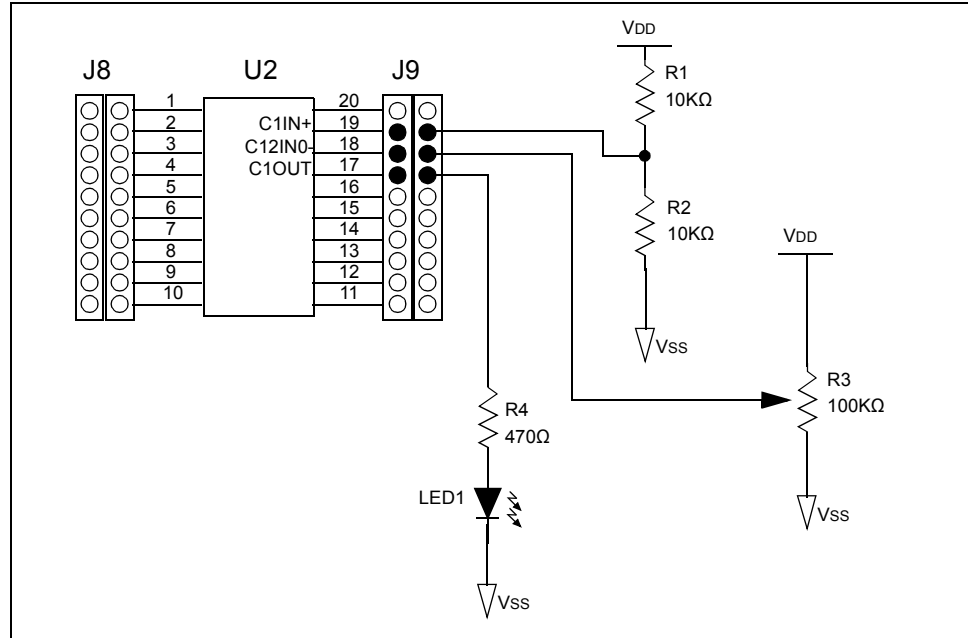
### 3.2.4.2 OVERVIEW

In this lab, Comparator 1 on the PIC16F690 is configured to perform a simple compare. A potentiometer connected to the inverting input (C12IN0-) of the comparator will be compared against the 2.5V connected to the non-inverting input (C1IN+) from a simple voltage divider circuit. A LED connected to the output of Comparator 1 (C1OUT) will light or turn off as follows:

- inverting reference > non-inverting reference = C1OUT is LOW = LED OFF
- inverting reference < non-inverting reference = C1OUT is HIGH = LED ON

The PICDEM Development Board configuration schematic is shown in Figure 3-1.

FIGURE 3-1: SCHEMATIC FOR COMPARATOR LAB 1



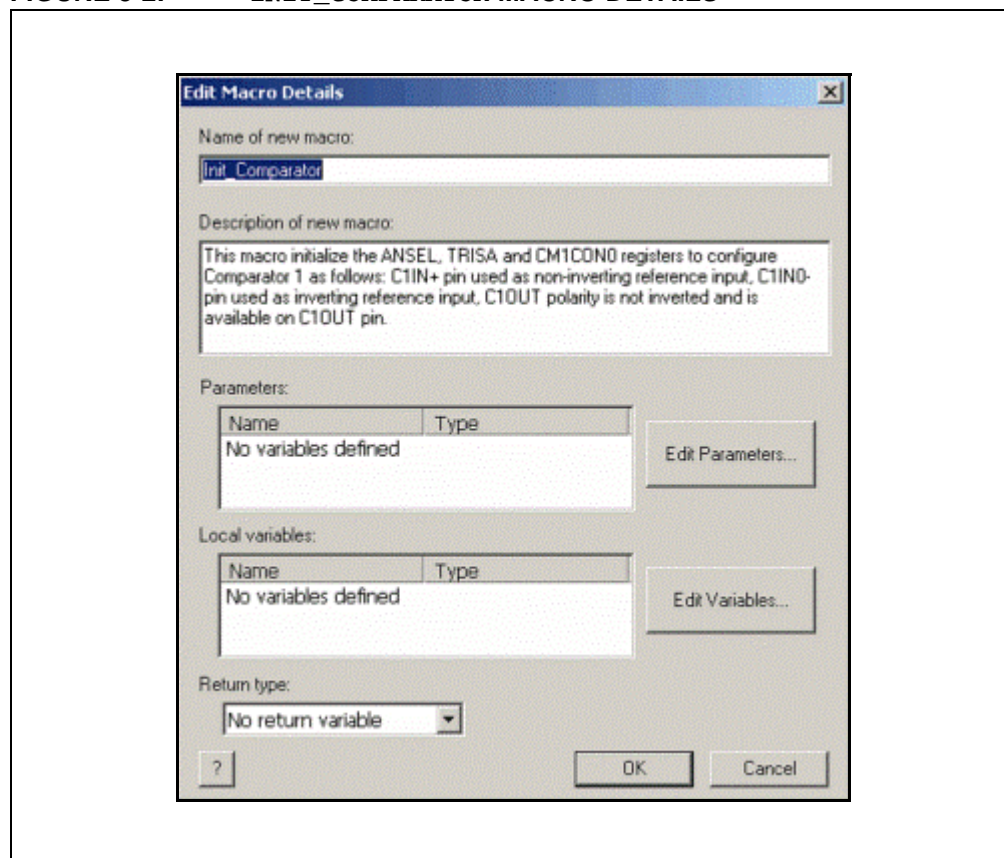
As mentioned in the introduction, comparators on the PIC16F690 are able to function independent of software logic. Therefore, all that is needed is to create a simple macro for the `Initialize` functional block to configure the TRISA register, ANSEL register, Comparator 1 peripheral register (CM1CON0) and include the While loop used in all the previous labs. The `Initialize` functional block macro will use the **C Code** icon to initialize the appropriate registers. Including a block of C code into the flowchart allows the user to configure any register on the PIC16F690.

### 3.2.4.3 PROCEDURE

1. Create a new project in Flowcode using steps 1-9 from `GPIO_Lab1` in the preceding chapter, and save the project as `Comparator_Lab1.fcf` in the `C:\PICDEM_Lab\flowcode\Comparator_Labs\Comparator_Lab1` directory.
2. In the Flowcode environment add a **Loop** icon between the `BEGIN` and `END` points of the main flowchart.
3. Next, create a new macro by selecting *Macro>New...*. Name the new macro `Init_Comparator` and provide a description of the macro similar to that shown in Figure 3-2.



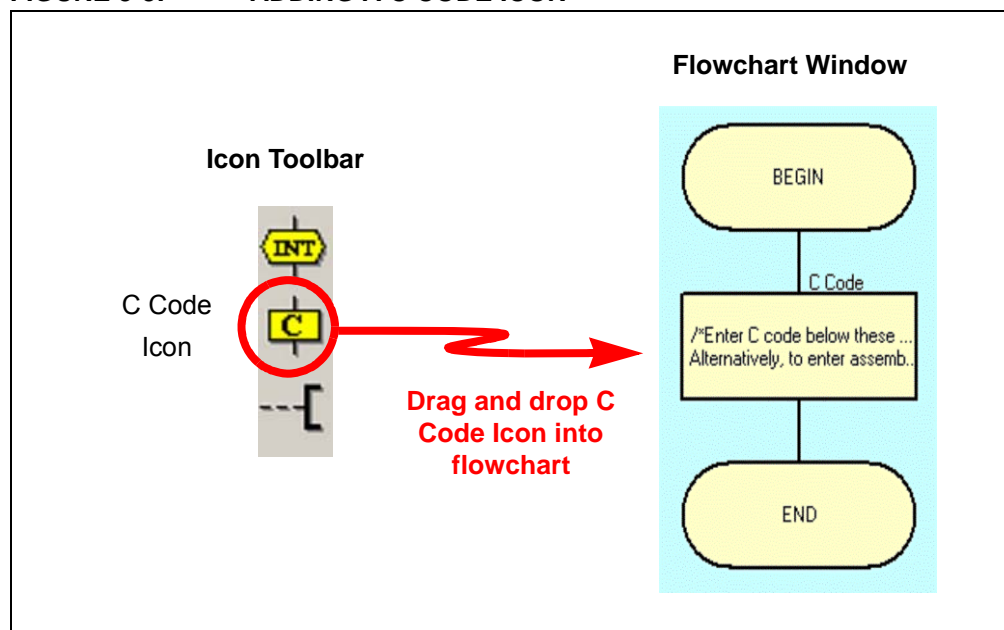
FIGURE 3-2: INIT\_COMPARATOR MACRO DETAILS



Click **OK** to continue.

4. In the `Init_Comparator` macro flowchart workspace, drag/drop a **C Code** icon between the BEGIN and END points (see Figure 3-3).

FIGURE 3-3: ADDING A C CODE ICON



# Comparator Peripheral

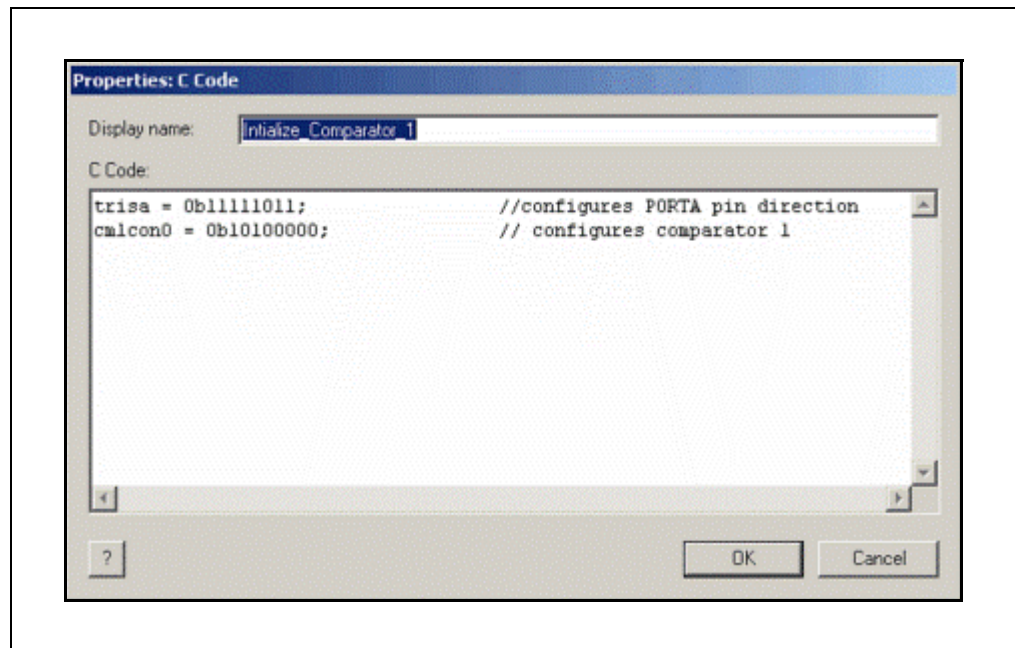
Double-click on the **C Code** icon to open its Properties window. Within the Properties: C Code window, remove any visible comments or code and add the following:

```
trisa = 0b11111011; //configures PORTA pin direction
cm1con0 = 0b10100000; //configures comparator 1
```

**Note:** The nomenclature to assign a value to a register on a particular device is to name the register (in lowercase), then assign a value to the register using the “=” operator. The binary radix is used here for ease of reference with the register bit designations in the PIC16F690 data sheet. The reader is encouraged to compare these binary values to the individual bits of each register as referenced at the beginning of this lab. Comments are not used by the PIC16F690 and are preceded in code with “//” operator.

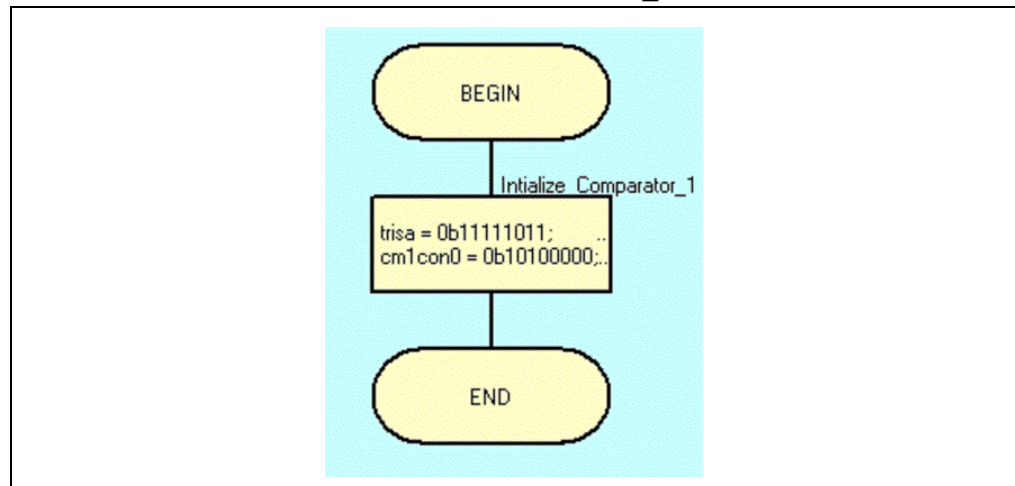
The Properties: C Code window should now resemble Figure 3-4.

**FIGURE 3-4: PROPERTIES: C CODE WINDOW WITH COMPARATOR CONFIGURATION CODE ADDED**



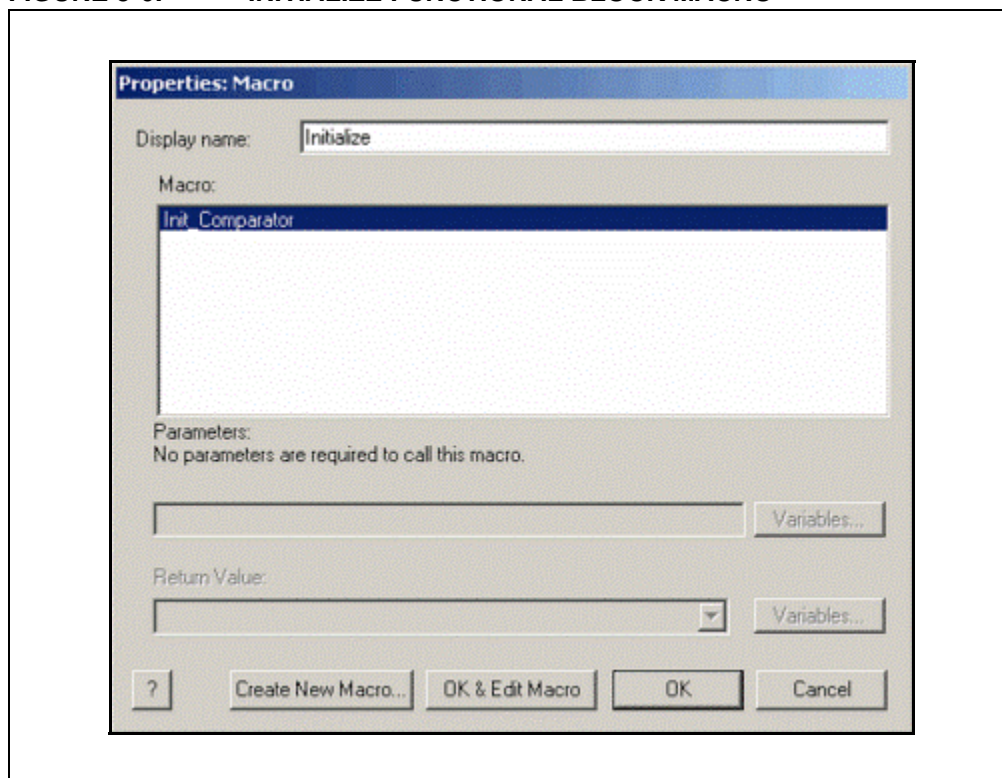
Click **OK** to continue. The flowchart should now resemble Figure 3-5.

**FIGURE 3-5: FINAL FLOWCHART FOR INIT\_COMPARATOR MACRO**



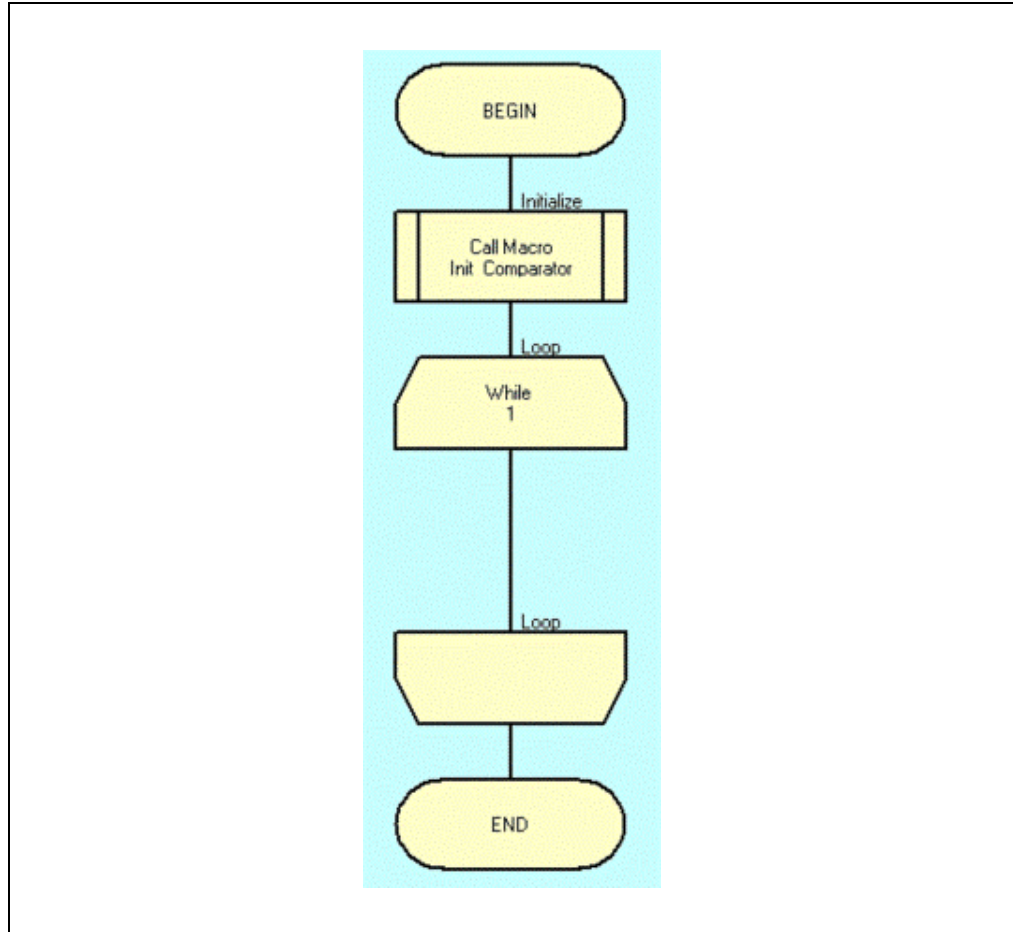
5. Return to the main Flowchart window. Drag/drop a **Macro** icon into the flowchart beneath the BEGIN point, but before the **Loop** icon.
6. Double-click on the **Macro** icon to open its properties. Change the “Display name:” to `Initialize`, highlight the `Init_Comparator` macro created in the previous steps and click **OK** (see Figure 3-6).

**FIGURE 3-6: INITIALIZE FUNCTIONAL BLOCK MACRO**



The final flowchart for this lab should now resemble Figure 3-7.

FIGURE 3-7: FINAL FLOWCHART FOR COMPARATOR\_LAB1



**Note:** Before compiling this project to the PIC16F690, be sure to disconnect the two jumper wires connecting the voltage divider and potentiometer circuits to pins RA0/C1IN+/ICSPDAT and RA1/C12IN0-/ICSPCLK. These pins are used during the programming process as the data (ICSPDAT) and clock (ICSPCLK) sources. Any components connected to these pins could interfere with the programming process and result in the PIC16F690 not being programmed. More information can be found in Section 14.9 “In-Circuit Serial Programming” in the PIC16F690 data sheet.

7. With the two pins disconnected as per the preceding note, compile the project to the PIC16F690. There should be no errors.

#### 3.2.4.4 TESTING THE APPLICATION

The LED connected to the C1OUT pin should light when the voltage present on the C12IN0- pin is less than the 2.5V present on the C1IN+ pin and turn off when the 2.5V is exceeded.

The solution for this project can be found in the

C:\PICDEM\_Lab\Flowcode\Comparator\_Labs\Comparator\_Lab1\solution directory.

## 3.2.5 Lab 2: Using the Comparator Voltage Reference

### 3.2.5.1 NEW REGISTERS USED IN THIS LAB

To configure the peripherals used in this lab, the following registers are used:

1. Voltage Reference Control Register: VRCON (Register 8-5 in Section 8 of the PIC16F690 data sheet)
  - Enables either the Comparator Voltage Reference or the 0.6V constant reference as the non-inverting reference input to Comparator C1 or Comparator C2.
  - Selects either a High or Low resolution 16-level voltage range
  - Enables a 0.6V reference
  - Uses three bits to configure the reference voltage level

### 3.2.5.2 OVERVIEW

This lab expands on Lab 1 by utilizing the internal Comparator Voltage Reference (CVREF) feature on the PIC16F690. The CVREF provides an internally generated voltage reference that can be used by the Comparator 1's non-inverting reference input, so that external components are not needed such as the resistor voltage divider used in the previous lab. The CVREF features:

- Independent comparator operation
- Two 16-level voltage ranges
- Ratiometric with VDD
- Fixed 0.6V reference option
- Output clamped to Vss

The CVREF has 2 ranges with 16 voltage levels in each range. Range selection is controlled by the CVREF Range Selection (VRR) bit in the VRCON (Voltage Reference Control Register) along with the CVREF Value Selection bits (VR<3:0>). The Value Selection bits hold a value based upon some simple calculations to set the internal reference voltage. The CVREF voltage is determined using Equation 3-1:

#### EQUATION 3-1: CVREF Output Voltage

VRR = 1 (Low-Range):

$$CVREF = (VR<3:0>/24) \times VDD$$

VRR = 0 (Low-Range):

$$CVREF = (VDD/4) + (VR<3:0> \times VDD/32)$$

This lab will implement the low-range calculation by setting the VRR bit in VRCON equal to 1. Equation 3-2 demonstrates how to calculate the VR<3:0> values, using the low-range method, to obtain a 2.5V internal reference. If higher resolutions are required, the High Range method should be used (see Section 8.10.2 in the PIC16F690 data sheet).

## EQUATION 3-2: Calculating a 2.5V Internal Reference (Low-Range Method)

$VRR = 1$  (Low-Range):

$$CVREF = (VR<3:0>/24) \times VDD$$

Known: desired  $CVREF = 2.5V$ ,  $VDD$  approximately  $5V$

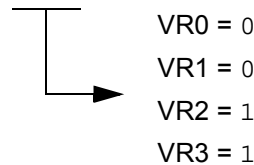
Therefore:

$$2.5V = (VR<3:0>/24) \times 5V$$

$$2.5V/5V = (VR<3:0>/24)$$

$$(2.5V/5V) \times 24 = VR<3:0>$$

$$VR<3:0> = 12_{10} \text{ or } 1100_2$$

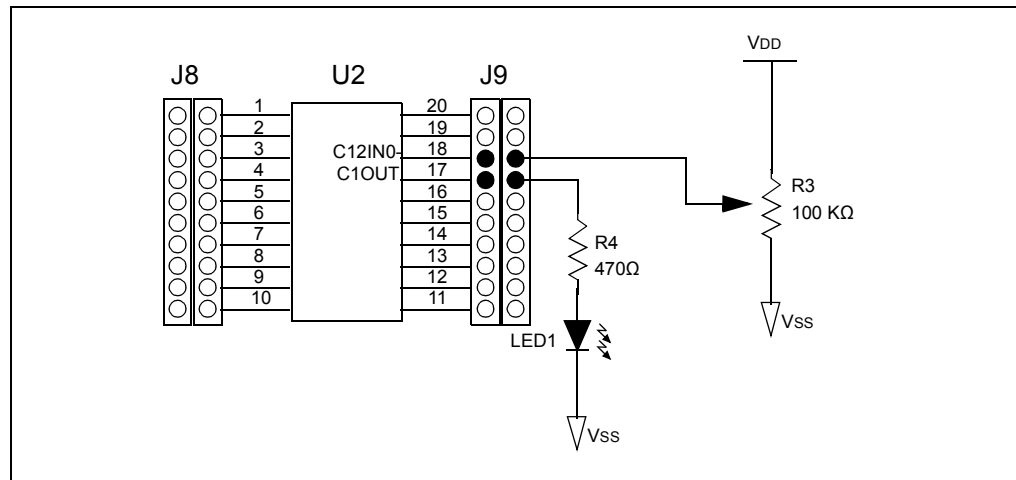


The `Initialize` functional block from the previous lab now must configure both the Comparator 1 peripheral and the  $CVREF$  as follows:

- Turn on Comparator 1
- Select  $CVREF$  the non-inverting reference for Comparator 1
- Continue to use the `C12IN0-` pin as the inverting reference
- Turn on  $CVREF$
- Select the low-range feature
- Set the  $CVREF$  Value Selection bits as per the calculation in Equation 3-2.

Changes to the PICDEM Development Board configuration schematic for this lab are shown in Figure 3-8.

**FIGURE 3-8: SCHEMATIC FOR COMPARATOR LAB 2**



### 3.2.5.3 PROCEDURE

1. Create a new project in Flowcode saving the project as `Comparator_Lab2.fcf` in the `C:\PICDEM_Lab\flowcode\Comparator_Labs\Comparator_Lab2` directory.



2. Build up the project flowchart exactly the same as was done in the previous lab (Figure 3-5 and Figure 3-7). There are some changes to the `Init_Comparator` macro that are needed in order to configure the CVREF peripheral. Therefore, the CM1CON0 register configuration will result in a different binary value by setting bit 2 (C1R: Comparator C1 Reference Select bit) to 1 so that the CVREF output connects to the non-inverting input of Comparator 1 changing the C code to:

```
cm1con0 = 0b10100100; // configures comparator 1 with CVREF
                    //as non-inverting input to
                    //Comparator 1
```

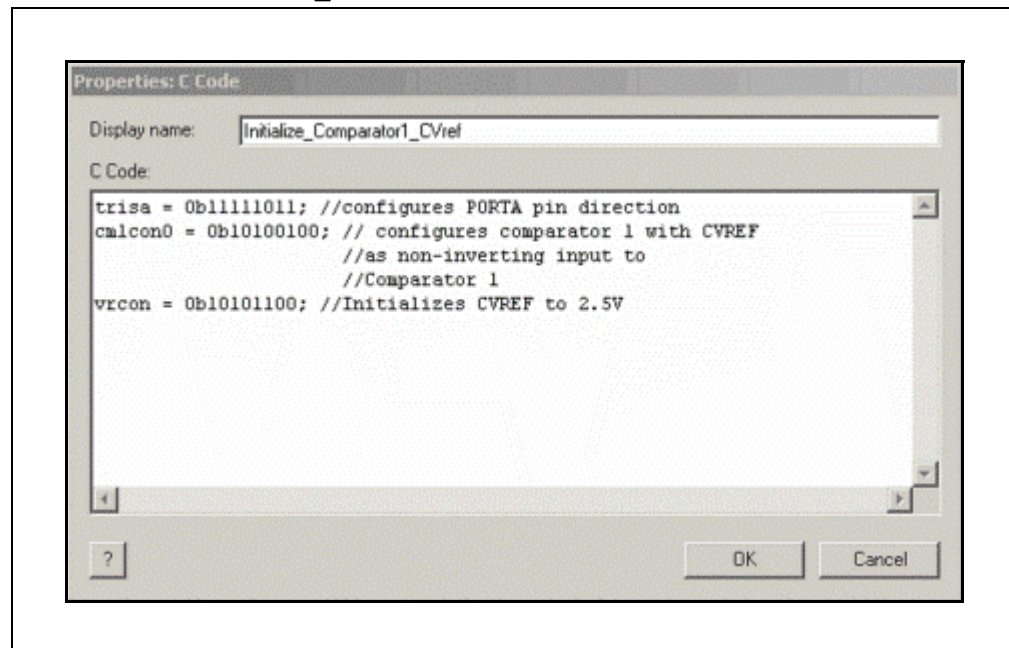
3. A new line of code will be added immediately following the CM1CON0 initialization to configure the Voltage Reference Control Register (VRCON) whose output is now connected to the non-inverting reference input of Comparator 1. Add the following code:

```
vrcon = 0b10101100; //Initializes CVREF to 2.5V
```

**Note:** The last four bits (i.e., 0bxxxx1100) are the Comparator Voltage Reference Value Selection bits that set CVREF to 2.5V.

The Properties: C Code window should now resemble Figure 3-9.

**FIGURE 3-9: UPDATED PROPERTIES: C CODE WINDOW FOR INIT\_COMPARATOR MACRO**



Click **OK** to continue.

4. Compile the project to chip, there should be no errors.

### 3.2.5.4 TESTING THE APPLICATION

The application should behave exactly as it did in Lab 1 with the exception of less components used.

The solution for this project can be found in the

C:\PICDEM\_Lab\Flowcode\Comparator\_Labs\Comparator\_Lab2\solution directory.



## 3.2.6 Lab 3 Higher Resolution Sensor Readings Using a Single Comparator

### 3.2.6.1 NEW REGISTERS USED IN THIS LAB

To configure the peripherals used in this lab, the following registers are used:

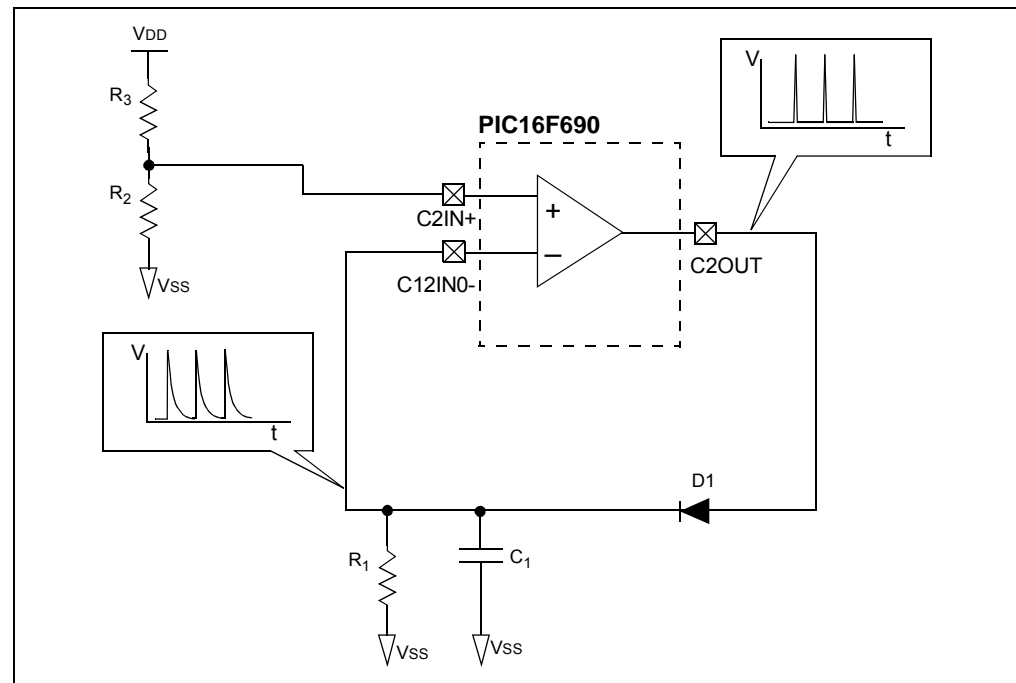
1. Comparator 2 Control Register 0: CM2CON0 (Register 8-2 in Section 8.0 of the PIC16F690 data sheet)
  - Same functionality as CM1CON0 outlined at the beginning of this chapter.
2. Timer0 Result Register: TMR0 (Section 5.0 of the PIC16F690 data sheet)
  - This 8-bit register increments on edge-transitions of the internal instruction clock ( $F_{osc}/4$ ) or an external clock source on the Timer0 Clock Input pin (TOCKI).

### 3.2.6.2 OVERVIEW

This lab expands on concepts discussed in the first comparator lab by implementing intelligence to create a higher resolution temperature sensor measurement application. Comparator 2 will be configured to operate as a simple relaxation oscillator with the addition of a few external components. The non-inverting reference will be connected to a variable voltage divider connected to the C2IN+ pin on the PIC16F690.

The basic oscillator circuit is shown in Figure 3-10.

**FIGURE 3-10: BASIC RELAXATION OSCILLATOR CIRCUIT**



Referring to Figure 3-10, at start-up, the capacitor connected to the inverting reference of Comparator 2 is completely discharged. Therefore, the voltage present on the inverting reference is 0V, which is less than the voltage on the non-inverting reference and Comparator 2's output goes high. This rapidly charges the capacitor through the diode (D1) to a level approximately equal to  $V_{DD}$ . Once the Comparator detects that the inverting reference input is greater than the non-inverting reference voltage, the output transitions low. The charge across the capacitor then discharges slowly across the resistor R1. Once the capacitor charge drops below the voltage on the non-inverting reference, the cycle repeats and the system oscillates. The frequency of this oscillation is dependant on the RC time constant ( $\tau = R \times C$ ), or the time it takes to

discharge the capacitor to 37% of its initial voltage. As either the resistance or capacitance decreases, so will  $\tau$  effectively increasing the frequency of the oscillator. If the resistor is replaced with a Negative Temperature Coefficient (NTC) thermistor where resistance decreases as temperature increases, any temperature change would cause a shift in resistance with a subsequent shift in the frequency of the oscillator.

This oscillator can be created quite easily by simply initializing the comparator and nothing more. However, with the addition of some intelligence and some additional peripherals, a higher resolution sensor measurement application can be achieved.

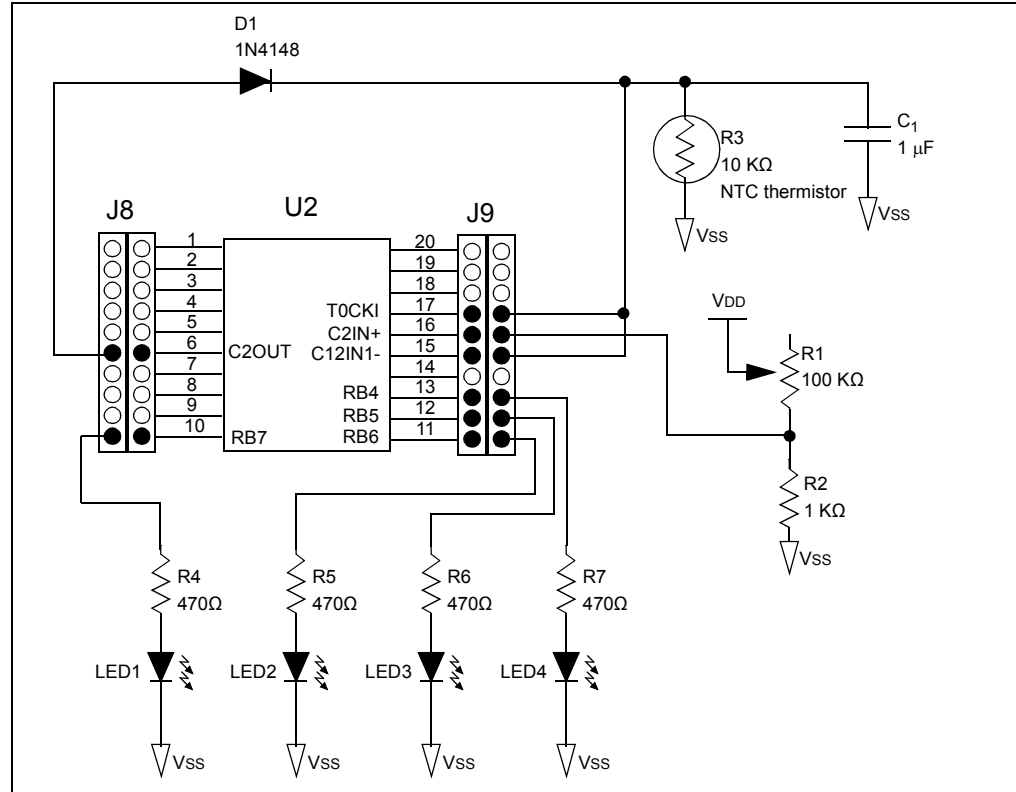
A common feature found on most PIC® microcontrollers is the 8-bit timer/counter peripheral Timer0. This timer can either use the internal instruction clock ( $F_{OSC}/4$ ) as its time base or an external clock source on the Timer0 Clock Input (T0CKI) pin to increment the value in an 8-bit result register TMR0. In this application, the oscillator described will be used as the Timer0 clock source. The Timer0 peripheral will be configured to increment the TMR0 register value with each low-to-high transition on the T0CKI pin effectively counting the number of pulses. This peripheral also features an interrupt on overflow. So, after 256 counts (0-255 inclusive), TMR0 will overflow triggering an interrupt. A counter variable within the main loop of the application will be used to count each time through the Software Control loop. On a Timer0 interrupt, the current value of the counter variable will be assigned to four LEDs connected to the PORTB RB4, RB5, RB6, RB7 pins. If heat is applied to the thermistor, the effective resistance will drop and the oscillator frequency will increase. Since the oscillator is the clock source for the Timer0 peripheral, TMR0 will overflow sooner and the counter variable value would be less. Colder temperatures will increase the effective resistance, slow the oscillator frequency and subsequently the overflow interrupt period and a higher counter value will be obtained. In other words:

**increase temp. = increased oscillator frequency = decrease counter value**

**decrease temp. = decreased oscillator frequency = increased counter value**

The schematic for this lab is shown in Figure 3-11.

FIGURE 3-11: SCHEMATIC FOR COMPARATOR LAB 3



The `Initialize` functional block from Comparator Lab 1 will change somewhat to configure Comparator 2 associated pins and the Timer0 clock source pin (T0CKI) as follows:

- Comparator 2
  - Enable Comparator 2
  - Make the Comparator 2 output available on the C2OUT pin configuring TRISC4 as an output.
  - Configure the RC0/C2IN+ pin as the non-inverting reference input
  - Select pin C12IN1- as the inverting reference (port pins shared with analog peripherals default to analog input on start-up. Therefore, the non-inverting or inverting input pins to Comparator 2 need not be initialized saving lines of C code for other commands).
- Timer0 Configuration:
  - Configure RA2/AN2/T0CKI pin as a digital input to accommodate the oscillator output to be used as the TMR0 clock source.

Using the **Interrupt** icon introduced in the General Purpose Input/Output Lab 5 from the previous chapter, the Timer0 interrupt on overflow is configured as follows:

- Use the T0CKI pin as the Timer0 clock source
- Using the prescaler feature, increment the TMR0 register on every 4th low-to-high transition on the T0CKI pin.
- Prescaler rates for the Timer0 module available are:
  - 1 TMR0 increment: 1 clock source pulse
  - 1 TMR0 increment: 2 clock source pulse
  - 1 TMR0 increment: 4 clock source pulse
  - 1 TMR0 increment: 8 clock source pulse
  - 1 TMR0 increment: 64 clock source pulse

- 1 TMR0 increment:128 clock source pulse
- 1 TMR0 increment: 256 clock source pulse

### 3.2.6.3 PROCEDURE

1. Create a new project in Flowcode using steps 1-9 from GPIO\_Lab1 in the preceding chapter saving the project as `Comparator_Lab3.fcf` in the `C:\PICDEM_Lab\flowcode\Comparator_Labs\Comparator_Lab3` directory.
2. Create a new macro for the `Initialize` functional block. Rename the macro `Init_Peripherals` and provide a description. Within the macro flowchart, drag/drop a **C Code** icon that will be used to initialize the Comparator 2 and Timer0 peripherals and pins. Double-click on the **C Code** icon to open the Properties: C Code window and rename to `Init_C2_TMR0` adding the following lines of C code to the "C Code:" section:

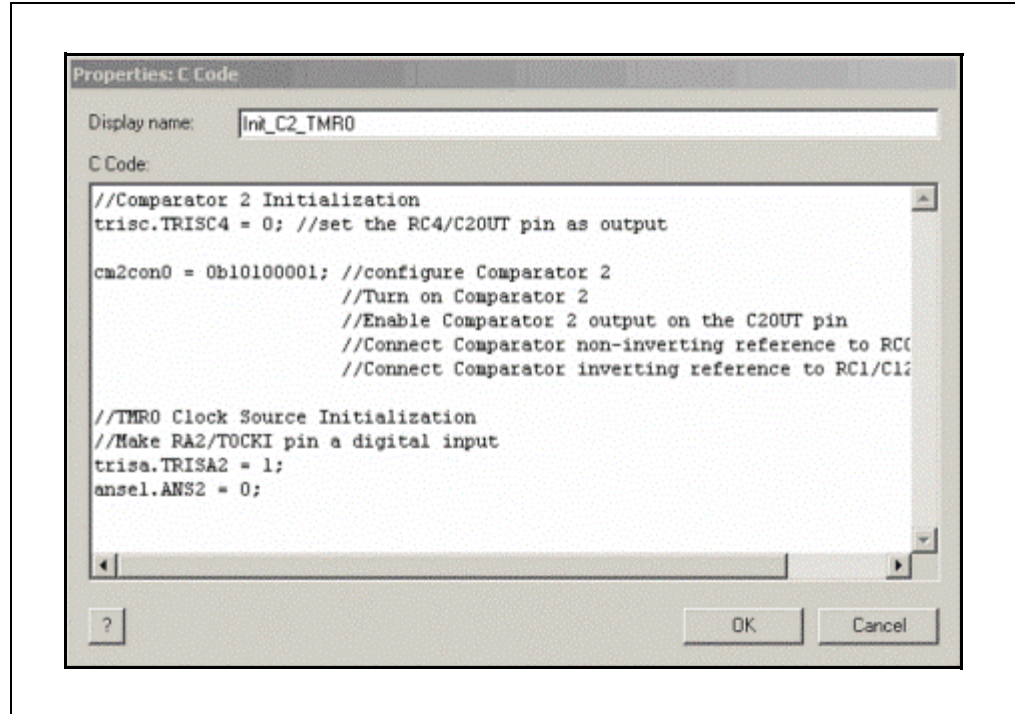
**FIGURE 3-12: ADDING LINES OF C CODE**

```
//Comparator 2 Initialization
trisc.TRISC4 = 0; //set the RC4/C2OUT pin as output
cm2con0 = 0b10100001; //configure Comparator 2
                    //Turn on Comparator 2
                    //Enable Comparator 2 output on the
                    //C2OUT pin
                    //Connect Comparator non-inverting
                    //reference to RC0/C2IN+ pin
                    //Connect Comparator inverting reference
                    //to RC1/C12IN1- pin

//TMR0 Clock Source Initialization
//Make RA2/T0CKI pin a digital input
trisa.TRISA2 = 1;
ansel.ANS2 = 0;
```

The Properties: C Code window should now resemble Figure 3-13.

FIGURE 3-13: PROPERTIES: C CODE WINDOW INIT\_C2\_TMR0

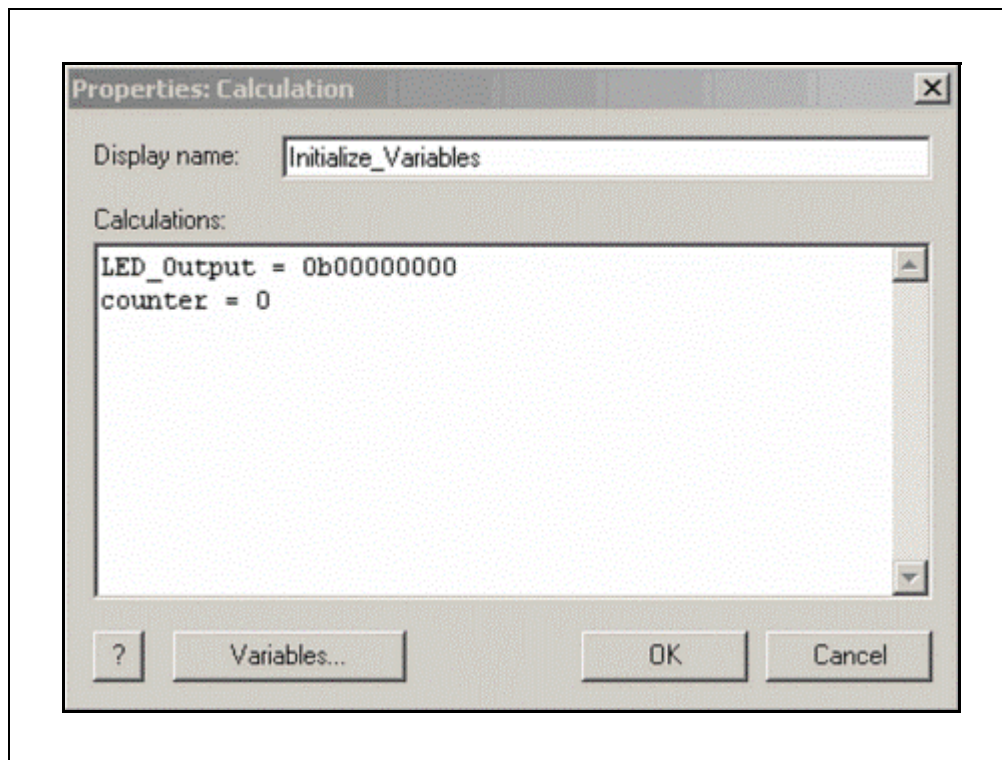


Click **OK**.

**Note:** The nomenclature to set individual bits in C requires the source register be identified in lowercase followed by a decimal then the specific bit in uppercase.

3. Drag/drop a **Calculation** icon to the `Init_Peripherals` immediately following the **C Code** icon from the previous step and rename `Initialize_Variables`. Click the **Variables...** button and create two 8-bit variables `LED_Output` and `counter`. Using the **Use Variable...** button, add each variable to the “Calculation” properties, click **Close** to return to the Properties: Calculation window and initialize both variables to ‘0’. The Properties: Calculation window should now resemble Figure 3-14.

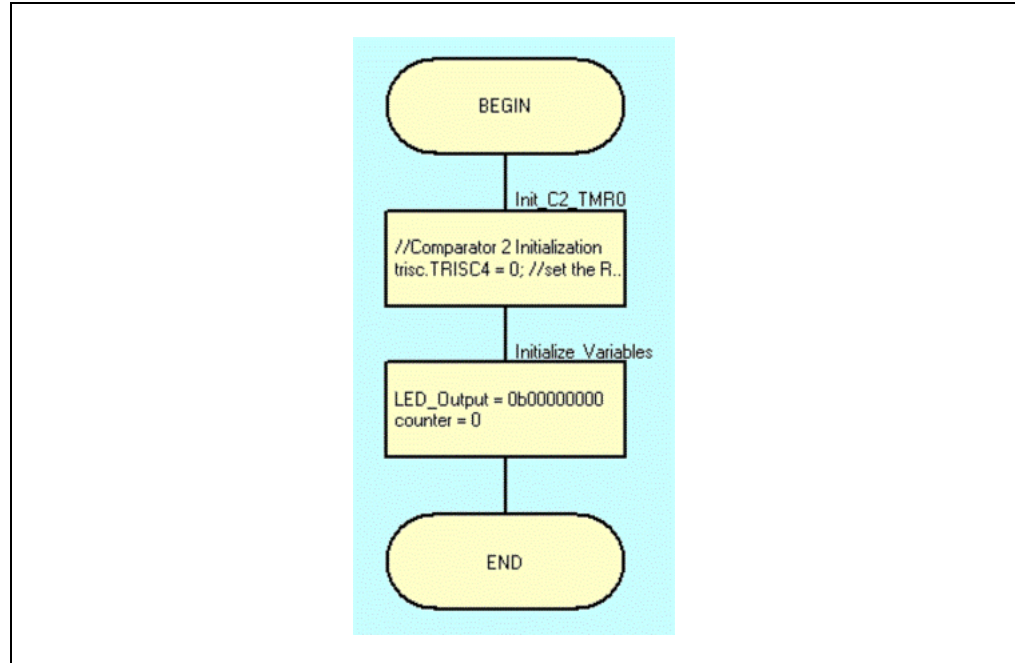
**FIGURE 3-14:      PROPERTIES: CALCULATION WINDOW FOR INITIALIZE\_VARIABLES**



Click **OK** to continue.

The `Init_Peripherals` macro flowchart should now resemble Figure 3-15.

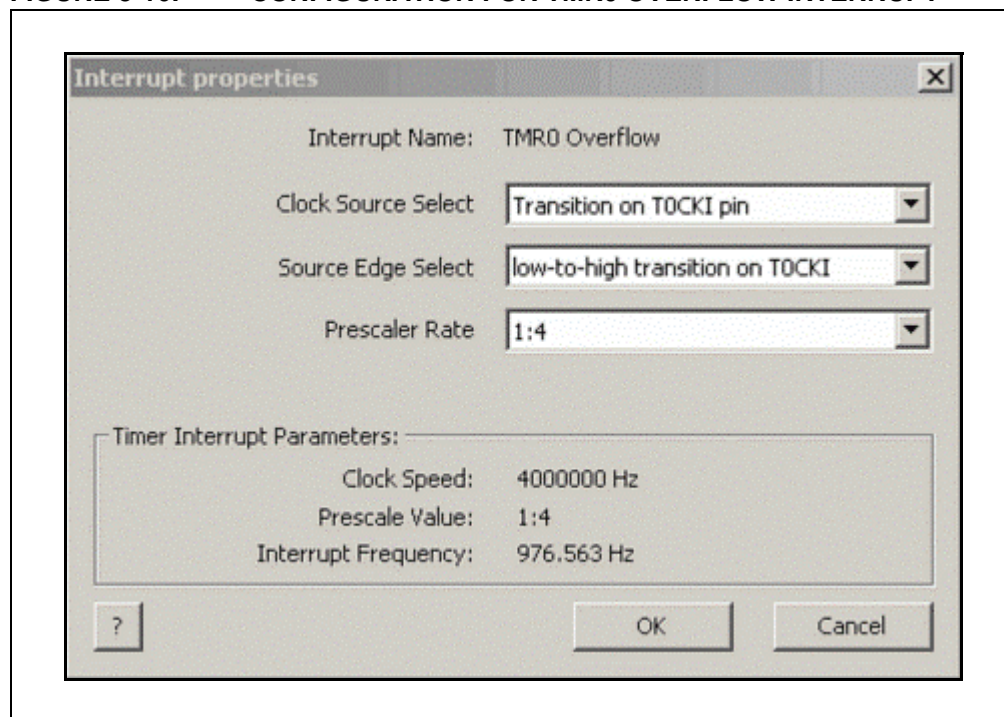
FIGURE 3-15: INIT\_PERIPHERALS FLOWCHART FOR COMPARATOR\_LAB3



4. Return to the main Flowchart window, drag/drop a **Macro** icon, rename it `Initialize` and call the `Init_Peripherals` macro created in the previous steps.
5. Next, the Timer0 interrupt will be created. Drag/drop an **Interrupt** icon into the main flowchart immediately following the `Initialize` functional block created in the preceding step. Double-click the **Interrupt** icon to open the Properties: Interrupt window. Select **TMR0 Overflow** from the "Interrupt on:" drop-down menu. Click on the **Properties...** button and using the associated drop-down menus, configure the Timer0 interrupt, as shown in Figure 3-16.



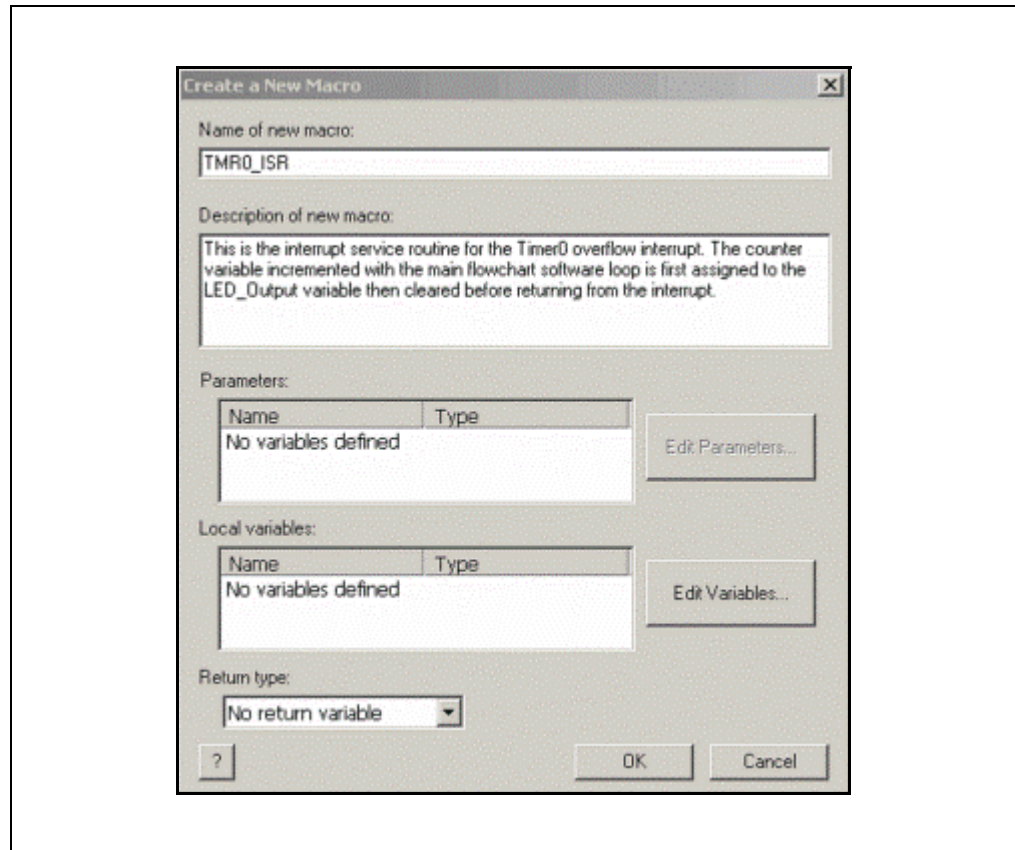
FIGURE 3-16: CONFIGURATION FOR TMR0 OVERFLOW INTERRUPT



Click **OK** to return to the Properties: Interrupt window.

6. In the Properties: Interrupt window click the **Create New Macro...** button to open the Create a New Macro window. Rename the macro `TMR0_ISR` and provide a description of the macro similar to that shown in Figure 3-17.

**FIGURE 3-17: CREATE A NEW MACRO WINDOW FOR TMR0 INTERRUPT SERVICE ROUTINE**



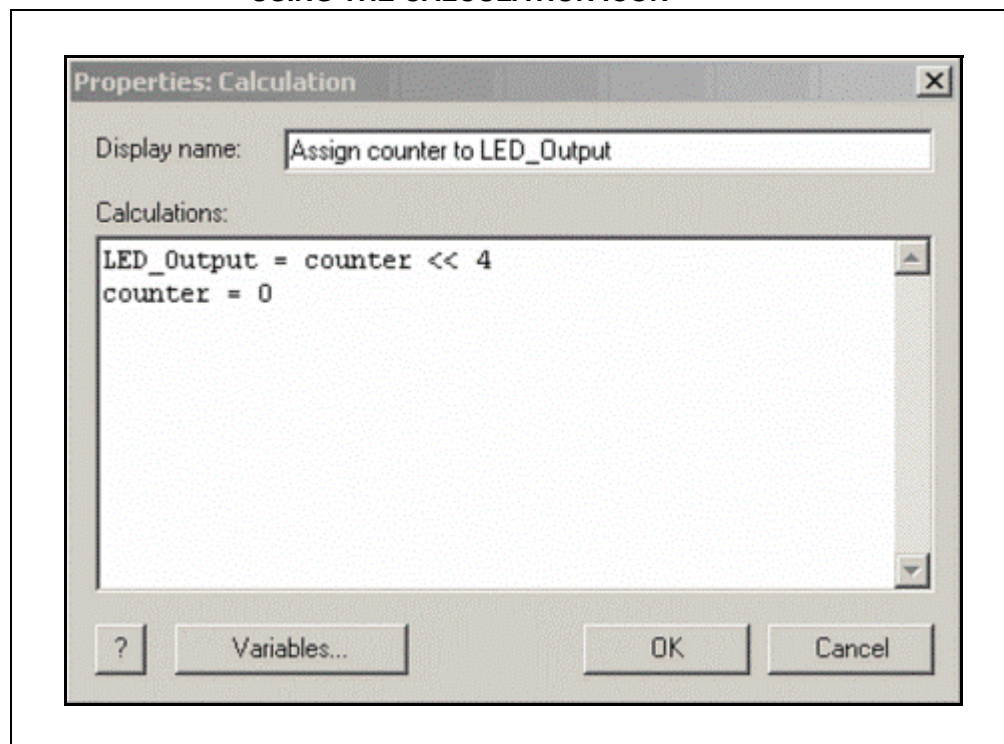
Click **OK** to continue.

7. In the Properties: Interrupt window, be sure that the TMR0\_ISR macro is selected in the "Will call macro:" drop-down menu and click the **OK & Edit Macro** button to open the TMR0\_ISR macro flowchart window.
8. In the TMR0\_ISR flowchart window, drag/drop a **Calculation** icon between the BEGIN and END points. Double-click on the icon to open the Properties: Calculation window. Rename the icon to something descriptive, like "Assign counter to LED\_Output", and add the following code to the "Calculations" section:

```
LED_Output = counter << 4  
counter = 0
```

The current value in the `counter` variable is shifted four bits to the left before being assigned to the `LED_Output` variable. The main flowchart will implement a 1mS delay in the Timing functional block that will ensure the `counter` variable never exceeds a 4-bit value. Once comfortable with the contents of this lab, the user is encouraged to manipulate various parameters of the application firmware such as the delay or using an 8-bit result to output to the LEDs connected to the PIC16F690 pins. The Properties: Calculation window should now resemble Figure 3-18.

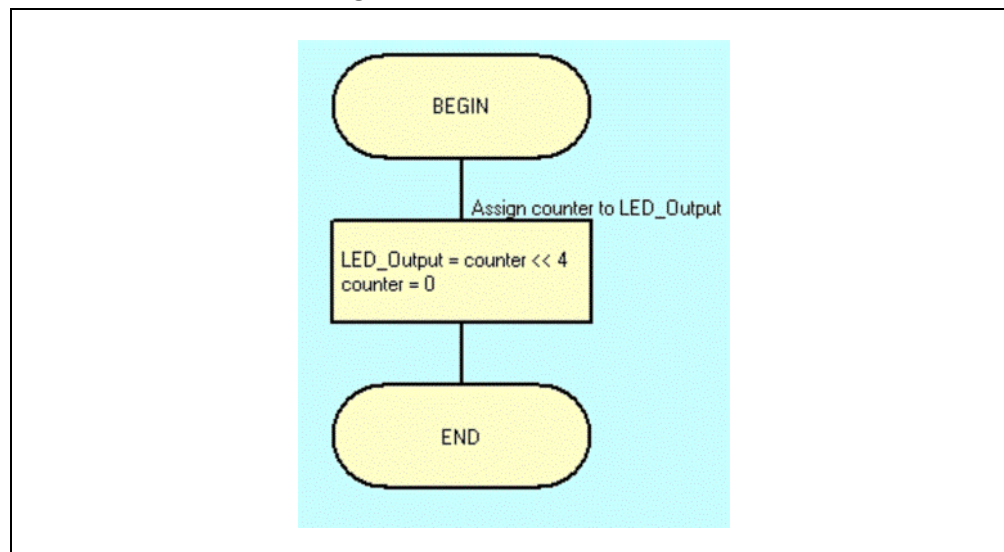
**FIGURE 3-18: ASSIGNING THE COUNTER VARIABLE TO LED\_OUTPUT USING THE CALCULATION ICON**



Click **OK** to continue.

The TMR0\_ISR flowchart should now resemble Figure 3-19.

**FIGURE 3-19: TMR0\_ISR FLOWCHART FOR TIMER0 OVERFLOW INTERRUPT**



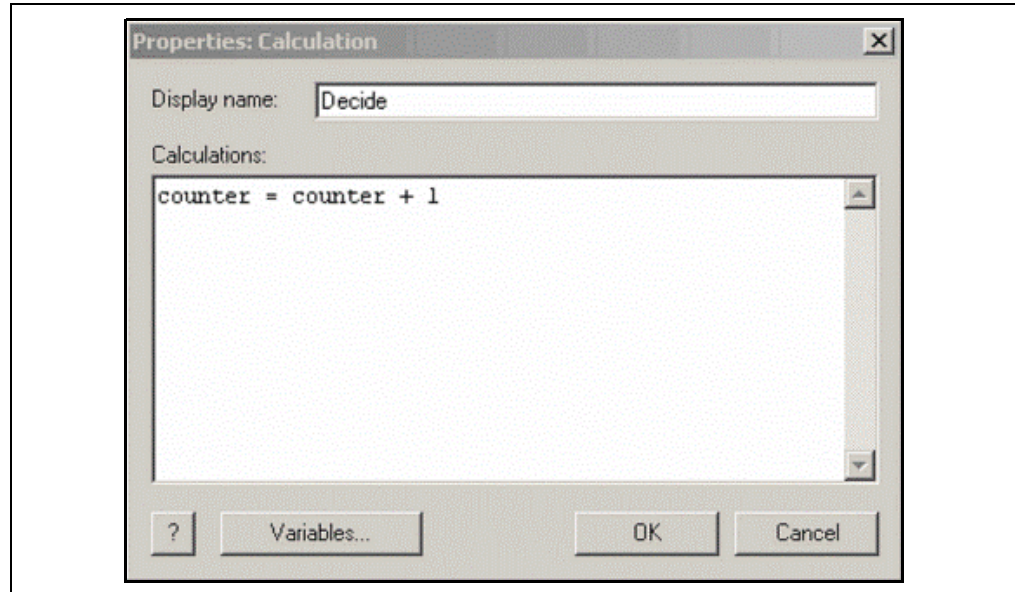
9. In the main flowchart window, drag/drop a **Loop** icon immediately beneath the **Interrupt** icon to create the Infinite loop used in all the previous labs.

10. Drag/drop a new **Calculation** icon within the loop boundaries and double-click to open the Properties: Calculation window. Change the "Display name:" to `Decide` and add the following code that will increment the `counter` variable each time through the loop to the "Calculations:" section:

```
counter = counter + 1
```

The Properties: Calculation window should now resemble Figure 3-20.

**FIGURE 3-20: PROPERTIES: CALCULATION CONFIGURATION FOR DECIDE FUNCTIONAL BLOCK**

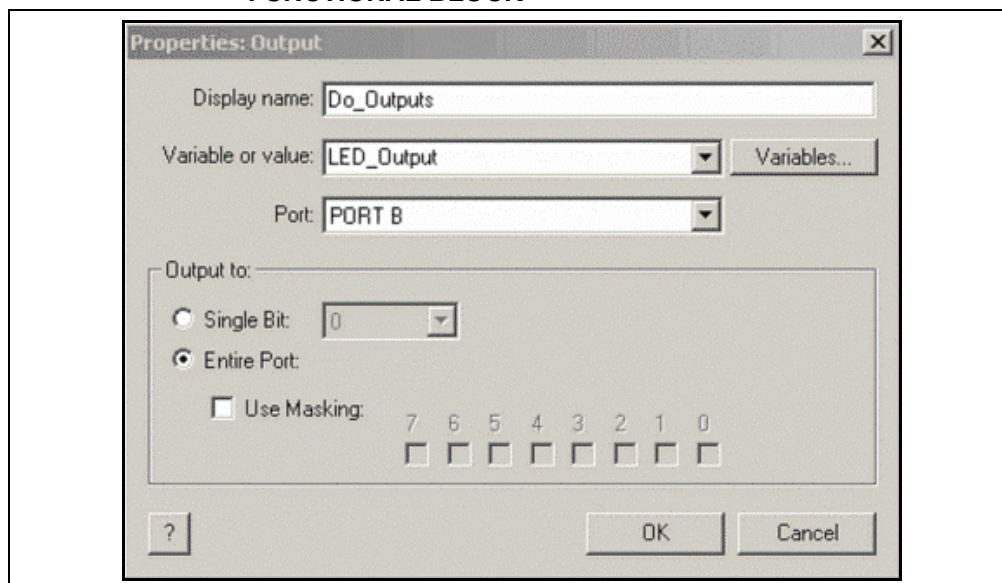


Click **OK** to continue.

11. Next, to assign the `LED_Output` variable to the LEDs connected to `PORTB`, drag/drop an **Output** icon immediately underneath the `Decide` functional block created in the previous step. Double-click the **Output** icon to open the Properties: Output. Change the "Display name:" to `Do_Outputs`, select the `LED_Output` variable from the "Variable or value:" drop-down menu and select **PORTB** from the "Port:" drop-down menu as the destination for the value in `LED_Output`. The Properties: Output window should now resemble Figure 3-21.



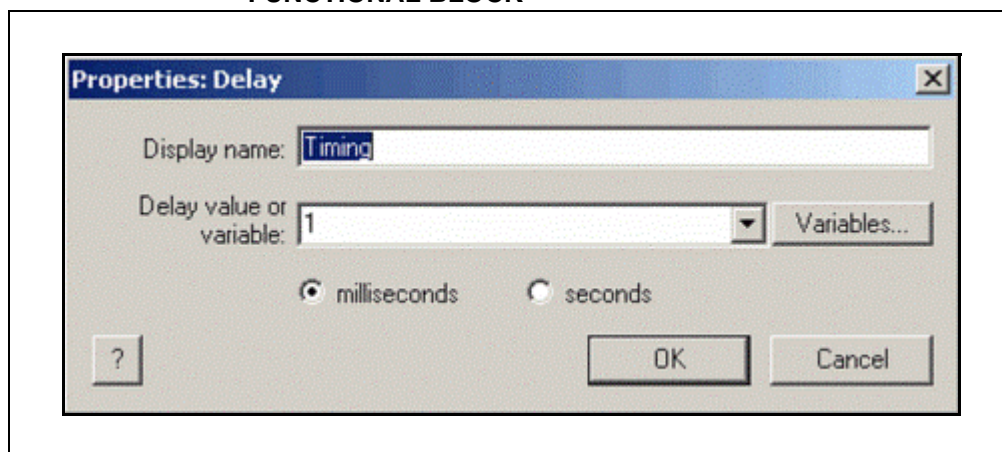
**FIGURE 3-21: PROPERTIES: OUTPUT WINDOW FOR DO\_OUTPUTS FUNCTIONAL BLOCK**



Click **OK** to continue.

12. Drag/drop a **Delay** icon immediately beneath the `Do_Outputs` functional block. Double-click the icon, rename to `Timing` and configure for 1mS delay as shown in Figure 3-22.

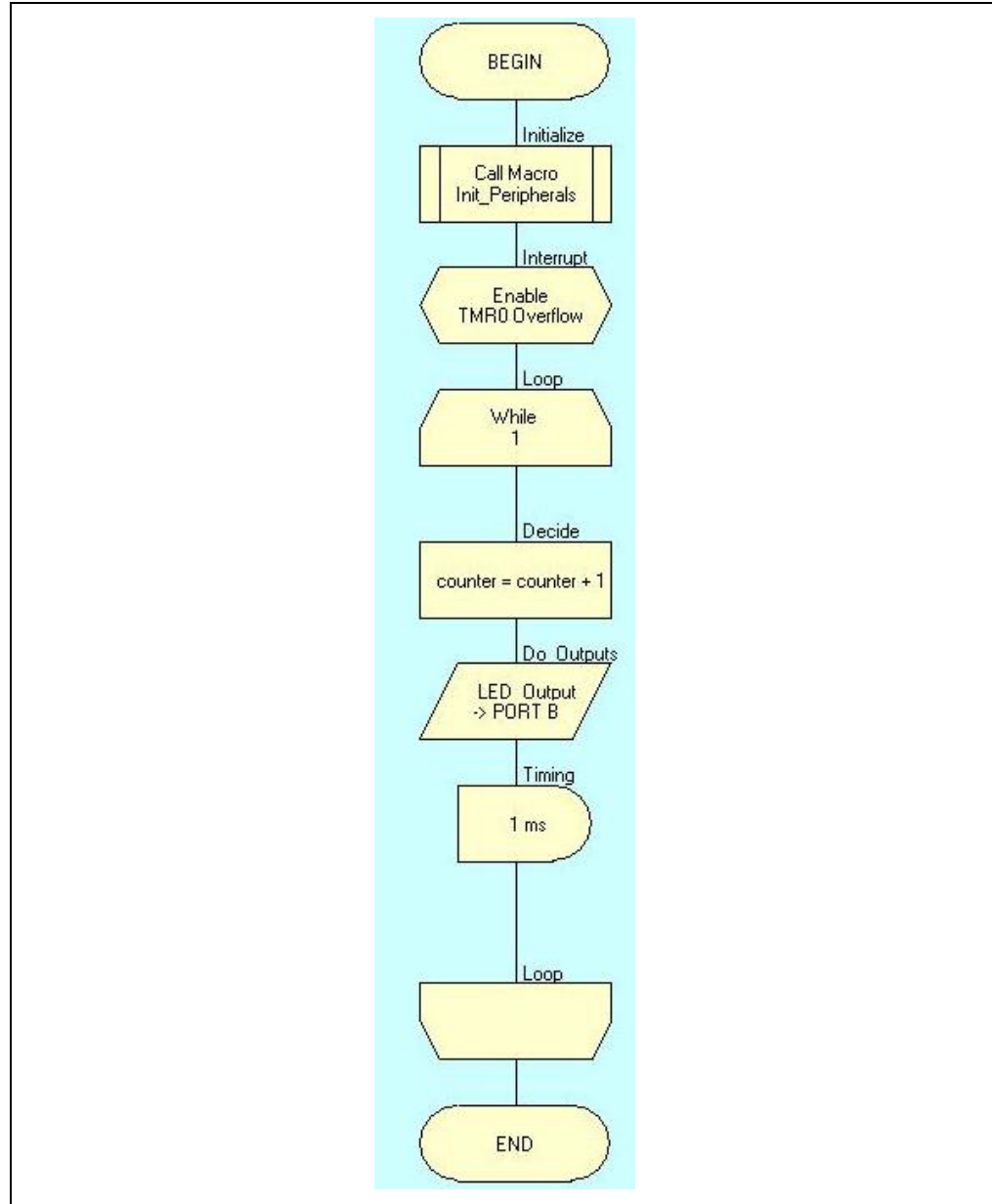
**FIGURE 3-22: PROPERTIES: DELAY CONFIGURATION FOR TIMING FUNCTIONAL BLOCK**



Click **OK** to continue.

The final flowchart for `Comparator_lab3` should now resemble Figure 3-23.

FIGURE 3-23: FINAL FLOWCHART FOR COMPARATOR\_LAB3



13. Compile the project to the PIC16F690. There should be no errors.

### 3.2.6.4 TESTING THE APPLICATION

Once programmed, the three LEDs connected to the PORTB pins RB4, RB5, RB6 and RB7 should display a 4-bit binary value. Touching the thermistor should introduce body heat, reduce the effective resistance and decrease the binary value displayed. Introducing cold should result in an increased binary result.

**Note:** If the application does not behave as detailed above, adjusting the potentiometer (R1) in the voltage divider can be used to alter the Comparator 2 “trip” level by changing the voltage on the non-inverting reference input.

The solution for this project can be found in the

C:\PICDEM\_Lab\Flowcode\Comparator\_Labs\Comparator\_Lab3\solution directory.



# PICDEM™ Lab Flowcode Companion Guide

---

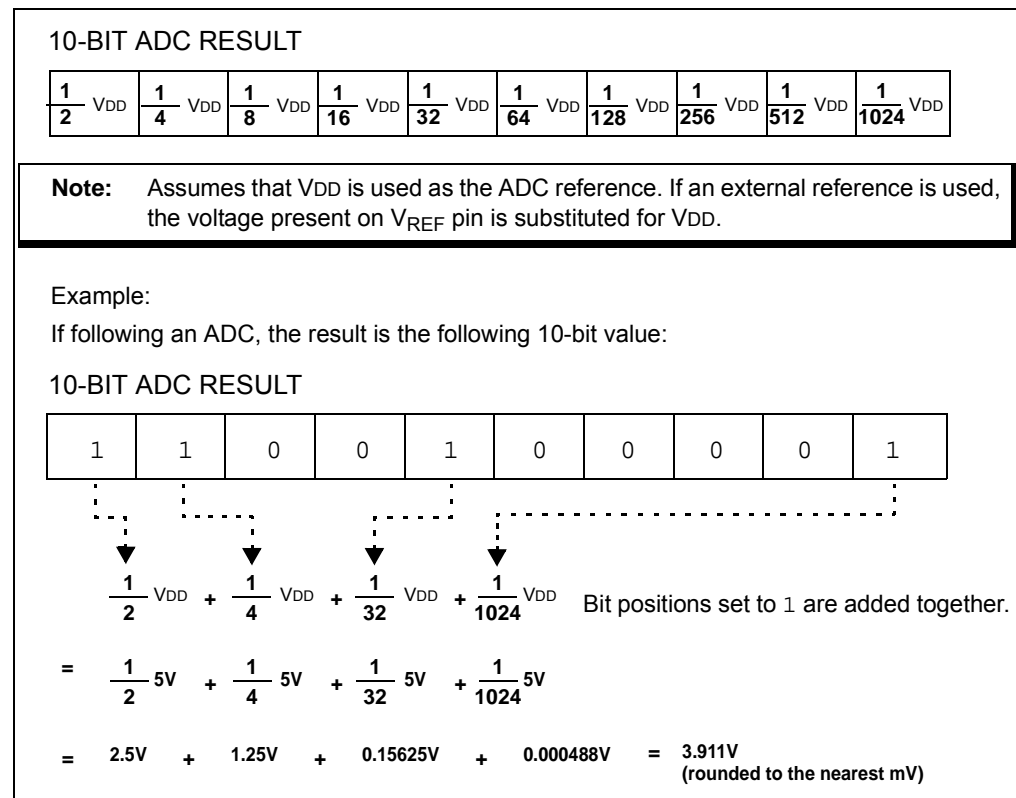
NOTES:

## Chapter 4. Analog-to-Digital Converter Peripheral Labs

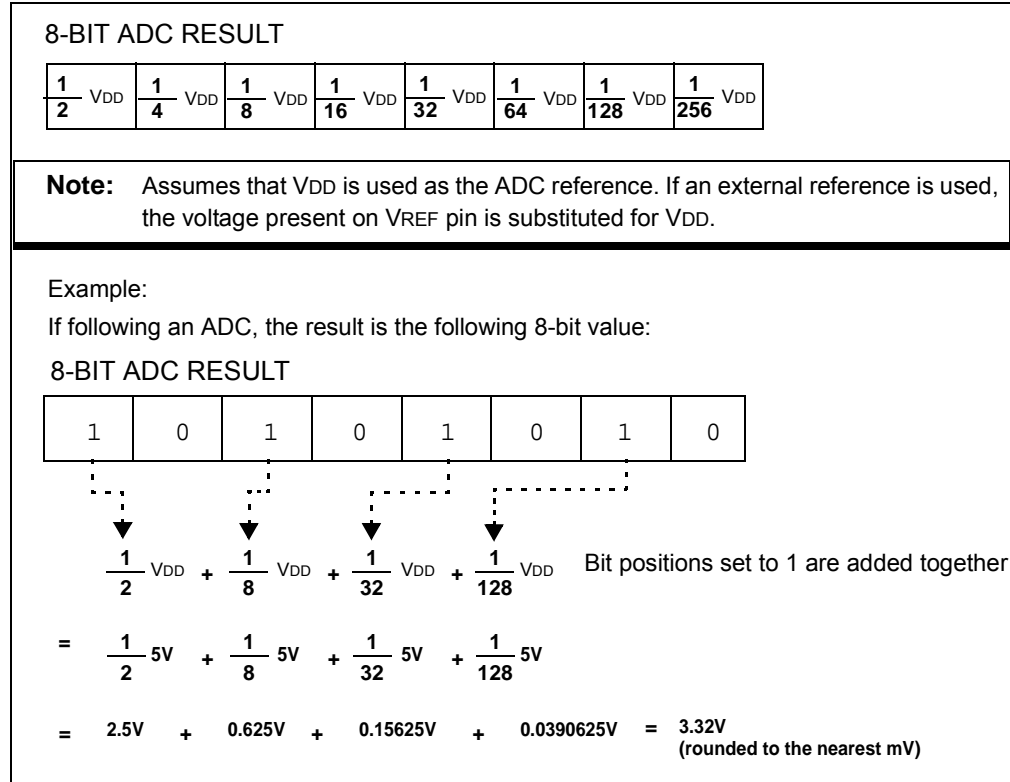
### 4.1 INTRODUCTION

The Analog-to-Digital Converter (ADC) peripheral allows conversion of an analog input voltage by comparing it to a reference voltage and through the use of successive approximation to convert the voltage level to a 10-bit binary value so that it can be used in firmware. This reference voltage can be an external input on the RA1/AN1/VREF pin (see data sheet if a different device than the PIC16F690 is used) or the VDD voltage used internally. The ADC result as compared to a reference voltage is demonstrated in Figure 4-1 and Figure 4-2.

**FIGURE 4-1: 10-BIT ADC RESULT BIT SIGNIFICANCE**



**FIGURE 4-2: 8-BIT ADC RESULT BIT SIGNIFICANCE**



The following labs will implement an 8-bit result using the  $V_{DD}$  voltage as a reference.

## 4.2 ADC LABS

The labs that will be implemented in this chapter are:

- Lab 1: Simple ADC
- Lab 2: Audible Temperature Sensor

### 4.2.1 Reference Documentation

Documentation is available as a free download from [www.microchip.com](http://www.microchip.com):

- DS41262D – PIC16F690 data sheet
  - Section 4: I/O Ports
  - Section 9: Analog-to-Digital Converter (ADC) Module

### 4.2.2 Equipment Required

To complete the labs in this section, the following components are required:

1. 1 – 100Ω resistor
2. 8 – 470Ω resistors
3. 1 – 1 KΩ resistor
4. 1 – 10 KΩ resistor
5. 1 – 100 KΩ potentiometer
6. 1 – 10 KΩ NTC Thermistor
7. 8 – Light Emitting Diodes
8. 1 – IRFD010 N-Channel MOSFET
9. PIC16F690 populating socket U2
10. Assorted jumper wires

## 4.2.3 Lab 1: Simple ADC

### 4.2.3.1 NEW REGISTERS USED IN THIS LAB

These registers are configured in the background automatically by the Flowcode Software using the **ADC** component and do not require any user manipulation:

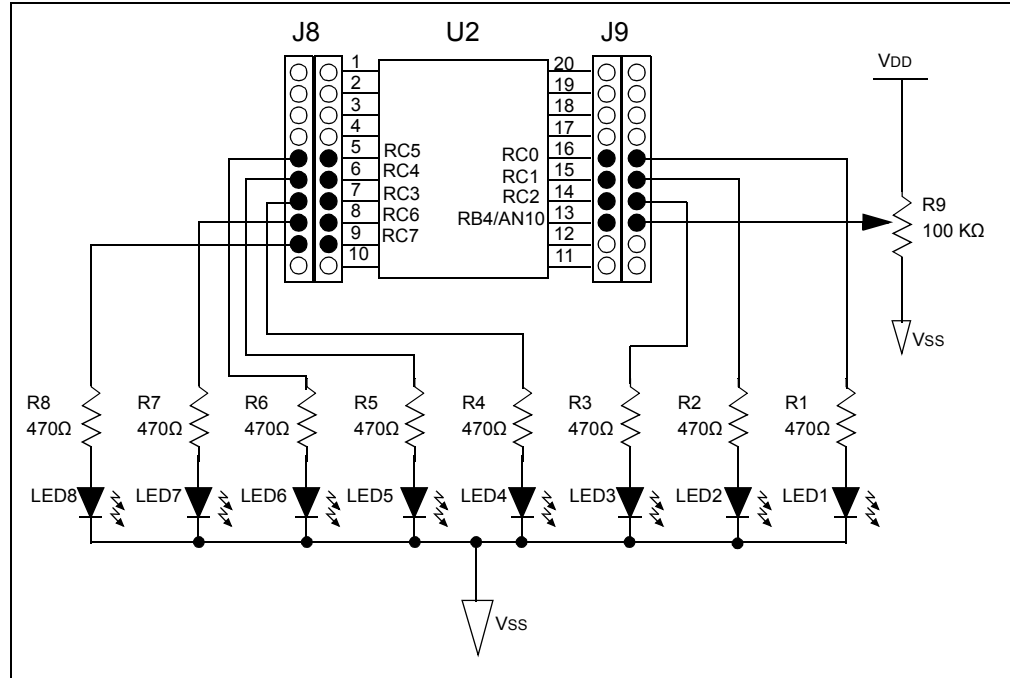
1. ADC Control Register 0: ADCON0 (Register 9-1 in Section 9 of the PIC16F690 data sheet)
  - Configures ADC conversion result justification
  - Select ADC reference voltage
  - Selects ADC input channel (i.e., pin with analog voltage to be converted)
  - Starts ADC conversion and determines when ADC conversion is complete
  - Enables the ADC peripheral
2. ADC Control Register 1: ADCON1 (Register 9-2 in Section 9 of the PIC16F690 data sheet)
  - Determines ADC conversion clock
3. ADC Result Register HIGH: ADRESH (see Register 9-3 in Section 9 of the PIC16F690 data sheet)
  - Holds upper 8-bits or upper 2-bits (depending on justification selected) of 10-bit ADC conversion result
4. ADC Result Register LOW: ADRESL (see Register 9-4 in Section 9 of the PIC16F690 data sheet)
  - Holds lower 8-bits or lower 2-bit (depending on justification selected) of 10-bit ADC conversion result

### 4.2.3.2 OVERVIEW

In this lab, the ADC peripheral on the PIC16F690 is used to perform a simple conversion of the analog voltage present on RB4/AN10 pin. The voltage is varied using a 100 K $\Omega$  potentiometer and compared against the V<sub>DD</sub> voltage internally. Flowcode features component specific macros for the ADC and will be used to capture the 8-bit representation of the voltage level. The 8-bit result will then be assigned to PORTC to light LEDs connected to its 8 pins for display.

The PICDEM Lab Development Board configuration schematic is shown in Figure 4-3.

FIGURE 4-3: SCHEMATIC FOR ADC LAB 1

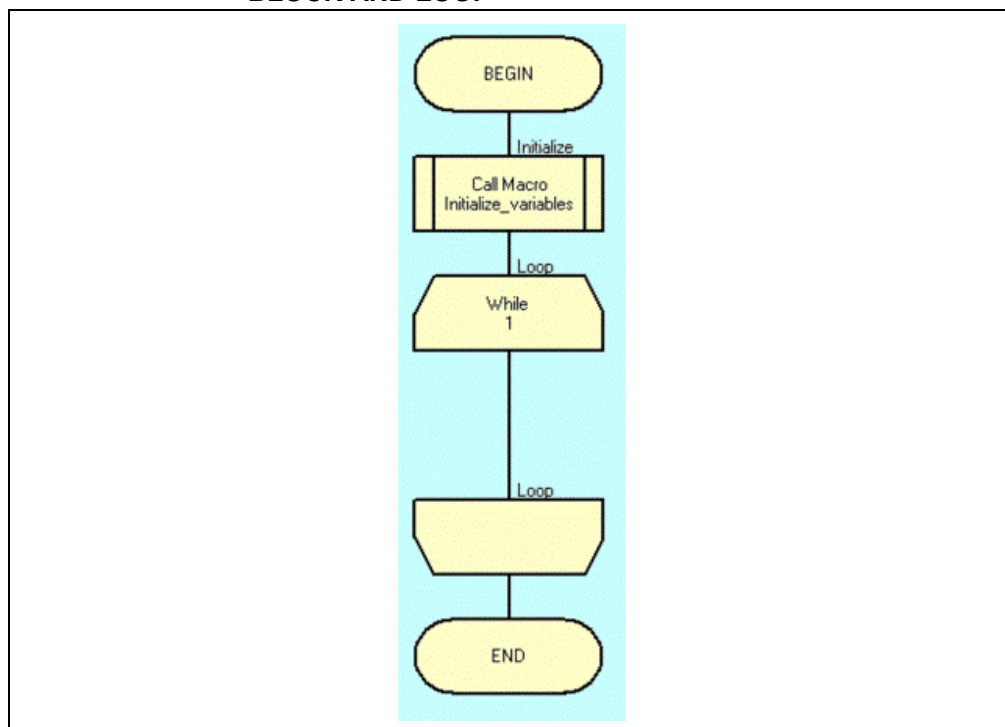


### 4.2.3.3 PROCEDURE

1. Create a new project in Flowcode using steps 1 through 9 from GPIO\_Lab1 saving the project as `ADC_Lab1.fcf` in the `C:\PICDEM_Lab\flowcode\ADC_Labs\ADC_Lab1` directory.
2. Create a new macro called `Init_variables`. Within the `Init_variables` macro, initialize an 8-bit variable, `LED_Output`, to zero using the **Calculation** icon and return to the main flowchart.
3. Drag/drop the **Macro** icon just beneath the BEGIN point in the flowchart, rename the icon to `Initialize` and call the `Init_variables` macro created in the previous steps.
4. Drag/drop the **Loop** icon into the flowchart beneath the `Initialize` functional block.  
The main flowchart should now resemble Figure 4-4.

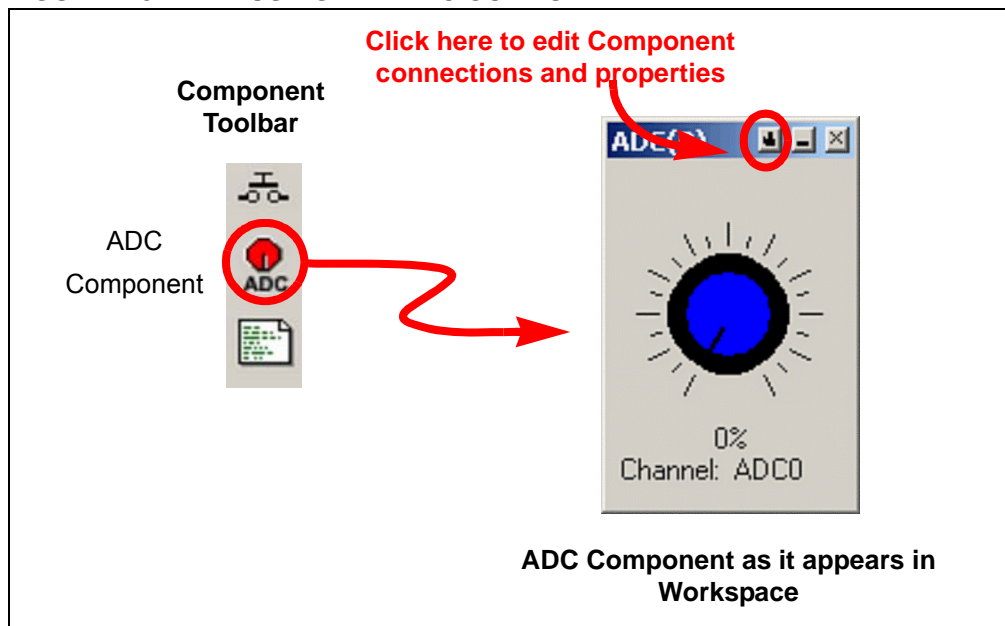
# Analog-to-Digital Converter Peripheral Labs

FIGURE 4-4: MAIN FLOWCHART WITH THE INITIALIZE FUNCTIONAL BLOCK AND LOOP



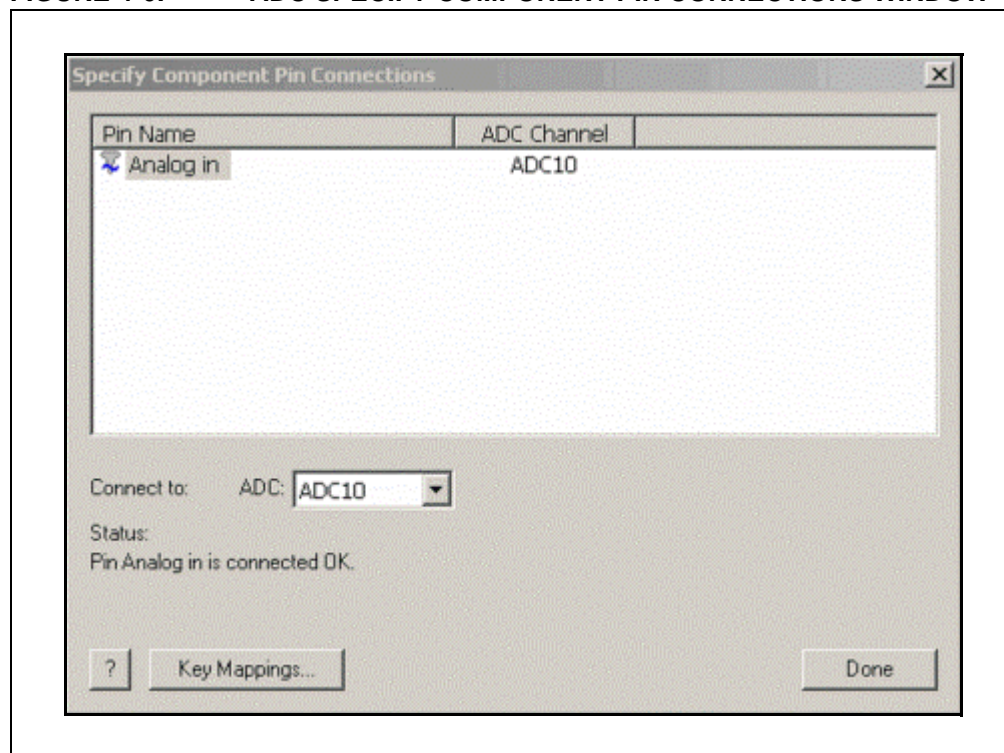
5. Next, click on the ADC component to add it to the flowchart workspace (see Figure 4-5).

FIGURE 4-5: USING THE ADC COMPONENT



6. Click on the small button on the component as shown in Figure 4-5 and select **Component Connections...** from the drop-down menu. The Specify Component Pin Connections window should now be open. Using the "Connect to: ADC:" drop-down menu, select **ADC10** to use the RB4/AN10 pin as the ADC input channel. The Specify Component Pin Connections window should now resemble Figure 4-6.

FIGURE 4-6: ADC SPECIFY COMPONENT PIN CONNECTIONS WINDOW

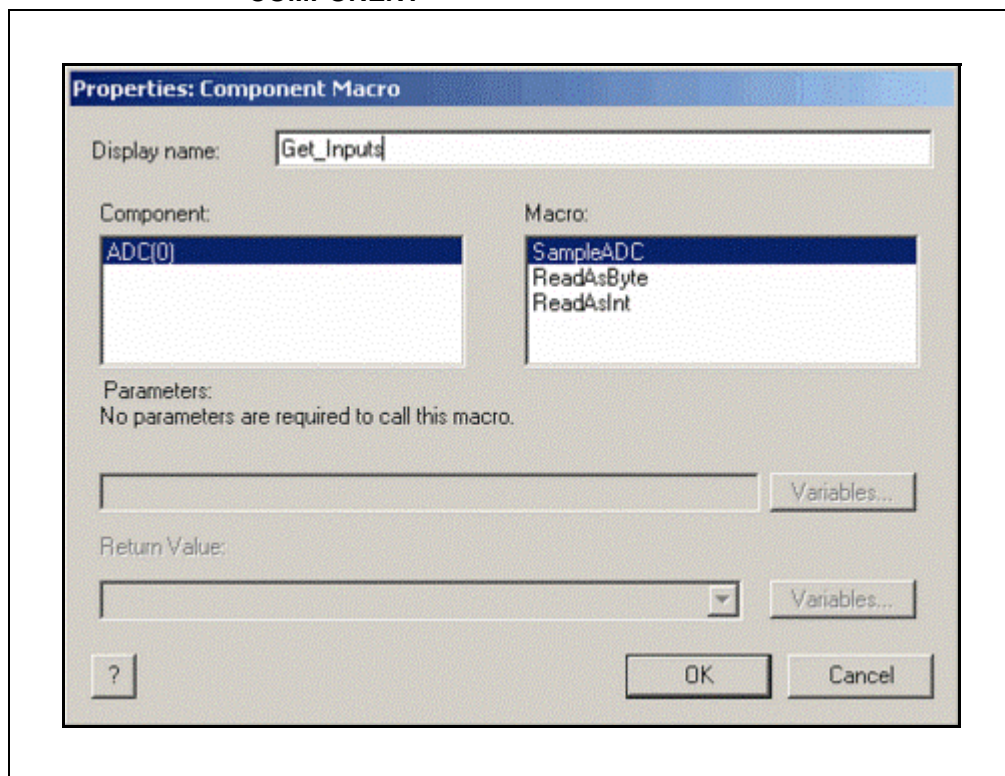


Click **Done** to exit and return to the main Flowchart window.

7. As mentioned in previous labs, each component features a number of predefined macros. In this lab, two macros specific to the ADC component will be used:
  - **SampleADC**: starts the ADC process, waits for completion and stores the result value in memory
  - **BYTE ReadValueAsByte**: Returns the 8 Most Significant bits of the 10-bit resulting ADC value. To return all 10-bits, the `INT ReadValueAsInt` macro would be used.The first step will be to build the `Get_Inputs` functional block that will initiate an ADC and obtain the 10-bit result.  
Drag/drop a **Component Macro** icon into the main flowchart within the loop.
8. Double-click the **Component Macro** icon to open the Properties: Component Macro window and rename to `Get_Inputs`. In the "Component:" list, select `ADC(0)` to reveal the available component macros. In the "Macro:" list, select the `SampleADC` macro (see Figure 4-7).



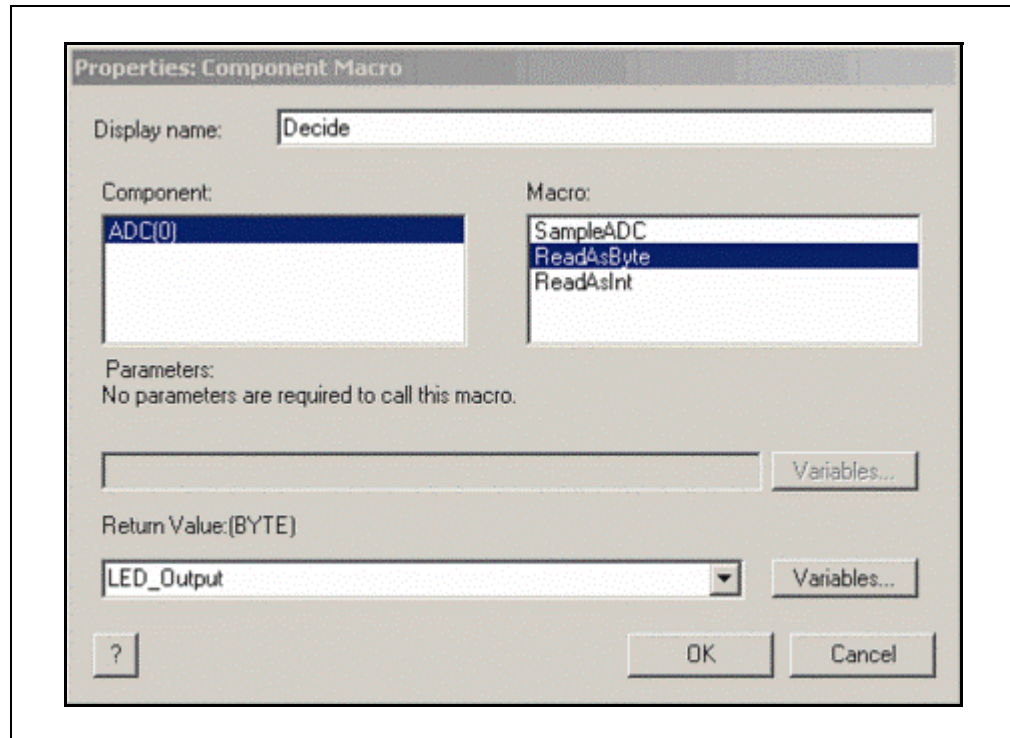
FIGURE 4-7: SELECTING THE `SAMPLEADC` MACRO FOR THE ADC COMPONENT



Click **OK** to exit and return to the main flowchart.

9. Next, the `Decide` functional block will again implement the **Component Macro** icon to assign 8-bits of the 10-bit result to the `LED_Output` initialized earlier. Drag/drop a **Component Macro** icon directly beneath the `Get_Inputs` functional block.
10. Double-click the **Component Macro**, rename `Decide` and do the following:
  - Highlight `ADC(0)` in the **Component** list.
  - Select the `ReadAsByte` macro from the "Macro:" list.
  - Select the `LED_Output` variable as the destination for the 8-bit ADC value from the "Return Value:(BYTE)" drop-down list.The Properties: Component Macro window should now resemble Figure 4-8.

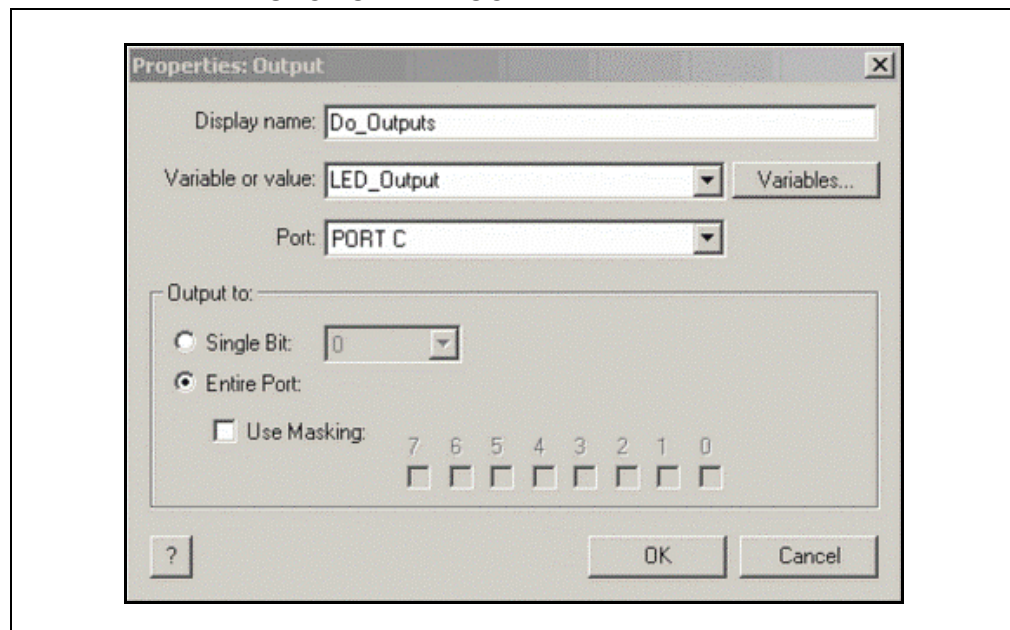
**FIGURE 4-8: PROPERTIES: COMPONENT MACRO WINDOW FOR DECIDE FUNCTIONAL BLOCK**



Click **OK** to return to the main Flowchart window.

11. The final block in the flowchart is to create the `Do_Outputs` functional block. The `LED_Output` value obtained in the previous step must now be assigned to the `PORTC` register for output to the LEDs connected to its associated pins. Drag/drop an **Output** icon immediately under the `Decide` functional block, rename to `Do_Outputs` and assign the `LED_Output` value to `PORTC` (Figure 4-9).

**FIGURE 4-9: OUTPUT ICON CONFIGURATION FOR DO\_OUTPUTS FUNCTIONAL BLOCK**

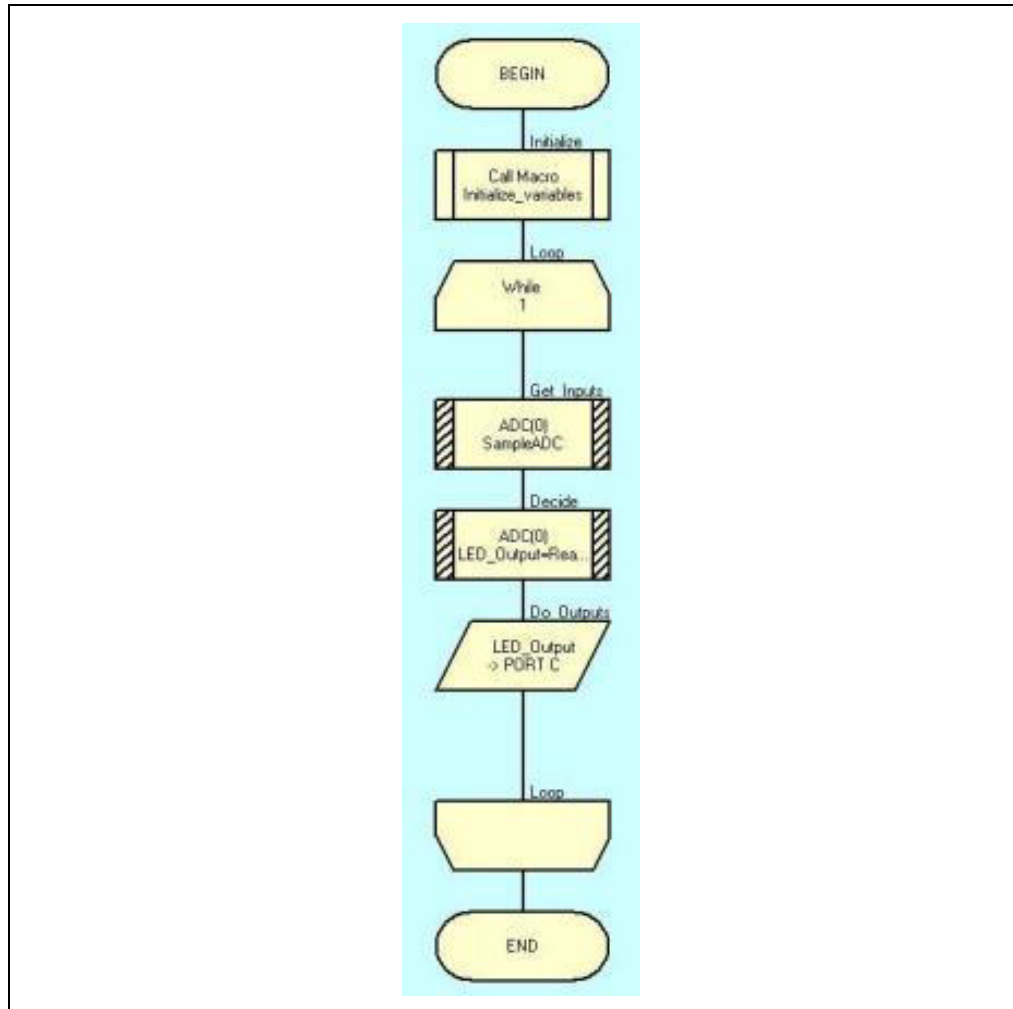


# Analog-to-Digital Converter Peripheral Labs

Click **OK** to continue.

The final flowchart should now resemble Figure 4-10.

**FIGURE 4-10: FINAL FLOWCHART FOR ADC\_LAB1**



Compile the project to chip. There should be no errors.

## 4.2.3.4 TESTING THE APPLICATION

Once programmed, turning the potentiometer connected to RB4/AN10 should light the LEDs sequentially in a binary fashion representing the ADC result value. The reader is encouraged to analyze these results as per the bit significance breakdown given in Figure 4-2.

The solution for this lab can be found in the

C:\PICDEM\_Lab\Flowcode\ADC\_Labs\ADC\_Lab1\solution directory.

## 4.2.4 Lab 2: Audible Temperature Sensor

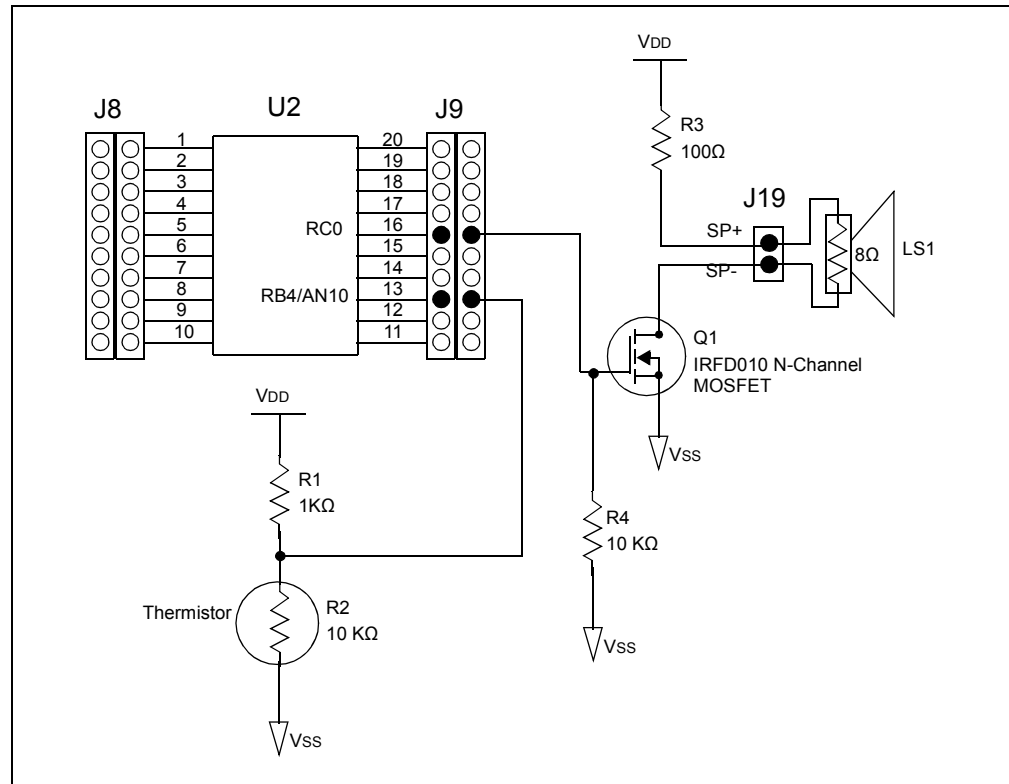
### 4.2.4.1 OVERVIEW

In this lab, the ADC peripheral on the PIC16F690 is used to alter the frequency of Pulse-Width Modulated Waveform (PWM) in relation to the temperature sensed by a thermistor connected to the input of the ADC peripheral. The PWM is used to drive an N-Channel MOSFET connected to the RC0 pin that will drive the speaker mounted on the PICDEM Lab Development Board. A resulting tone will be generated by the

speaker. Any change to the PWM frequency will result in a change in the frequency of the tone emitted by the speaker. The PWM waveform is generated by simply toggling the RC0 voltage level high-to-low or low-to-high each time through the main Software Control loop. The frequency of the PWM will be determined by a delay that will tie up the main Software loop for a period of time based on the value of the ADC result from a voltage divider implemented using the thermistor and a 1KΩ resistor. The ADC result captured will be decremented each time through the loop. Once the value reaches zero, the loop will break and the Software Control loop will continue to execute. Therefore, a higher resulting ADC value will generate a longer delay, decreasing the frequency of the toggling RC0 pin. A smaller ADC result will generate a shorter delay, thereby increasing the frequency of the toggling RC0 pin.

The schematic for this lab is shown in Figure 4-11.

**FIGURE 4-11: SCHEMATIC FOR ADC LAB 2**



Referring the schematic in Figure 4-11, the RC0 pin connects directly to the gate of the IRFD010 N-Channel MOSFET Q1. Notice that the PIC16F690 output pins can directly drive MOSFET gates without the need of any driver circuitry. Resistor R4 pulls the gate input low ensuring the transistor will remain OFF until a high voltage level is present on the RC0 output. When the PWM transitions high, Q1 is ON and current flows through the 8Ω speaker. The 100Ω R3 resistor is used to limit the current through the speaker to manufacturer specified power ratings.

#### 4.2.4.2 PROCEDURE

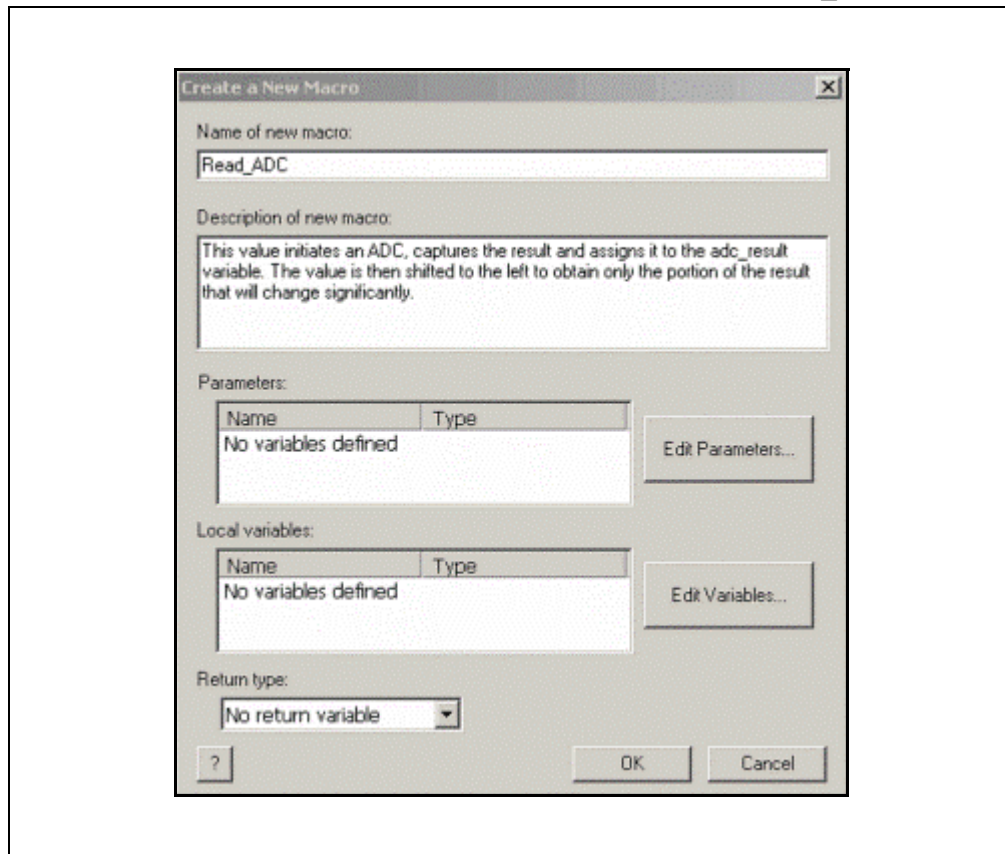
Using the firmware developed in the previous lab, make the following changes:

1. Create a new project in Flowcode using steps 1 through 9 from `GPIO_Lab1` saving the project as `ADC_Lab2.fcf` in the `C:\PICDEM_Lab\flowcode\ADC_Labs\ADC_Lab2` directory.

# Analog-to-Digital Converter Peripheral Labs

2. Create the `Initialize` functional block using the **Calculation** icon and initialize two 8-bit variables as follows:
  - `toggle = 0b11111111`
  - `adc_result = 0`.
3. Next, add a **Loop** icon to implement the Infinite loop used in all preceding labs.
4. Create a new macro called `Read_ADC` and provide a description similar to that shown in Figure 4-12.

**FIGURE 4-12: CREATE A NEW MACRO WINDOW FOR `READ_ADC`**

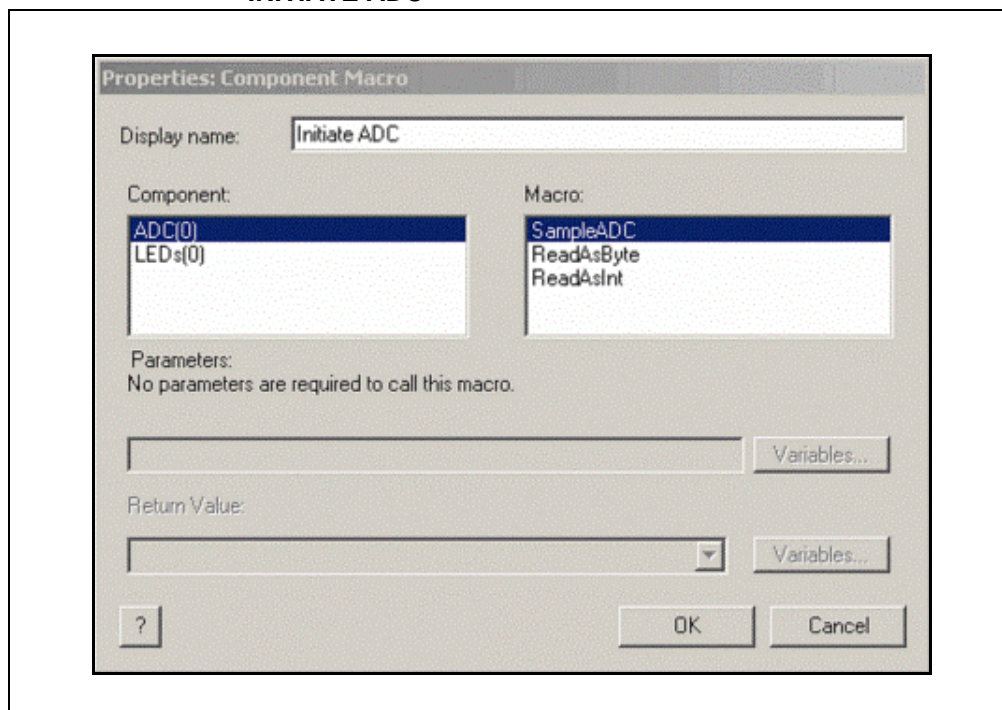


Click **OK** to open the `Read_ADC` Flowchart window.

5. Ensure that the **ADC Component** is selected and appears in the workspace. Drag/drop two **Component Macro** icons into the flowchart. Double-click on the top icon to open the Properties window and rename to `Initiate ADC`. Select the `SampleADC` macro as shown in Figure 4-13.



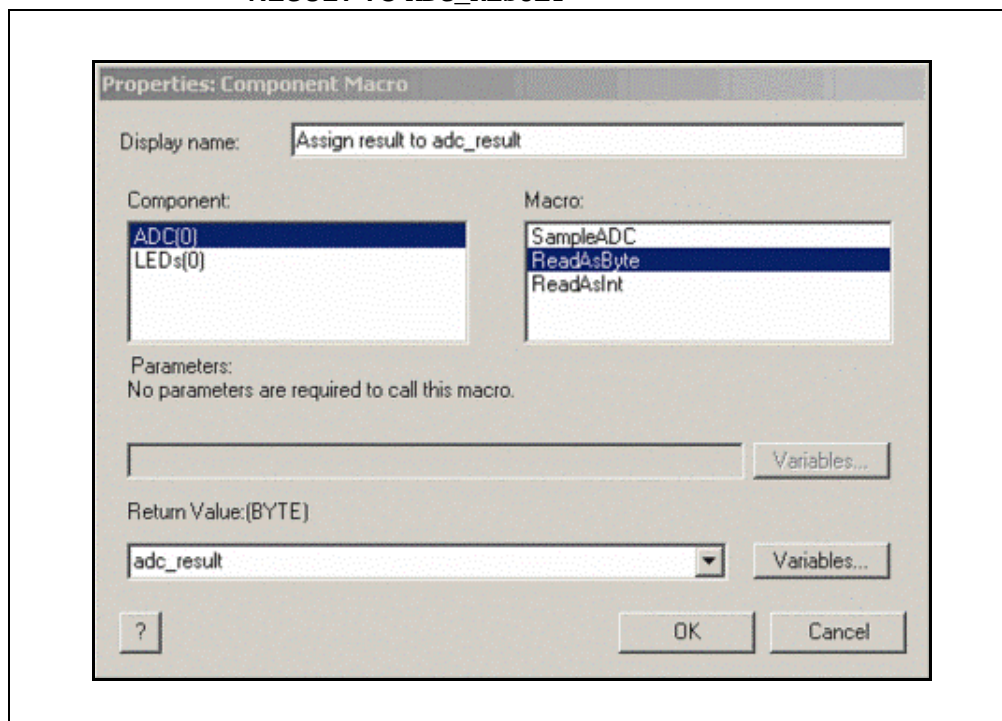
**FIGURE 4-13: PROPERTIES: COMPONENT MACRO WINDOW FOR INITIATE ADC**



Click **OK** to continue.

6. Double-click the second **Component** macro to edit the properties. Rename to `Assign result to adc_result`, select the `ReadAsByte` macro and assign to the `adc_result` variable created in the `Initialize` functional block (see Figure 4-14).

**FIGURE 4-14: PROPERTIES: COMPONENT MACRO WINDOW FOR ASSIGN RESULT TO ADC\_RESULT**

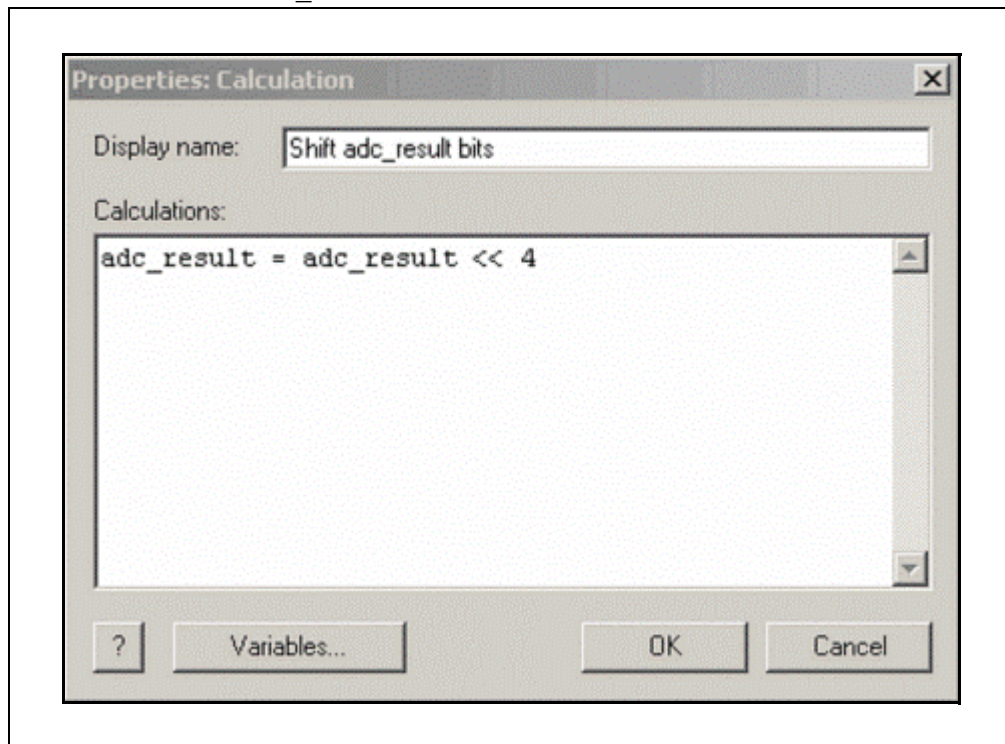


# Analog-to-Digital Converter Peripheral Labs

Click **OK** to continue.

- Next, drag/drop a **Calculation** icon immediately beneath the **Component Macro** icons from the previous steps. Double-click the **Calculation** icon rename to **Shift adc\_result bit** and enter the code to shift the `adc_result` variable bits by four to the left (see Figure 4-15).

**FIGURE 4-15: PROPERTIES: CALCULATION WINDOW FOR SHIFT ADC\_RESULT BITS**



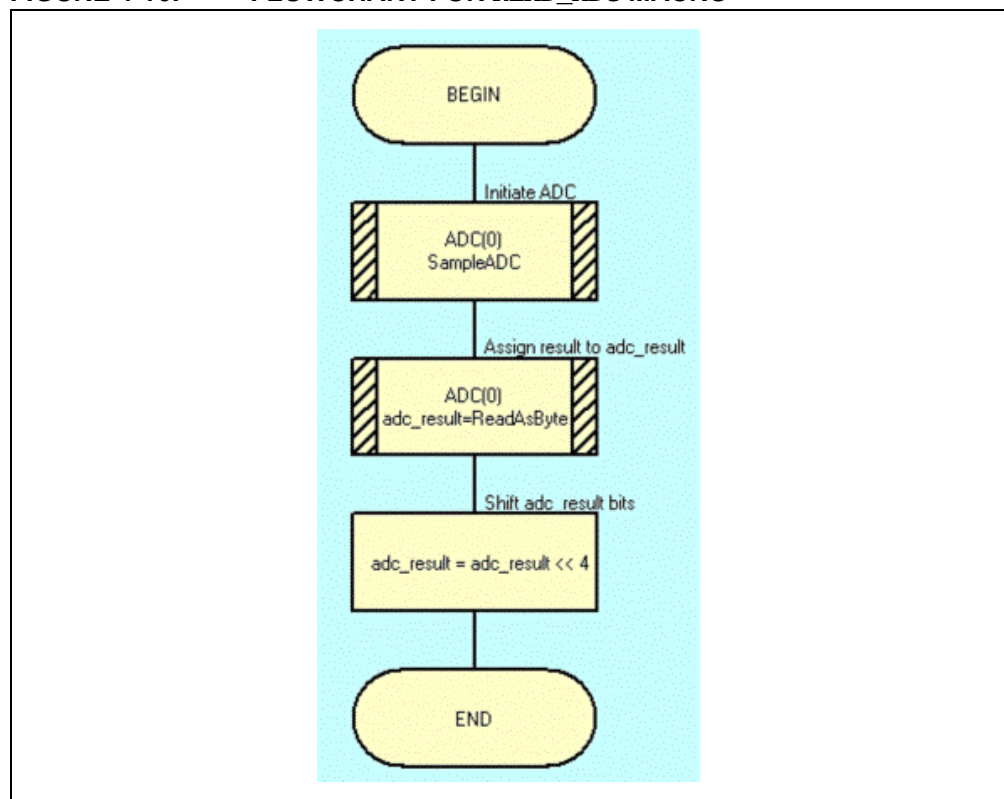
Click **OK** to continue.

**Note:** The `adc_result` bits are shifted to provide a more significant response to the change in ADC result from the resistor/thermistor voltage divider. Once comfortable with the contents of this lab, the reader is encouraged to manipulate this code and observe the results.

The `Read_ADC` macro flowchart should now resemble Figure 4-16.

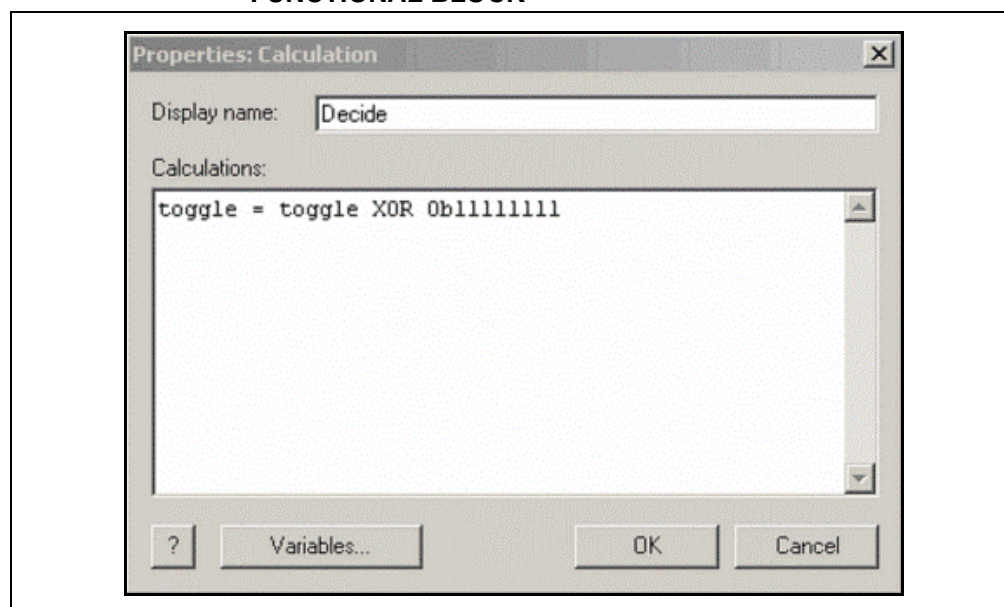


FIGURE 4-16: FLOWCHART FOR READ\_ADC MACRO



8. In the main flowchart, drag/drop a **Macro** icon with the main loop. Double-click on the icon to open the Properties window, rename `Get_Inputs` and call the `Read_ADC` macro created in the previous steps.
9. Immediately beneath the `Get_Inputs` functional block, drag/drop a new **Calculation** icon and double-click to open the Properties window. In the Properties window, rename the icon to `Decide` and enter the code to toggle the bits within the `toggle` variable (see Figure 4-17).

FIGURE 4-17: PROPERTIES: CALCULATION WINDOW FOR DECIDE FUNCTIONAL BLOCK

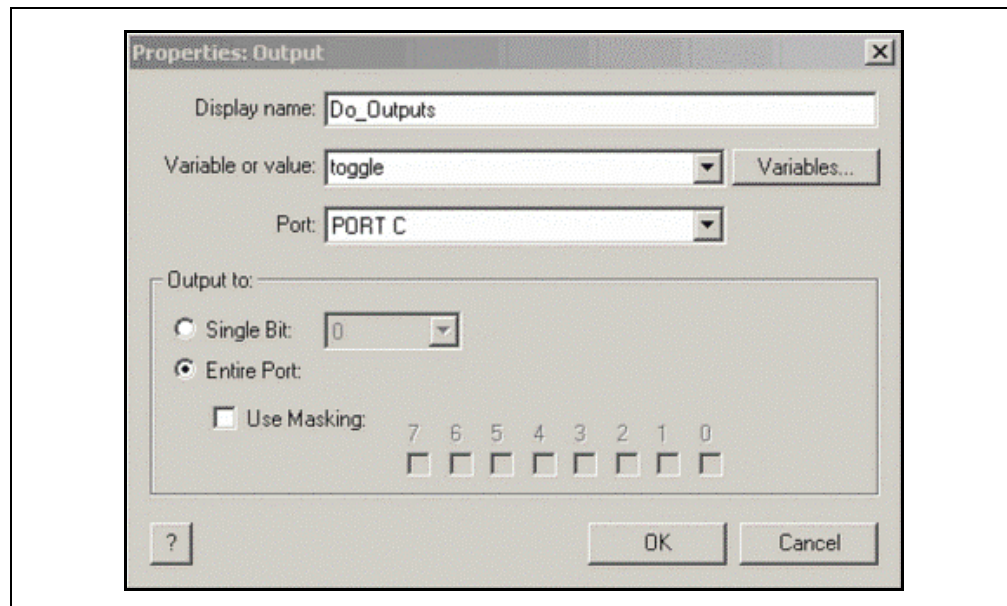


# Analog-to-Digital Converter Peripheral Labs

Click **OK** to continue.

- Next, drag/drop an **Output** icon to the flowchart immediately following the **Decide** functional block. Double-click on the icon to open the Properties window, rename the icon to `Do_Outputs`, select `PORTC` as the port and assign the `toggle` variable as the “Variable or value:” (see Figure 4-18).

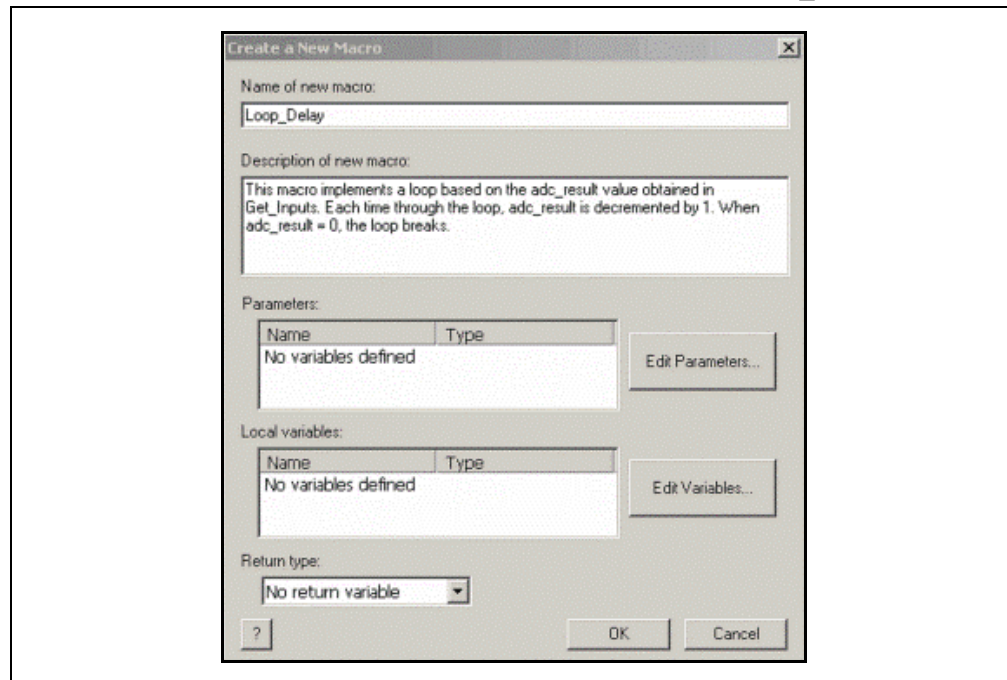
**FIGURE 4-18: PROPERTIES: OUTPUT WINDOW FOR THE DO\_OUTPUTS FUNCTIONAL BLOCK**



Click **OK** to continue.

- Next, the Delay loop that will vary the frequency of the RC0 toggle will be created. Create a new macro called `Loop_Delay` and provide a description similar to that shown in Figure 4-19.

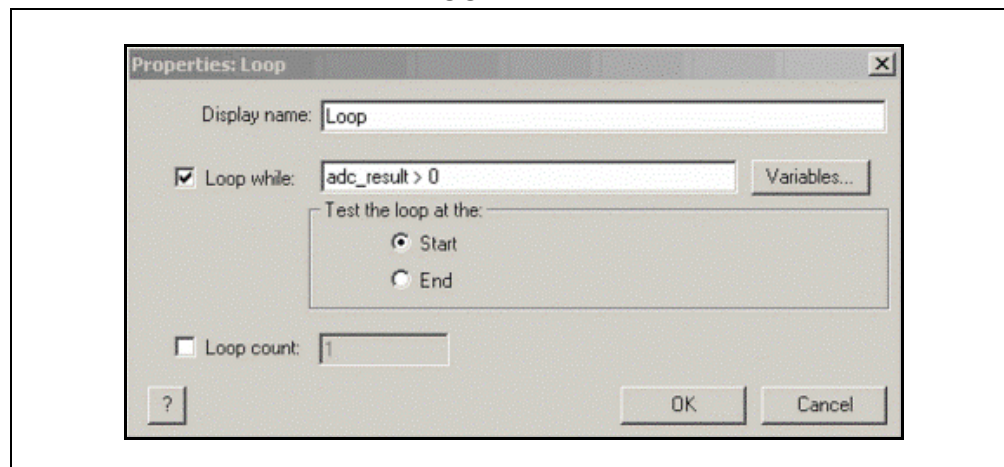
**FIGURE 4-19: CREATE A NEW MACRO WINDOW FOR LOOP\_DELAY MACRO**



Click **OK** to open the `Loop_Delay` macro flowchart.

12. Drag/drop a **Loop** icon into the flowchart for the `Loop_Delay` macro. Double-click on the icon to open the Properties window. In the Properties window, ensure that the “Loop while:” check box is selected. The `adc_result` variable will be used as the condition for the loop. So long as the `adc_result` value is greater than zero, the loop will execute and no other code or functional blocks will be executed. Once the `adc_result` is equal to zero, the loop will break and the rest of the Software Control loop can then continue to execute. Configure the “Properties: Loop” as shown in Figure 4-20.

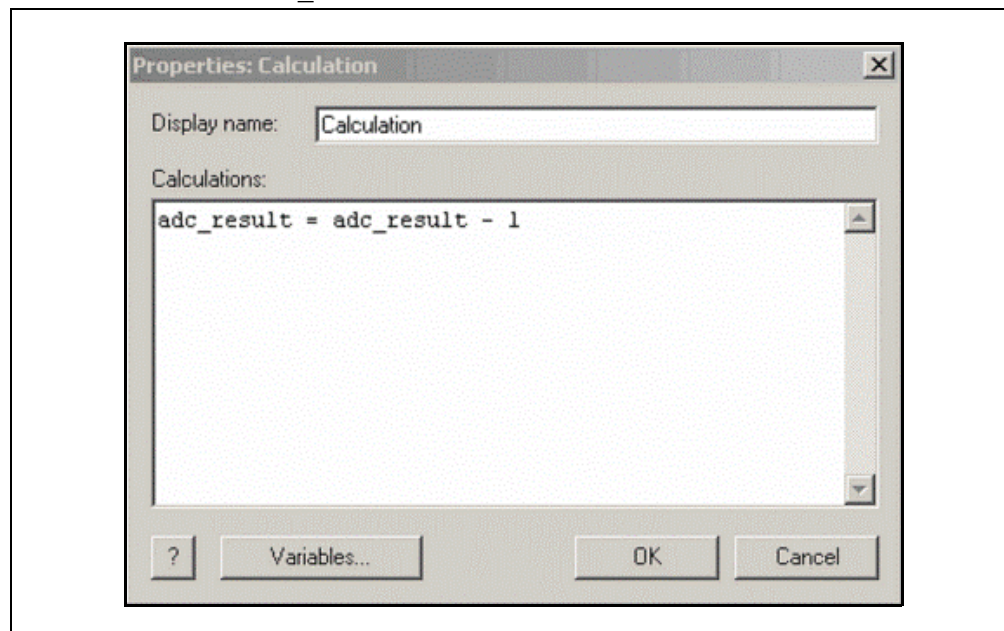
**FIGURE 4-20: PROPERTIES: LOOP FOR `ADC_RESULT` VARIABLE DEPENDANT LOOP**



Click **OK** to continue.

13. To decrement the `adc_result` variable each time through the loop, drag/drop a **Calculation** icon into the newly created loop and add the code shown in Figure 4-21.

**FIGURE 4-21: PROPERTIES: CALCULATION WINDOW TO DECREMENT `ADC_RESULT` VARIABLE**

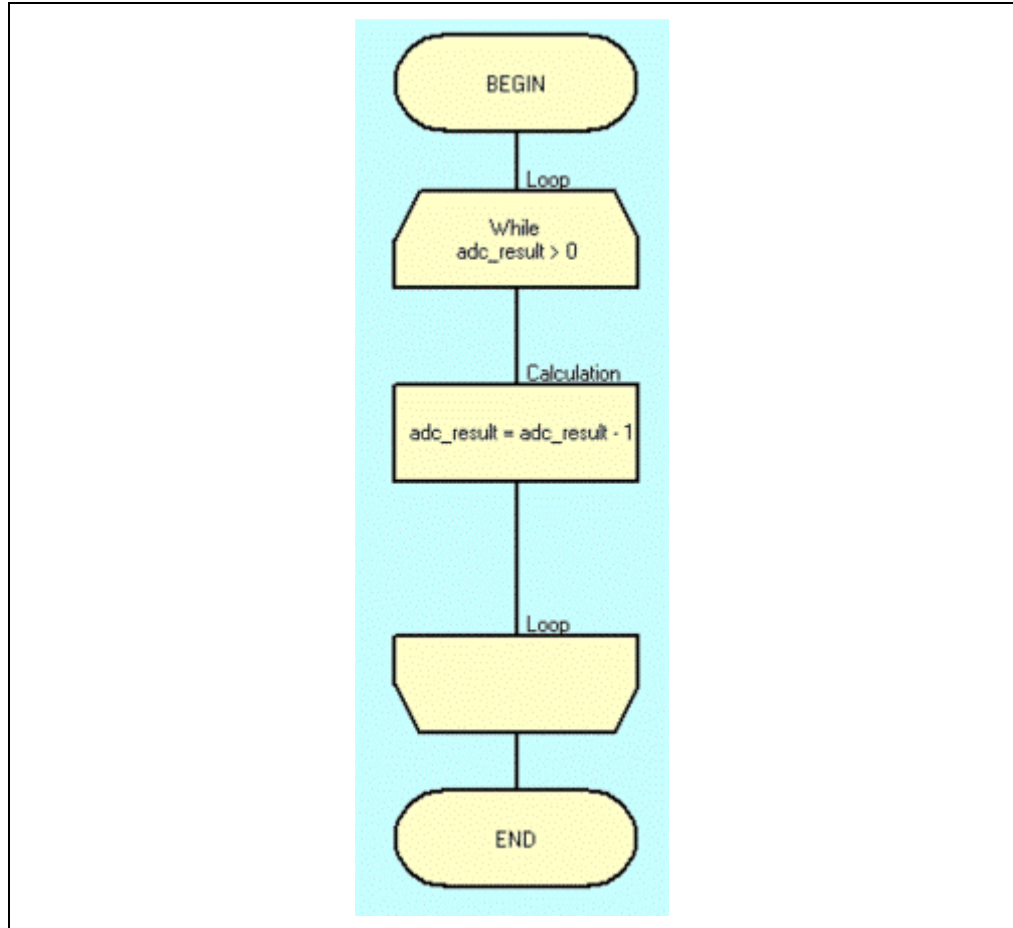


# Analog-to-Digital Converter Peripheral Labs

Click **OK** to continue.

The final flowchart for the `Loop_Delay` macro should now resemble Figure 4-22.

**FIGURE 4-22: FINAL FLOWCHART FOR LOOP\_DELAY MACRO**

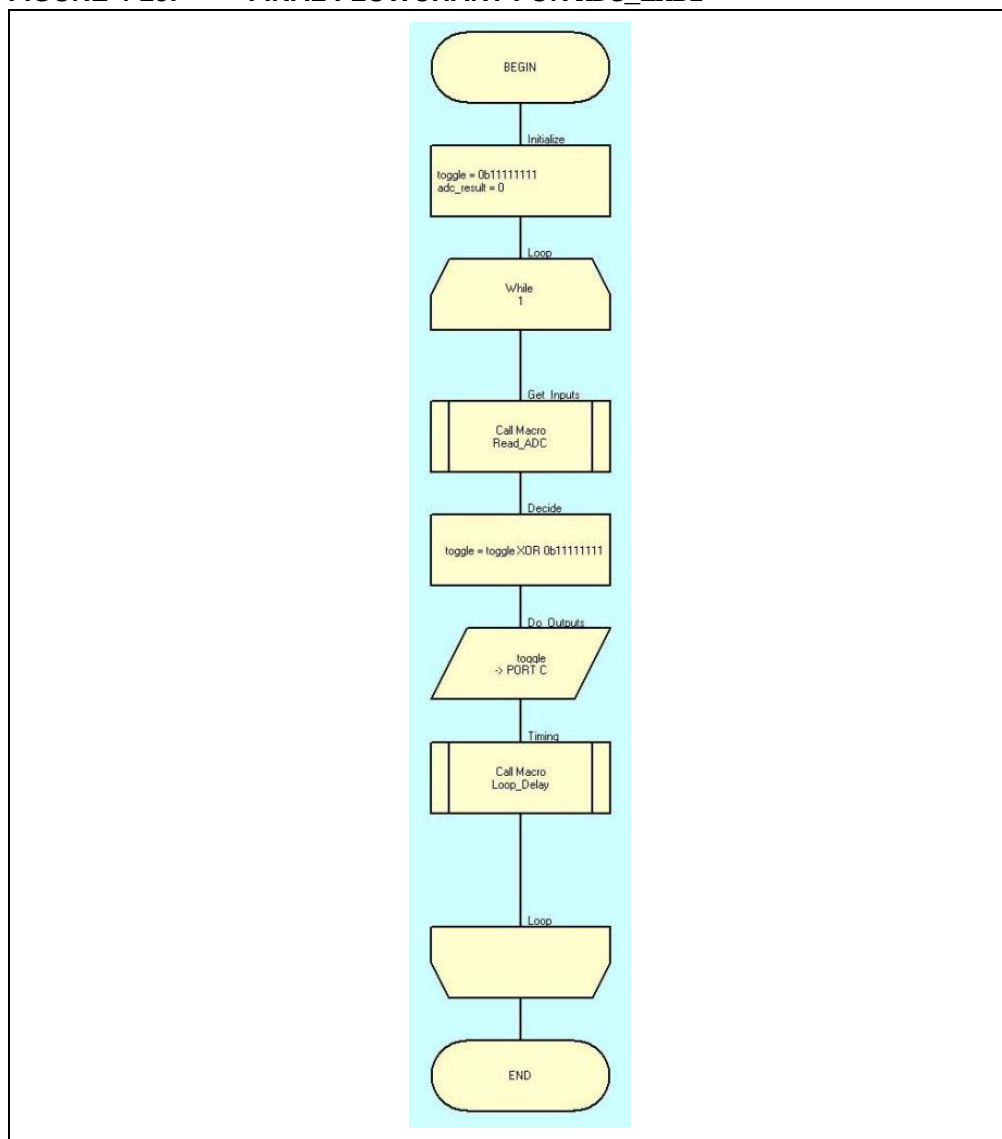


14. In the main Flowchart window, drag/drop a **Macro** icon within the main loop immediately following the `Do_Outputs` functional block. Double-click on the icon to open the Properties window and rename to `Timing` and call the `Loop_Delay` macro created in the previous steps. Click **OK** to close the window.

The main flowchart for this application should now resemble Figure 4-23.



FIGURE 4-23: FINAL FLOWCHART FOR ADC\_LAB2



Compile the project to chip, there should be no errors.

#### 4.2.4.3 TESTING THE APPLICATION

Once the PIC16F690 is programmed, an audible tone should emit from the speaker. Pinching the thermistor should introduce body heat to the component thereby increasing the frequency of the speaker output. Colder temperature sources applied to the thermistor should reduce the speaker output frequency.

The solution for this project can be found in the

C:\PICDEM\_Lab\Flowcode\ADC\_Labs\ADC\_Lab2\solution directory.

# PICDEM™ Lab Flowcode Companion Guide

---

---

NOTES:



## WORLDWIDE SALES AND SERVICE

### AMERICAS

#### Corporate Office

2355 West Chandler Blvd.  
Chandler, AZ 85224-6199  
Tel: 480-792-7200  
Fax: 480-792-7277  
Technical Support:  
<http://support.microchip.com>  
Web Address:  
[www.microchip.com](http://www.microchip.com)

#### Atlanta

Duluth, GA  
Tel: 678-957-9614  
Fax: 678-957-1455

#### Boston

Westborough, MA  
Tel: 774-760-0087  
Fax: 774-760-0088

#### Chicago

Itasca, IL  
Tel: 630-285-0071  
Fax: 630-285-0075

#### Cleveland

Independence, OH  
Tel: 216-447-0464  
Fax: 216-447-0643

#### Dallas

Addison, TX  
Tel: 972-818-7423  
Fax: 972-818-2924

#### Detroit

Farmington Hills, MI  
Tel: 248-538-2250  
Fax: 248-538-2260

#### Kokomo

Kokomo, IN  
Tel: 765-864-8360  
Fax: 765-864-8387

#### Los Angeles

Mission Viejo, CA  
Tel: 949-462-9523  
Fax: 949-462-9608

#### Santa Clara

Santa Clara, CA  
Tel: 408-961-6444  
Fax: 408-961-6445

#### Toronto

Mississauga, Ontario,  
Canada  
Tel: 905-673-0699  
Fax: 905-673-6509

### ASIA/PACIFIC

#### Asia Pacific Office

Suites 3707-14, 37th Floor  
Tower 6, The Gateway  
Harbour City, Kowloon  
Hong Kong  
Tel: 852-2401-1200  
Fax: 852-2401-3431

#### Australia - Sydney

Tel: 61-2-9868-6733  
Fax: 61-2-9868-6755

#### China - Beijing

Tel: 86-10-8528-2100  
Fax: 86-10-8528-2104

#### China - Chengdu

Tel: 86-28-8665-5511  
Fax: 86-28-8665-7889

#### China - Hong Kong SAR

Tel: 852-2401-1200  
Fax: 852-2401-3431

#### China - Nanjing

Tel: 86-25-8473-2460  
Fax: 86-25-8473-2470

#### China - Qingdao

Tel: 86-532-8502-7355  
Fax: 86-532-8502-7205

#### China - Shanghai

Tel: 86-21-5407-5533  
Fax: 86-21-5407-5066

#### China - Shenyang

Tel: 86-24-2334-2829  
Fax: 86-24-2334-2393

#### China - Shenzhen

Tel: 86-755-8203-2660  
Fax: 86-755-8203-1760

#### China - Wuhan

Tel: 86-27-5980-5300  
Fax: 86-27-5980-5118

#### China - Xiamen

Tel: 86-592-2388138  
Fax: 86-592-2388130

#### China - Xian

Tel: 86-29-8833-7252  
Fax: 86-29-8833-7256

#### China - Zhuhai

Tel: 86-756-3210040  
Fax: 86-756-3210049

### ASIA/PACIFIC

#### India - Bangalore

Tel: 91-80-3090-4444  
Fax: 91-80-3090-4080

#### India - New Delhi

Tel: 91-11-4160-8631  
Fax: 91-11-4160-8632

#### India - Pune

Tel: 91-20-2566-1512  
Fax: 91-20-2566-1513

#### Japan - Yokohama

Tel: 81-45-471-6166  
Fax: 81-45-471-6122

#### Korea - Daegu

Tel: 82-53-744-4301  
Fax: 82-53-744-4302

#### Korea - Seoul

Tel: 82-2-554-7200  
Fax: 82-2-558-5932 or  
82-2-558-5934

#### Malaysia - Kuala Lumpur

Tel: 60-3-6201-9857  
Fax: 60-3-6201-9859

#### Malaysia - Penang

Tel: 60-4-227-8870  
Fax: 60-4-227-4068

#### Philippines - Manila

Tel: 63-2-634-9065  
Fax: 63-2-634-9069

#### Singapore

Tel: 65-6334-8870  
Fax: 65-6334-8850

#### Taiwan - Hsin Chu

Tel: 886-3-6578-300  
Fax: 886-3-6578-370

#### Taiwan - Kaohsiung

Tel: 886-7-536-4818  
Fax: 886-7-536-4803

#### Taiwan - Taipei

Tel: 886-2-2500-6610  
Fax: 886-2-2508-0102

#### Thailand - Bangkok

Tel: 66-2-694-1351  
Fax: 66-2-694-1350

### EUROPE

#### Austria - Wels

Tel: 43-7242-2244-39  
Fax: 43-7242-2244-393

#### Denmark - Copenhagen

Tel: 45-4450-2828  
Fax: 45-4485-2829

#### France - Paris

Tel: 33-1-69-53-63-20  
Fax: 33-1-69-30-90-79

#### Germany - Munich

Tel: 49-89-627-144-0  
Fax: 49-89-627-144-44

#### Italy - Milan

Tel: 39-0331-742611  
Fax: 39-0331-466781

#### Netherlands - Drunen

Tel: 31-416-690399  
Fax: 31-416-690340

#### Spain - Madrid

Tel: 34-91-708-08-90  
Fax: 34-91-708-08-91

#### UK - Wokingham

Tel: 44-118-921-5869  
Fax: 44-118-921-5820

03/26/09