In-Circuit Serial Programming™
(ICSPTM) 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contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip’s products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, KeeLOG, MPLAB, PIC, PICmicro, PICSTART, PRO MATE and PowerSmart are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

FilterLab, microID, MXDEV, MXLAB, PICMASTER, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Accuron, Application Maestro, dsPIC, dsPICDEM, dsPICDEM.net, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, PICC, PICKit, PICDEM, PICDEM.net, PowerCal, PowerInfo, PowerMate, PowerTool, rPLAB, rPIC, Select Mode, SmartSensor, SmartShunt, SmartTel and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

Serialized Quick Turn Programming (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.

© 2003, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.
# Table of Contents

## SECTION 1
**INTRODUCTION**

- In-Circuit Serial Programming™ (ICSP™) Guide ................................................................. 1-1

## SECTION 2
**TECHNICAL BRIEFS**

- How to Implement ICSP™ Using PIC12C5XX OTP MCUs .................................................... 2-1
- How to Implement ICSP™ Using PIC16CXXX OTP MCUs ..................................................... 2-9
- How to Implement ICSP™ Using PIC17CXXX OTP MCUs ..................................................... 2-15
- How to Implement ICSP™ Using PIC16F8X FLASH MCUs ................................................... 2-21

## SECTION 3
**PROGRAMMING SPECIFICATIONS**

- In-Circuit Serial Programming for PIC12C5XX OTP MCUs .................................................. 3-1
- In-Circuit Serial Programming for PIC12C67X and PIC12CE67X OTP MCUs ......................... 3-15
- In-Circuit Serial Programming for PIC14000 OTP MCUs .................................................... 3-27
- In-Circuit Serial Programming for PIC16C55X OTP MCUs .................................................. 3-39
- Programming Specifications for PIC16C6XX/7XX/9XX OTP MCUs ........................................ 3-51
- In-Circuit Serial Programming for PIC17C7XX OTP MCUs .................................................. 3-75
- In-Circuit Serial Programming for PIC18CXXX OTP MCUs .................................................. 3-101
- PIC16F8X EEPROM Memory Programming Specification .................................................... 3-147
- PIC16F62X EEPROM Memory Programming Specification .................................................. 3-161
- PIC16F87X EEPROM Memory Programming Specification .................................................. 3-181

## SECTION 4
**APPLICATION NOTES**

- In-Circuit Serial Programming™ (ICSP™) of Calibration Parameters
  Using a PICmicro® Microcontroller ................................................................. 4-1
INTRODUCTION

WHAT IS IN-CIRCUIT SERIAL PROGRAMMING (ICSP)?

In-System Programming (ISP) is a technique where a programmable device is programmed after the device is placed in a circuit board.

In-Circuit Serial Programming (ICSP) is an enhanced ISP technique implemented in Microchip’s PICmicro® One-Time-Programmable (OTP) and FLASH RISC microcontrollers (MCU). Use of only two I/O pins to serially input and output data makes ICSP easy to use and less intrusive on the normal operation of the MCU.

Because they can accommodate rapid code changes in a manufacturing line, PICmicro OTP and FLASH MCUs offer tremendous flexibility, reduce development time and manufacturing cycles, and improve time to market.

In-Circuit Serial Programming enhances the flexibility of the PICmicro even further.

This In-Circuit Serial Programming Guide is designed to show you how you can use ICSP to get an edge over your competition. Microchip has helped its customers implement ICSP using PICmicro MCUs since 1992. Contact your local Microchip sales representative today for more information on implementing ICSP in your product.

PICmicro MCUs MAKE IN-CIRCUIT SERIAL PROGRAMMING A CINCH

Unlike many other MCUs, most PICmicro MCUs offer a simple serial programming interface using only two I/O pins (plus power, ground and Vpp). Following very simple guidelines, these pins can be fully utilized as I/O pins during normal operation and programming pins during ICSP.

ICSP can be activated through a simple 5-pin connector and a standard PICmicro programmer supporting Serial Programming mode such as Microchip’s PRO MATE® II.

No other MCU has a simpler and less intrusive Serial Programming mode to facilitate your ICSP needs.

WHAT CAN I DO WITH IN-CIRCUIT SERIAL PROGRAMMING?

ICSP is truly an enabling technology that can be used in a variety of ways including:

• Reduce Cost of Field Upgrades

The cost of upgrading a system’s code can be dramatically reduced using ICSP. With very little effort and planning, a PICmicro OTP- or FLASH-based system can be designed to have code updates in the field.

For PICmicro FLASH devices, the entire code memory can be rewritten with new code. In PICmicro OTP devices, new code segments and parameter tables can be easily added in program memory areas left blank for update purpose. Often, only a portion of the code (such as a key algorithm) requires update.

• Reduce Time to Market

In instances where one product is programmed with different customer codes, generic systems can be built and inventoried ahead of time. Based on actual mix of customer orders, the PICmicro MCU can be programmed using ICSP, then tested and shipped. The lead-time reduction and simplification of finished goods inventory are key benefits.

• Calibrate Your System During Manufacturing

Many systems require calibration in the final stages of manufacturing and testing. Typically, calibration parameters are stored in Serial EEPROM devices. Using PICmicro MCUs, it is possible to save the additional system cost by programming the calibration parameters directly into the program memory.

• Add Unique ID Code to Your System During Manufacturing

Many products require a unique ID number or a serial number. An example application would be a remote keyless entry device. Each transmitter has a unique “binary key” that makes it very easy to program in the access code at the very end of the manufacturing process and prior to final test.

Serial number, revision code, date code, manufacturer ID and a variety of other useful information can also be added to any product for traceability. Using ICSP, you can eliminate the need for DIP switches or jumpers.

In-Circuit Serial Programming and ICSP are trademarks of Microchip Technology Inc. SQTP is a service mark of Microchip Technology Inc.

© 2003 Microchip Technology Inc.
Introduction

In fact, this capability is so important to many of our customers that Microchip offers a factory programming service called Serialized Quick Turn Programming (SQTP™), where each PICmicro MCU device is coded with up to 16 bytes of unique code.

**Calibrate Your System in the Field**
Calibration need not be done only in the factory. During installation of a system, ICSP can be used to further calibrate the system to actual operating environment.
In fact, recalibration can be easily done during periodic servicing and maintenance. In OTP parts, newer calibration data can be written to blank memory locations reserved for such use.

**Customize and Configure Your System in the Field**
Like calibration, customization need not be done in the factory only. In many situations, customizing a product at installation time is very useful. A good example is home or car security systems where ID code, access code and other such information can be burned in after the actual configuration is determined. Additionally, you can save the cost of DIP switches and jumpers, which are traditionally used.

**Program Dice When Using Chip-On-Board (COB)**
If you are using COB, Microchip offers a comprehensive die program. You can get dice that are preprogrammed, or you may want to program the die once the circuit board is assembled. Programming and testing in one single step in the manufacturing process is simpler and more cost effective.

**PROGRAMMING TIME CONSIDERATIONS**

Programming time can be significantly different between OTP and FLASH MCUs. OTP (EPROM) bytes typically program with pulses in the order of several hundred microseconds. FLASH, on the other hand, require several milliseconds or more per byte (or word) to program.

Figure 1 and Figure 2 below illustrate the programming time differences between OTP and FLASH MCUs. Figure 1 shows programming time in an ideal programmer or tester, where the only time spent is actually programming the device. This is only important to illustrate the minimum time required to program such devices, where the programmer or the tester is fully optimized.

Figure 2 is a more realistic programming time comparison, where the “overhead” time for programmer or a tester is built in. The programmer often requires 3 to 5 times the “theoretically” minimum programming time.

**FIGURE 1: PROGRAMMING TIME FOR FLASH AND OTP MCUS (THEORETICAL MINIMUM TIMES)**

![Programming Time Graph]

**Note 1:** The programming times shown here only include the total programming time for all memory. Typically, a programmer will have quite a bit of overhead over this “theoretical minimum” programming time.

**Note 2:** In the PIC16CXX MCU (used here for comparison) each word is 14-bits wide. For the sake of simplicity, each word is viewed as “two bytes”.

DS30277D-page 1-2 © 2003 Microchip Technology Inc.
Introduction

FIGURE 2: PROGRAMMING TIME FOR FLASH AND OTP MCUS
(TYPICAL PROGRAMMING TIMES ON A PROGRAMMER)

![Programming Time Graph]

<table>
<thead>
<tr>
<th>Memory Size (in bytes)</th>
<th>Typical Flash MCU</th>
<th>Microchip OTP MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td></td>
<td></td>
</tr>
<tr>
<td>260</td>
<td></td>
<td></td>
</tr>
<tr>
<td>280</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note 1: The programming times shown are actual programming times on vendor supplied programmers.

2: Microchip OTP programming times are based on PRO MATE II programmer.

Ramifications
The programming time differences between FLASH and OTP MCUs are not particular material for prototyping quantities. However, its impact can be significant in large volume production.

MICROCHIP PROVIDES A COMPLETE SOLUTION FOR ICSP

Products
Microchip offers the broadest line of ICSP-capable MCUs:
- PIC12C5XX OTP, 8-pin Family
- PIC12C67X OTP, 8-pin Family
- PIC12CE67X OTP, 8-pin Family
- PIC16C6XX OTP, Mid-Range Family
- PIC17C7XX OTP High-End Family
- PIC18CXXX OTP, High-End Family
- PIC16F62X FLASH, Mid-Range Family
- PIC16F8X FLASH, Mid-Range Family
- PIC6F8XX FLASH, Mid-Range Family

All together, Microchip currently offers over 40 MCUs capable of ICSP.

Development Tools
Microchip offers a comprehensive set of development tools for ICSP that allow system engineers to quickly prototype, make code changes and get designs out the door faster than ever before.

PRO MATE II Production Programmer – a production quality programmer designed to support the Serial Programming mode in MCUs up to midvolume production. PRO MATE II runs under DOS in a Command Line mode, Microsoft® Windows® 3.1, Windows® 95/98, and Windows NT®. PRO MATE II is also capable of Serialized Quick Turn ProgrammingSM (SQTPSM), where each device can be programmed with up to 16 bytes of unique code.

Microchip offers an ICSP kit that can be used with the Universal Microchip Device Programmer, PRO MATE II. Together these two tools allow you to implement ICSP with minimal effort and use the ICSP capability of Microchip’s PICmicro MCUs.

Technical support
Microchip has been delivering ICSP capable MCUs since 1992. Many of our customers are using ICSP capability in full production. Our field and factory application engineers can help you implement ICSP in your product.
Introduction

NOTES:
Section 2 – Technical Briefs

HOW TO IMPLEMENT ICSP™ USING PIC12C5XX OTP MCUS ........................................................... 2-1
HOW TO IMPLEMENT ICSP™ USING PIC16CXXX OTP MCUS .......................................................... 2-9
HOW TO IMPLEMENT ICSP™ USING PIC17CXXX OTP MCUS ........................................................ 2-15
HOW TO IMPLEMENT ICSP™ USING PIC16F8X FLASH MCUS ....................................................... 2-21
INTRODUCTION
The technical brief describes how to implement in-circuit serial programming™ (ICSP) using the PIC12C5XX OTP PICmicro® MCU.

ICSP is a simple way to manufacture your board with an unprogrammed PICmicro MCU and program the device just before shipping the product. Programming the PIC12C5XX MCU in-circuit has many advantages for developing and manufacturing your product.

- **Reduces inventory of products with old firmware.** With ICSP, the user can manufacture product without programming the PICmicro MCU. The PICmicro MCU will be programmed just before the product is shipped.
- **ICSP in production.** New software revisions or additional software modules can be programmed during production into the PIC12C5XX MCU.
- **ICSP in the field.** Even after your product has been sold, a service man can update your program with new program modules.
- **One hardware with different software.** ICSP allows the user to have one hardware, whereas the PIC12C5XX MCU can be programmed with different types of software.
- **Last minute programming.** Last minute programming can also facilitate quick turnarounds on custom orders for your products.

FIGURE 1: TYPICAL APPLICATION CIRCUIT

ICSp Connector

**IN-CIRCUIT SERIAL PROGRAMMING**

To implement ICSP into an application, the user needs to consider three main components of an ICSP system: Application Circuit, Programmer and Programming Environment.

**Application Circuit**

During the initial design phase of the application circuit, certain considerations have to be taken into account. Figure 1 shows a typical circuit that addresses the details to be considered during design. In order to implement ICSP on your application board you have to put the following issues into consideration:

1. Isolation of the GP3/MCLR/VPP pin from the rest of the circuit.
2. Isolation of pins GP1 and GP0 from the rest of the circuit.
3. Capacitance on each of the Vdd, GP3/MCLR/VP, GP1, and GP0 pins.
4. Interface to the programmer.
5. Minimum and maximum operating voltage for Vdd.

**FIGURE 1: TYPICAL APPLICATION CIRCUIT**

PICmicro, PRO MATE and PICSTART are registered trademarks of Microchip Technology Inc. In-Circuit Serial Programming and ICSP are trademarks of Microchip Technology Inc.
Isolation of the GP3/MCLR/VPP Pin from the Rest of the Circuit

PIC12C5XX devices have two ways of configuring the MCLR pin:

- MCLR can be connected either to an external RC circuit or
- MCLR is tied internally to VDD

When GP3/MCLR/VPP pin is connected to an external RC circuit, the pull-up resistor is tied to VDD, and a capacitor is tied to ground. This circuit can affect the operation of ICSP depending on the size of the capacitor.

Another point of consideration with the GP3/MCLR/VPP pin, is that when the PICmicro MCU is programmed, this pin is driven up to 13V and also to ground. Therefore, the application circuit must be isolated from the voltage coming from the programmer.

When MCLR is tied internally to VDD, the user has only to consider that up to 13V are present during programming of the GP3/MCLR/VPP pin. This might affect other components connected to that pin.

For more information about configuring the GP3/MCLR/VPP pin, is that when the PICmicro MCU is programmed, this pin is driven up to 13V and also to ground. Therefore, the application circuit must be isolated from the voltage coming from the programmer.

Total Capacitance on VDD, GP3/MCLR/VPP, GP1, and GP0

The total capacitance on the programming pins affects the rise rates of these signals as they are driven out of the programmer. Typical circuits use several hundred microfarads of capacitance on VDD, which helps to dampen noise and improve electromagnetic interference. However, this capacitance requires a fairly strong driver in the programmer to meet the rise rate timings for VDD.

Interface to the Programmer

Most programmers are designed to simply program the PICmicro MCU itself and don't have strong enough drivers to power the application circuit.

One solution is to use a driver board between the programmer and the application circuit. The driver board needs a separate power supply that is capable of driving the VPP, VDD, GP1, and GP0 pins with the correct ramp rates and also should provide enough current to power-up the application circuit.

The cable length between the programmer and the circuit is also an important factor for ICSP. If the cable between the programmer and the circuit is too long, signal reflections may occur. These reflections can momentarily cause up to twice the voltage at the end of the cable, that was sent from the programmer. This voltage can cause a latch-up. In this case, a termination resistor has to be used at the end of the signal line.

Minimum and Maximum Operating Voltage for VDD

The PIC12C5XX programming specification states that the device should be programmed at 5V. Special considerations must be made if your application circuit operates at 3V only. These considerations may include totally isolating the PICmicro MCU during programming. The other point of consideration is that the device must be verified at minimum and maximum operating voltage of the circuit in order to ensure proper programming margin.

For example, a battery driven system may operate from three 1.5V cells giving an operating voltage range of 2.7V to 4.5V. The programmer must program the device at 5V and must verify the program memory contents at both 2.7V and 4.5V to ensure that proper programming margins have been achieved.
**THE PROGRAMMER**

PIC12C5XX MCUs only use serial programming and, therefore, all programmers supporting these devices will support the ICSP. One issue with the programmer is the drive capability. As discussed before, it must be able to provide the specified rise rates on the ICSP signals and also provide enough current to power the application circuit. It is recommended that you buffer the programming signals.

Another point of consideration for the programmer is what $V_{DD}$ levels are used to verify the memory contents of the PICmicro MCU. For instance, the PRO MATE II verifies program memory at the minimum and maximum $V_{DD}$ levels for the specified device and is therefore considered a production quality programmer. On the other hand, the PICSTART® Plus only verifies at 5V and is for prototyping use only. The PIC12C5XX programming specifications state that the program memory contents should be verified at both the minimum and maximum $V_{DD}$ levels that the application circuit will be operating. This implies that the application circuit must be able to handle the varying $V_{DD}$ voltages.

There are also several third-party programmers that are available. You should select a programmer based on the features it has and how it fits into your programming environment. The Microchip Development Systems Ordering Guide (DS30177) provides detailed information on all our development tools. The Microchip Third Party Guide (DS00104) provides information on all of our third party development tool developers. Please consult these two references when selecting a programmer. Many options exist including serial or parallel PC host connection, stand-alone operation, and single or gang programmers.

**PROGRAMMING ENVIRONMENT**

The programming environment will affect the type of programmer used, the programmer cable length, and the application circuit interface. Some programmers are well suited for a manual assembly line while others are desirable for an automated assembly line. A gang programmer should be chosen for programming multiple MCUs at one time. The physical distance between the programmer and the application circuit affects the load capacitance on each of the programming signals. This will directly affect the drive strength needed to provide the correct signal rise rates and current. Finally, the application circuit interface to the programmer depends on the size constraints of the application circuit itself and the assembly line. A simple header can be used to interface the application circuit to the programmer. This might be more desirable for a manual assembly line, where a technician plugs the programmer cable into the board.

A different method is the use of spring loaded test pins (often referred to as pogo-pins). The application circuit has pads on the board for each of the programming signals. Then there is a movable fixture that has pogo pins in the same configuration as the pads on the board. The application circuit is moved into position and the fixture is moved such that the spring loaded test pins come into contact with the board. This method might be more suitable for an automated assembly line.

After taking into consideration the issues with the application circuit, the programmer, and the programming environment, anyone can build a high quality, reliable manufacturing line based on ICSP.

**OTHER BENEFITS**

ICSP provides several other benefits such as calibration and serialization. If program memory permits, it would be cheaper and more reliable to store calibration constants in program memory instead of using an external serial EEPROM.

**Field Programming of PICmicro OTP MCUs**

An OTP device is not normally capable of being reprogrammed, but the PICmicro MCU architecture gives you this flexibility provided the size of your firmware is less than half that of the desired device.

This method involves using jump tables for the reset and interrupt vectors. Example 1 shows the location of a main routine and the reset vector for the first time a device with 0.5K-words of program memory is programmed. Example 2 shows the location of a second main routine and its reset vector for the second time the same device is programmed. You will notice that the GOTO Main that was previously at location 0x0002 is replaced with an NOP. An NOP is a program memory location with all the bits programmed as 0s. When the reset vector is executed, it will execute an NOP and then a GOTO Main1 instruction to the new code.
EXAMPLE 1: LOCATION OF THE FIRST MAIN ROUTINE AND ITS INTERRUPT VECTOR

**TABLE: PROGRAM MEMORY**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>MOVWF OSCAL</td>
</tr>
<tr>
<td>0x001</td>
<td>GOTO MAIN1</td>
</tr>
<tr>
<td>0x040</td>
<td>MAIN1</td>
</tr>
<tr>
<td>0x080</td>
<td>UNPROGRAMMED</td>
</tr>
<tr>
<td>0x1FF</td>
<td>MOVWF XX</td>
</tr>
</tbody>
</table>

**Legend:** XX = CALIBRATION VALUE

**Notes:**
- RESET VECTOR
- MAIN1 ROUTINE
- UNPROGRAMMED
- PROGRAM MEMORY
- CALIBRATION VALUE
EXAMPLE 2: LOCATION OF THE SECOND MAIN ROUTINE AND IT INTERRUPT VECTOR (AFTER SECOND PROGRAMMING)

LEGEND: XX = CALIBRATION VALUE
Since the program memory of the PIC12C5XX devices is organized in 256 x 12 word pages, placement of such information as look-up tables and CALL instructions must be taken into account. For further information, please refer to application note AN581, Implementing Long Calls and application note AN556, Implementing a Table Read.

CONCLUSION

Microchip Technology Inc. is committed to supporting your ICSP needs by providing you with our many years of experience and expertise in developing in-circuit system programming solutions. Anyone can create a reliable in-circuit system programming station by coupling our background with some forethought to the circuit design and programmer selection issues previously mentioned. Your local Microchip representative is available to answer any questions you have about the requirements for ICSP.
APPENDIX A: SAMPLE DRIVER BOARD SCHEMATIC

Note: The driver board design MUST be tested in the user's application to determine the effects of the application circuit on the programming signals timing. Changes may be required if the application places a significant load on VDD, VPP, GP0 or GP1.
INTRODUCTION

In-Circuit Serial Programming™ (ICSP) is a great way to reduce your inventory overhead and time-to-market for your product. By assembling your product with a blank Microchip microcontroller (MCU), you can stock one design. When an order has been placed, these units can be programmed with the latest revision of firmware, tested, and shipped in a very short time. This method also reduces scrapped inventory due to old firmware revisions. This type of manufacturing system can also facilitate quick turnarounds on custom orders for your product.

Most people would think to use ICSP with PICmicro® OTP MCUs only on an assembly line where the device is programmed once. However, there is a method by which an OTP device can be programmed several times depending on the size of the firmware. This method, explained later, provides a way to field upgrade your firmware in a way similar to EEPROM- or Flash-based devices.

HOW DOES ICSP WORK?

Now that ICSP appeals to you, what steps do you take to implement it in your application? There are three main components of an ICSP system: Application Circuit, Programmer and Programming Environment.

Application Circuit

The application circuit must be designed to allow all the programming signals to be directly connected to the PICmicro MCU. Figure 1 shows a typical circuit that is a starting point for when designing with ICSP. The application must compensate for the following issues:

1. Isolation of the MCLR/Vpp pin from the rest of the circuit.
2. Isolation of pins RB6 and RB7 from the rest of the circuit.
3. Capacitance on each of the Vdd, MCLR/Vpp, RB6, and RB7 pins.
4. Minimum and maximum operating voltage for Vdd.
5. PICmicro Oscillator.
6. Interface to the programmer.

The MCLR/Vpp pin is normally connected to an RC circuit. The pull-up resistor is tied to Vdd and a capacitor is tied to ground. This circuit can affect the operation of ICSP depending on the size of the capacitor. It is, therefore, recommended that the circuit in Figure 1 be used when an RC is connected to MCLR/Vpp. The diode should be a Schottky-type device. Another issue with MCLR/Vpp is that when the PICmicro MCU device is programmed, this pin is driven to approximately 13V and also to ground. Therefore, the application circuit must be isolated from this voltage provided by the programmer.
Pins RB6 and RB7 are used by the PICmicro MCU for serial programming. RB6 is the clock line and RB7 is the data line. RB6 is driven by the programmer. RB7 is a bidirectional pin that is driven by the programmer when programming, and driven by the PICmicro MCU when verifying. These pins must be isolated from the rest of the application circuit so as not to affect the signals during programming. You must take into consideration the output impedance of the programmer when isolating RB6 and RB7 from the rest of the circuit. This isolation circuit must account for RB6 being an input on the PICmicro MCU, and for RB7 being bidirectional (can be driven by both the PICmicro MCU and the programmer). For instance, PRO MATE® II has an output impedance of 1kΩ. If the design permits, these pins should not be used by the application. This is not the case with most applications so it is recommended that the designer evaluate whether these signals need to be buffered. As a designer, you must consider what type of circuitry is connected to RB6 and RB7 and then make a decision on how to isolate these pins. Figure 1 does not show any circuitry to isolate RB6 and RB7 on the application circuit because this is very application dependent.

The total capacitance on the programming pins affects the rise rates of these signals as they are driven out of the programmer. Typical circuits use several hundred microfarads of capacitance on VDD which helps to dampen noise and ripple. However, this capacitance requires a fairly strong driver in the programmer to meet the rise rate timings for VDD. Most programmers are designed to simply program the PICmicro MCU itself and don’t have strong enough drivers to power the application circuit. One solution is to use a driver board between the programmer and the application circuit. The driver board requires a separate power supply that is capable of driving the Vpp and VDD pins with the correct rise rates and should also provide enough current to power the application circuit. RB6 and RB7 are not buffered on this schematic but may require buffering depending upon the application. A sample driver board schematic is shown in Appendix A.

**Note:** The driver board design MUST be tested in the user's application to determine the effects of the application circuit on the programming signals timing. Changes may be required if the application places a significant load on VDD, Vpp, RB6 OR RB7.

The Microchip programming specification states that the device should be programmed at 5V. Special considerations must be made if your application circuit operates at 3V only. These considerations may include totally isolating the PICmicro MCU during programming. The other issue is that the device must be verified at the minimum and maximum voltages at which the application circuit will be operating. For instance, a battery operated system may operate from three 1.5V cells giving an operating voltage range of 2.7V to 4.5V. The programmer must program the device at 5V and must verify the program memory contents at both 2.7V and 4.5V to ensure that proper programming margins have been achieved. This ensures the PICmicro MCU option over the voltage range of the system.

This final issue deals with the oscillator circuit on the application board. The voltage on MCLR/Vpp must rise to the specified program mode entry voltage before the device executes any code. The crystal modes available on the PICmicro MCU are not affected by this issue because the Oscillator Start-up Timer waits for 1024 oscillations before any code is executed. However, RC oscillators do not require any startup time and, therefore, the Oscillator Startup Timer is not used. The programmer must drive MCLR/Vpp to the program mode entry voltage before the RC oscillator toggles four times. If the RC oscillator toggles four or more times, the program counter will be incremented to some value X. Now when the device enters programming mode, the program counter will not be zero and the programmer will start programming your code at an offset of X. There are several alternatives that can compensate for a slow rise rate on MCLR/Vpp. The first method would be to not populate the R, program the device, and then insert the R. The other method would be to have the programming interface drive the OSC1 pin of the PICmicro MCU to ground while programming. This will prevent any oscillations from occurring during programming.

Now all that is left is how to connect the application circuit to the programmer. This depends a lot on the programming environment and will be discussed in that section.

**Programmer**

The second consideration is the programmer. PIC16CXXX MCUs only use serial programming and therefore all programmers supporting these devices will support ICSP. One issue with the programmer is the drive capability. As discussed before, it must be able to provide the specified rise rates on the ICSP signals and also provide enough current to power the application circuit. Appendix A shows an example driver board. This driver schematic does not show any buffer circuitry for RB6 and RB7. It is recommended that an evaluation be performed to determine if buffering is required. Another issue with the programmer is what VDD levels are used to verify the memory contents of the PICmicro MCU. For instance, the PRO MATE II verifies program memory at the minimum and maximum VDD levels for the specified device and is therefore considered a production quality programmer. On the other hand, the PICSTART® Plus only verifies at 5V and is for prototyping use only. The Microchip programming specifications state that the program memory contents should be verified at both the minimum and maximum VDD levels that the application circuit will be operating. This implies that the application circuit must be able to handle the varying VDD voltages.
There are also several third party programmers that are available. You should select a programmer based on the features it has and how it fits into your programming environment. The Microchip Development Systems Ordering Guide (DS30177) provides detailed information on all our development tools. The Microchip Third Party Guide (DS00104) provides information on all of our third party tool developers. Please consult these two references when selecting a programmer. Many options exist including serial or parallel PC host connection, stand-alone operation, and single or gang programmers. Some of the third party developers include Advanced Transdata Corporation, BP Microsystems, Data I/O, Emulation Technology and Logical Devices.

Programming Environment

The programming environment will affect the type of programmer used, the programmer cable length, and the application circuit interface. Some programmers are well suited for a manual assembly line while others are desirable for an automated assembly line. You may want to choose a gang programmer to program multiple systems at a time.

The physical distance between the programmer and the application circuit affects the load capacitance on each of the programming signals. This will directly affect the drive strength needed to provide the correct signal rise rates and current. This programming cable must also be as short as possible and properly terminated and shielded, or the programming signals may be corrupted by ringing or noise.

Finally, the application circuit interface to the programmer depends on the size constraints of the application circuit itself and the assembly line. A simple header can be used to interface the application circuit to the programmer. This might be more desirable for a manual assembly line where a technician plugs the programmer cable into the board. A different method is the use of spring loaded test pins (commonly referred to as pogo pins). The application circuit has pads on the board for each of the programming signals. Then there is a fixture that has pogo pins in the same configuration as the pads on the board. The application circuit or fixture is moved into position such that the pogo pins come into contact with the board. This method might be more suitable for an automated assembly line.

After taking into consideration the issues with the application circuit, the programmer, and the programming environment, anyone can build a high quality, reliable manufacturing line based on ICSP.

Other Benefits

ICSP provides other benefits, such as calibration and serialization. If program memory permits, it would be cheaper and more reliable to store calibration constants in program memory instead of using an external serial EEPROM. For example, your system has a thermistor which can vary from one system to another. Storing some calibration information in a table format allows the microcontroller to compensate in software for external component tolerances. System cost can be reduced without affecting the required performance of the system by using software calibration techniques. But how does this relate to ICSP? The PICmicro MCU has already been programmed with firmware that performs a calibration cycle. The calibration data is transferred to a calibration fixture. When all calibration data has been transferred, the fixture places the PICmicro MCU in programming mode and programs the PICmicro MCU with the calibration data. Application note AN656, In-Circuit Serial Programming of Calibration Parameters Using a PICmicro Microcontroller, shows exactly how to implement this type of calibration data programming.

The other benefit of ICSP is serialization. Each individual system can be programmed with a unique or random serial number. One such application of a unique serial number would be for security systems. A typical system might use DIP switches to set the serial number. Instead, this number can be burned into program memory, thus reducing the overall system cost and lowering the risk of tampering.

Field Programming of PICmicro OTP MCUs

An OTP device is not normally capable of being reprogrammed, but the PICmicro MCU architecture gives you this flexibility provided the size of your firmware is at least half that of the desired device and the device is not code protected. If your target device does not have enough program memory, Microchip provides a wide spectrum of devices from 0.5K to 8K program memory with the same set of peripheral features that will help meet the criteria.

The PIC16CXXX microcontrollers have two vectors, reset and interrupt, at locations 0x0000 and 0x0004. When the PICmicro MCU encounters a reset or interrupt condition, the code located at one of these two locations in program memory is executed. The first listing of Example 1 shows the code that is first programmed into the PICmicro MCU. The second listing of Example 1 shows the code that is programmed into the PICmicro MCU for the second time.
### EXAMPLE 1: PROGRAMMING CYCLE LISTING FILES

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Assembly</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 2808</td>
<td>goto Main ;Main loop</td>
<td></td>
<td>0000 0000</td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>0001 3FFF</td>
<td>&lt;blank&gt; ;at 0x0008</td>
<td></td>
<td>0001 2860</td>
<td>goto Main ;Main now</td>
<td></td>
</tr>
<tr>
<td>0002 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>0002 3FFF</td>
<td>&lt;blank&gt; ;at 0x0060</td>
<td></td>
</tr>
<tr>
<td>0003 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>0003 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
</tr>
<tr>
<td>0004 2848</td>
<td>goto ISR ;ISR at</td>
<td></td>
<td>0004 0000</td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>0005 3FFF</td>
<td>&lt;blank&gt; ;0x0048</td>
<td></td>
<td>0005 28A8</td>
<td>goto ISR ;ISR now at</td>
<td></td>
</tr>
<tr>
<td>0006 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>0006 3FFF</td>
<td>&lt;blank&gt; ;0x00A8</td>
<td></td>
</tr>
<tr>
<td>0007 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>0007 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
</tr>
<tr>
<td>0008 1683</td>
<td>bsf STATUS,RP0</td>
<td></td>
<td>0008 1683</td>
<td>bsf STATUS,RP0</td>
<td></td>
</tr>
<tr>
<td>0009 3007</td>
<td>movlw 0x07</td>
<td></td>
<td>0009 3007</td>
<td>movlw 0x07</td>
<td></td>
</tr>
<tr>
<td>000A 009F</td>
<td>movwf ADCON1</td>
<td></td>
<td>000A 009F</td>
<td>movwf ADCON1</td>
<td></td>
</tr>
<tr>
<td>000A 009F</td>
<td>movwf ADCON1</td>
<td></td>
<td>000A 009F</td>
<td>movwf ADCON1</td>
<td></td>
</tr>
<tr>
<td>000B 009F</td>
<td>movwf ADCON1</td>
<td></td>
<td>000B 009F</td>
<td>movwf ADCON1</td>
<td></td>
</tr>
<tr>
<td>000C 009F</td>
<td>movwf ADCON1</td>
<td></td>
<td>000C 009F</td>
<td>movwf ADCON1</td>
<td></td>
</tr>
<tr>
<td>000D 009F</td>
<td>movwf ADCON1</td>
<td></td>
<td>000D 009F</td>
<td>movwf ADCON1</td>
<td></td>
</tr>
<tr>
<td>000E 009F</td>
<td>movwf ADCON1</td>
<td></td>
<td>000E 009F</td>
<td>movwf ADCON1</td>
<td></td>
</tr>
<tr>
<td>000F 009F</td>
<td>movwf ADCON1</td>
<td></td>
<td>000F 009F</td>
<td>movwf ADCON1</td>
<td></td>
</tr>
<tr>
<td>0010 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>0010 1683</td>
<td>bsf STATUS,RP0</td>
<td></td>
</tr>
<tr>
<td>0011 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>0011 3005</td>
<td>movlw 0x05</td>
<td></td>
</tr>
<tr>
<td>0012 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>0012 009F</td>
<td>movwf ADCON1</td>
<td></td>
</tr>
<tr>
<td>0013 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>0013 009F</td>
<td>movwf ADCON1</td>
<td></td>
</tr>
<tr>
<td>0014 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>0014 009F</td>
<td>movwf ADCON1</td>
<td></td>
</tr>
<tr>
<td>0015 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>0015 009F</td>
<td>movwf ADCON1</td>
<td></td>
</tr>
<tr>
<td>0016 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>0016 009F</td>
<td>movwf ADCON1</td>
<td></td>
</tr>
<tr>
<td>0017 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>0017 009F</td>
<td>movwf ADCON1</td>
<td></td>
</tr>
<tr>
<td>0018 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>0018 1683</td>
<td>bsf STATUS,RP0</td>
<td></td>
</tr>
<tr>
<td>0019 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>0019 28A8</td>
<td>goto EndISR</td>
<td></td>
</tr>
<tr>
<td>001A 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>001A 1806</td>
<td>btfsc PORTB,0</td>
<td></td>
</tr>
<tr>
<td>001B 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>001B 1806</td>
<td>btfsc PORTB,0</td>
<td></td>
</tr>
<tr>
<td>001C 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>001C 1806</td>
<td>btfsc PORTB,0</td>
<td></td>
</tr>
<tr>
<td>001D 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>001D 1806</td>
<td>btfsc PORTB,0</td>
<td></td>
</tr>
<tr>
<td>001E 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>001E 1806</td>
<td>btfsc PORTB,0</td>
<td></td>
</tr>
<tr>
<td>001F 3FFF</td>
<td>&lt;blank&gt;</td>
<td></td>
<td>001F 1806</td>
<td>btfsc PORTB,0</td>
<td></td>
</tr>
</tbody>
</table>
The example shows that to program the PICmicro MCU a second time the memory location 0x0000, originally goto Main (0x2808), is reprogrammed to all 0’s which happens to be a nop instruction. This location cannot be reprogrammed to the new opcode (0x2860) because the bits that are 0’s cannot be reprogrammed to 1’s, only bits that are 1’s can be reprogrammed to 0’s. The next memory location 0x0001 was originally blank (all 1’s) and now becomes a goto Main (0x2860). When a reset condition occurs, the PICmicro MCU executes the instruction at location 0x0000 which is the nop, a completely benign instruction, and then executes the goto Main to start the execution of code. The example also shows that all program memory locations after 0x005A are blank in the original program so that the second time the PICmicro MCU is programmed, the revised code can be programmed at these locations. The same descriptions can be given for the interrupt vector at location 0x0004.

This method changes slightly for PICmicro MCUs with >2K words of program memory. Each of the goto Main and goto ISR instructions are replaced by the following code segments due to paging on devices with >2K words of program memory.

```
movlw <page>   movlw <page>  
movwf PCLATH movwf PCLATH  
goto Main      goto ISR
```

Now your one time programmable PICmicro MCU is exhibiting more EEPROM- or Flash-like qualities.

CONCLUSION

Microchip Technology Inc. is committed to supporting your ICSP needs by providing you with our many years of experience and expertise in developing ICSP solutions. Anyone can create a reliable ICSP programming station by coupling our background with some forethought to the circuit design and programmer selection issues previously mentioned. Your local Microchip representative is available to answer any questions you have about the requirements for ICSP.
APPENDIX A: SAMPLE DRIVER BOARD SCHEMATIC

Note: The driver board design MUST be tested in the user's application to determine the effects of the application circuit on the programming signals timing. Changes may be required if the application places a significant load on Vdd, Vpp, RB6 or RB7.

*see text in technical brief.*

*see text in technical brief.*
INTRODUCTION

PIC17CXXX microcontroller (MCU) devices can be serially programmed using an RS-232 or equivalent serial interface. As shown in Figure 1, using just three pins, the PIC17CXXX can be connected to an external interface and programmed. In-Circuit Serial Programming (ICSP™) allows for a greater flexibility in an application as well as a faster time to market for the user's product.

This technical brief will demonstrate the practical aspects associated with ICSP using the PIC17CXXX. It will also demonstrate some key capabilities of OTP devices when used in conjunction with ICSP.

Implementation

The PIC17CXXX devices have special instructions, which enables the user to program and read the PIC17CXXX’s program memory. The instructions are TABLWT and TLWT which implement the program memory write operation and TABLRD and TLRD which perform the program memory read operation. For more details, please check the In-Circuit Serial Programming for PIC17CXXX OTP Microcontrollers Specification (DS30273), PIC17C4X data sheet (DS30412) and PIC17C75X data sheet (DS30264).

When doing ICSP, the PIC17CXXX runs a boot code, which configures the USART port and receives data serially through the RX line. This data is then programmed at the address specified in the serial data string. A high voltage (about 13V) is required for the EPROM cell to get programmed, and this is usually supplied by the programming header as shown in Figure 1 and Figure 2. The PIC17CXXX’s boot code enables and disables the high voltage line using a dedicated I/O line.

FIGURE 1: PIC17CXXX IN-CIRCUIT SERIAL PROGRAMMING USING TABLE WRITE INSTRUCTIONS

PRO MATE and PICSTART are registered trademarks and ICSP is a trademark of Microchip Technology Inc.
ICSP Boot Code

The boot code is normally programmed, into the PIC17CXX device using a PRO MATE® or PICSTART® Plus or any third party programmer. As depicted in the flowchart in Figure 4, on power-up, or a reset, the program execution always vectors to the boot code. The boot code is normally located at the bottom of the program memory space e.g. 0x700 for a PIC17C42A (Figure 3).

Several methods could be used to reset the PIC17CXXX when the ICSP header is connected to the system board. The simplest method, as shown in Figure 2, is to derive the system 5V, from the 13V supplied by the ICSP header. It is quite common in manufacturing lines, to have system boards programmed with only the boot code ready and available for testing, calibration or final programming. The ICSP header would thus supply the 13V to the system and this 13V would then be stepped down to supply the 5V required to power the system. Please note that the 13V supply should have enough drive capability to supply power to the system as well as maintain the programming voltage of 13V.

The first action of the boot code (as shown in flowchart Figure 4) is to configure the USART to a known baud rate and transmit a request sequence to the ICSP host system. The host immediately responds with an acknowledgment of this request. The boot code then gets ready to receive ICSP data. The host starts sending the data and address byte sequences to the PIC17CXXX. On receiving the address and data information, the 16-bit address is loaded into the TBLPTR registers and the 16-bit data is loaded into the TABLAT registers. The RA2 pin is driven low to enable 13V at MCLR. The PIC17CXXX device then executes a table write instruction. This instruction in turn causes a long write operation, which disables further code execution. Code execution is resumed when an internal interrupt occurs. This delay ensures that the programming pulse width of 1 ms (max.) is met. Once a location is written, RA2 is driven high to disable further writes and a verify operation is done using the Table read instruction. If the result is good, an acknowledge is sent to the host. This process is repeated till all desired locations are programmed.

In normal operation, when the ICSP header is not connected, the boot code would still execute and the PIC17CXXX would send out a request to the host. However it would not get a response from the host, so it would abort the boot code and start normal code execution.

FIGURE 2: PIC17CXXX IN-CIRCUIT SERIAL PROGRAMMING SCHEMATIC

FIGURE 3: BOOT CODE EXAMPLE FOR PIC17C42A
FIGURE 4: FLOWCHART FOR ICSP BOOT CODE

1. Start
2. Goto Boot Code
3. Configure USART and send request
   - If Received Host's ACK?
     - No: Prepare to receive ICSP data
     - Yes: Time-out complete?
       - Yes: Start Code Execution
       - No: Received Address and Data info?
         - No: Received Address and Data info?
           - Yes: Do Table Write operation
           - No: Interrupt?
             - Yes: Read Program Location
             - No: Program location verified correctly?
               - No: Signal Programming Error
               - Yes: END
         - Yes: Received Address and Data info?
           - No: Received Address and Data info?
             - Yes: Do Table Write operation
             - No: Interrupt?
               - Yes: Read Program Location
               - No: Program location verified correctly?
                 - No: Signal Programming Error
                 - Yes: END
   - Yes: Do Table Write operation
     - No: Interrupt?
       - Yes: Read Program Location
       - No: Program location verified correctly?
         - No: Signal Programming Error
         - Yes: END
USING THE ICSP FEATURE ON PIC17CXXX OTP DEVICES

The ICSP feature is a very powerful tool when used in conjunction with OTP devices.

Saving Calibration Information Using ICSP

One key use of ICSP is to store calibration constants or parameters in program memory. It is quite common to interface a PIC17CXXX device to a sensor. Accurate, pre-calibrated sensors can be used, but they are more expensive and have long lead times. Un-calibrated sensors on the other hand are inexpensive and readily available. The only caveat is that these sensors have to be calibrated in the application. Once the calibration constants have been determined, they would be unique to a given system, so they have to be saved in program memory. These calibration parameters/constants can then be retrieved later during program execution and used to improve the accuracy of low cost un-calibrated sensors. ICSP thus offers a cost reduction path for the end user in the application.

Saving Field Calibration Information Using ICSP

Sensors typically tend to drift and lose calibration over time and usage. One expensive solution would be to replace the sensor with a new one. A more cost effective solution however, is to re-calibrated the system and save the new calibration parameter/constants into the PIC17CXXX devices using ICSP. The user program however has to take into account certain issues:

1. Un-programmed or blank locations have to be reserved at each calibration constant location in order to save new calibration parameters/constants.
2. The old calibration parameters/constants are all programmed to 0, so the user program will have to be "intelligent" and differentiate between blank (0xFFFF), zero (0x0000), and programmed locations.

Figure 5 shows how this can be achieved.

Programming Unique Serial Numbers Using ICSP

There are applications where each system needs to have a unique and sometimes random serial number. Example: security devices. One common solution is to have a set of DIP switches which are then set to a unique value during final test. A more cost effective solution however would be to program unique serial numbers into the device using ICSP. The user application can thus eliminate the need for DIP switches and subsequently reduce the cost of the system.

FIGURE 5: FIELD CALIBRATION USING ICSP

![Field Calibration Using ICSP Diagram]
How to Implement ICSP™ Using PIC17CXXX OTP MCUs

Code Updates in the Field Using ICSP

With fast time to market it is not uncommon to see application programs which need to be updated or corrected for either enhancements or minor errors/bugs. If ROM parts were used, updates would be impossible and the product would either become outdated or recalled from the field. A more cost effective solution is to use OTP devices with ICSP and program them in the field with the new updates. Figure 6 shows an example where the user has allowed for one field update to his program.

Here are some of the issues which need to be addressed:

1. The user has to reserve sufficient blank memory to fit his updated code.
2. At least one blank location needs to be saved at the reset vector as well as for all the interrupts.
3. Program all the old "goto" locations (located at the reset vector and the interrupts vectors) to 0 so that these instructions execute as NOPs.
4. Program new "goto" locations (at the reset vector and the interrupt vectors) just below the old "goto" locations.
5. Finally, program the new updated code in the blank memory space.

CONCLUSION

ICSP is a very powerful feature available on the PIC17CXXX devices. It offers tremendous design flexibility to the end user in terms of saving calibration constants and updating code in final production as well as in the field, thus helping the user design a low-cost and fast time-to-market product.

FIGURE 6: CODE UPDATES USING ICSP

![Flowchart Diagram]

- Production Program
  - Goto Boot: 0x0000
  - Main: Goto Main1
  - Main1: Goto Main2
  - Main2: Goto Main

- Code Update #1
  - Goto Boot: 0x0000
  - Main: Goto Main2
  - Main2: Goto Main
INTRODUCTION

In-Circuit Serial Programming™ (ICSP) with PICmicro® FLASH microcontrollers (MCU) is not only a great way to reduce your inventory overhead and time-to-market for your product, but also to easily provide field upgrades of firmware. By assembling your product with a Microchip FLASH-based MCU, you can stock the shelf with one system. When an order has been placed, these units can be programmed with the latest revision of firmware, tested, and shipped in a very short time. This type of manufacturing system can also facilitate quick turnaround on custom orders for your product. You don’t have to worry about scrapped inventory because of the FLASH-based program memory. This gives you the advantage of upgrading the firmware at any time to fix those “features” that pop up from time to time.

HOW DOES ICSP WORK?

Now that ICSP appeals to you, what steps do you take to implement it in your application? There are three main components of an ICSP system.

These are the: Application Circuit, Programmer and Programming Environment.

Application Circuit

The application circuit must be designed to allow all the programming signals to be directly connected to the PICmicro MCUs. Figure 1 shows a typical circuit that is a starting point for when designing with ICSP. The application must compensate for the following issues:

1. Isolation of the MCLR/VPP pin from the rest of the circuit.
2. Isolation of pins RB6 and RB7 from the rest of the circuit.
3. Capacitance on each of the Vdd, MCLR/VPP, RB6, and RB7 pins.
4. Minimum and maximum operating voltage for Vdd.
5. PICmicro Oscillator.
6. Interface to the programmer.

The MCLR/VPP pin is normally connected to an RC circuit. The pull-up resistor is tied to Vdd and a capacitor is tied to ground. This circuit can affect the operation of ICSP depending on the size of the capacitor. It is, therefore, recommended that the circuit in Figure 1 be used when an RC is connected to MCLR/VPP. The diode should be a Schottky-type device. Another issue with MCLR/VPP is that when the PICmicro MCU device is programmed, this pin is driven to approximately 13V and also to ground. Therefore, the application circuit must be isolated from this voltage provided by the programmer.

FIGURE 1: TYPICAL APPLICATION CIRCUIT
Pins RB6 and RB7 are used by the PICmicro MCU for serial programming. RB6 is the clock line and RB7 is the data line. RB6 is driven by the programmer. RB7 is a bidirectional pin that is driven by the programmer when programming, and driven by the PICmicro MCU when verifying. These pins must be isolated from the rest of the application circuit so as not to affect the signals during programming. You must take into consideration the output impedance of the programmer when isolating RB6 and RB7 from the rest of the circuit. This isolation circuit must account for RB6 being an input on the PICmicro MCU and for RB7 being bidirectional (can be driven by both the PICmicro MCU and the programmer). For instance, PRO MATE® II has an output impedance of 1kΩ. If the design permits, these pins should not be used by the application. This is not the case with most applications so it is recommended that the designer evaluate whether these signals need to be buffered. As a designer, you must consider what type of circuitry is connected to RB6 and RB7 and then make a decision on how to isolate these pins. Figure 1 does not show any circuitry to isolate RB6 and RB7 on the application circuit because this is very application dependent.

The total capacitance on the programming pins affects the rise rates of these signals as they are driven out of the programmer. Typical circuits use several hundred microfarads of capacitance on Vdd which helps to dampen noise and ripple. However, this capacitance requires a fairly strong driver in the programmer to meet the rise rate timings for Vdd. Most programmers are designed to simply program the PICmicro MCU itself and don’t have strong enough drivers to power the application circuit. One solution is to use a driver board between the programmer and the application circuit. The driver board requires a separate power supply that is capable of driving the VPP and VDD pins with the correct rise rates and should also provide enough current to power the application circuit. RB6 and RB7 are not buffered on this schematic but may require buffering depending upon the application. A sample driver board schematic is shown in Appendix A.

**Note:** The driver board design MUST be tested in the user’s application to determine the effects of the application circuit on the programming signals timing. Changes may be required if the application places a significant load on Vdd, VPP, RB6 or RB7.

The Microchip programming specification states that the device should be programmed at 5V. Special considerations must be made if your application circuit operates at 3V only. These considerations may include totally isolating the PICmicro MCU during programming. The other issue is that the device must be verified at the minimum and maximum voltages at which the application circuit will be operating. For instance, a battery operated system may operate from three 1.5V cells giving an operating voltage range of 2.7V to 4.5V. The programmer must program the device at 5V and must verify the program memory contents at both 2.7V and 4.5V to ensure that proper programming margins have been achieved. This ensures the PICmicro MCU option over the voltage range of the system.

This final issue deals with the oscillator circuit on the application board. The voltage on MCLR/VPP must rise to the specified program mode entry voltage before the device executes any code. The crystal modes available on the PICmicro MCU are not affected by this issue because the Oscillator Start-up Timer waits for 1024 oscillations before any code is executed. However, RC oscillators do not require any startup time and, therefore, the Oscillator Startup Timer is not used. The programmer must drive MCLR/VPP to the program mode entry voltage before the RC oscillator toggles four times. If the RC oscillator toggles four or more times, the program counter will be incremented to some value X. Now when the device enters programming mode, the program counter will not be zero and the programmer will start programming your code at an offset of X. There are several alternatives that can compensate for a slow rise rate on MCLR/VPP. The first method would be to not populate the R, program the device, and then insert the R. The other method would be to have the programming interface drive the OSC1 pin of the PICmicro MCU to ground while programming. This will prevent any oscillations from occurring during programming.

Now all that is left is how to connect the application circuit to the programmer. This depends a lot on the programming environment and will be discussed in that section.

**Programmer**

The second consideration is the programmer. PIC16F8X MCUs only use serial programming and therefore all programmers supporting these devices will support ICSP. One issue with the programmer is the drive capability. As discussed before, it must be able to provide the specified rise rates on the ICSP signals and also provide enough current to power the application circuit. Appendix A shows an example driver board. This driver schematic does not show any buffer circuitry for RB6 and RB7. It is recommended that an evaluation be performed to determine if buffering is required. Another issue with the programmer is what Vdd levels are used to verify the memory contents of the PICmicro MCU. For instance, the PRO MATE II verifies program memory at the minimum and maximum Vdd levels for the specified device and is therefore considered a production quality programmer. On the other hand, the PICSTART® Plus only verifies at 5V and is for prototyping use only. The Microchip programming specifications state that the program memory contents should be verified at both the minimum and maximum Vdd levels that the application circuit will be operating. This implies that the application circuit must be able to handle the varying Vdd voltages.
There are also several third party programmers that are available. You should select a programmer based on the features it has and how it fits into your programming environment. The Microchip Development Systems Ordering Guide (DS30177) provides detailed information on all our development tools. The Microchip Third Party Guide (DS00104) provides information on all of our third party tool developers. Please consult these two references when selecting a programmer. Many options exist including serial or parallel PC host connection, stand-alone operation, and single or gang programmers. Some of the third party developers include Advanced Transdata Corporation, BP Microsystems, Data I/O, Emulation Technology and Logical Devices.

Programming Environment

The programming environment will affect the type of programmer used, the programmer cable length, and the application circuit interface. Some programmers are well suited for a manual assembly line while others are desirable for an automated assembly line. You may want to choose a gang programmer to program multiple systems at a time.

The physical distance between the programmer and the application circuit affects the load capacitance on each of the programming signals. This will directly affect the drive strength needed to provide the correct signal rise rates and current. This programming cable must also be as short as possible and properly terminated and shielded or the programming signals may be corrupted by ringing or noise.

Finally, the application circuit interface to the programmer depends on the size constraints of the application circuit itself and the assembly line. A simple header can be used to interface the application circuit to the programmer. This might be more desirable for a manual assembly line where a technician plugs the programmer cable into the board. A different method is the use of spring loaded test pins (commonly referred to as pogo pins). The application circuit has pads on the board for each of the programming signals. Then there is a fixture that has pogo pins in the same configuration as the pads on the board. The application circuit or fixture is moved into position such that the pogo pins come into contact with the board. This method might be more suitable for an automated assembly line.

After taking into consideration the issues with the application circuit, the programmer, and the programming environment, anyone can build a high quality, reliable manufacturing line based on ICSP.

Other Benefits

ICSP provides other benefits, such as calibration and serialization. If program memory permits, it would be cheaper and more reliable to store calibration constants in program memory instead of using an external serial EEPROM. For example, your system has a thermistor which can vary from one system to another. Storing some calibration information in a table format allows the microcontroller to compensate in software for external component tolerances. System cost can be reduced without affecting the required performance of the system by using software calibration techniques. But how does this relate to ICSP? The PICmicro MCU has already been programmed with firmware that performs a calibration cycle. The calibration data is transferred to a calibration fixture. When all calibration data has been transferred, the fixture places the PICmicro MCU in programming mode and programs the PICmicro MCU with the calibration data. Application note AN656, In-Circuit Serial Programming of Calibration Parameters Using a PICmicro Microcontroller, shows exactly how to implement this type of calibration data programming.

The other benefit of ICSP is serialization. Each individual system can be programmed with a unique or random serial number. One such application of a unique serial number would be for security systems. A typical system might use DIP switches to set the serial number. Instead, this number can be burned into program memory thus reducing the overall system cost and lowering the risk of tampering.

Field Programming of FLASH PICmicro MCUs

With the ISP interface circuitry already in place, these FLASH-based PICmicro MCUs can be easily reprogrammed in the field. These FLASH devices allow you to reprogram them even if they are code protected. A portable ISP programming station might consist of a laptop computer and programmer. The technician plugs the ISP interface cable into the application circuit and downloads the new firmware into the PICmicro MCU. The next thing you know the system is up and running without those annoying “bugs”. Another instance would be that you want to add an additional feature to your system. All of your current inventory can be converted to the new firmware and field upgrades can be performed to bring your installed base of systems up to the latest revision of firmware.

CONCLUSION

Microchip Technology Inc. is committed to supporting your ICSP needs by providing you with our many years of experience and expertise in developing ICSP solutions. Anyone can create a reliable ICSP programming station by coupling our background with some forethought to the circuit design and programmer selection issues previously mentioned. Your local Microchip representative is available to answer any questions you have about the requirements for ICSP.
APPENDIX A: SAMPLE DRIVER BOARD SCHEMATIC

Note:
The driver board design MUST be tested in the user's application to determine the effects of the application circuit on the programming signals timing. Changes may be required if the application places a significant load on Vdd, Vpp, RB6 or RB7.

* see text in technical brief.* see text in technical brief.
Section 3 – Programming Specifications

IN-CIRCUIT SERIAL PROGRAMMING FOR PIC12C5XX OTP MCUs .................................................. 3-1
IN-CIRCUIT SERIAL PROGRAMMING FOR PIC12C67X AND PIC12CE67X OTP MCUs ...................... 3-15
IN-CIRCUIT SERIAL PROGRAMMING FOR PIC14000 OTP MCUs ................................................... 3-27
IN-CIRCUIT SERIAL PROGRAMMING FOR PIC16C55X OTP MCUs .................................................. 3-39
IN-CIRCUIT SERIAL PROGRAMMING FOR PIC16C6XX/7XX/9XX OTP MCUs ............................... 3-51
IN-CIRCUIT SERIAL PROGRAMMING FOR PIC17C7XX OTP MCUs .................................................. 3-71
IN-CIRCUIT SERIAL PROGRAMMING FOR PIC18CXXX OTP MCUs .................................................. 3-97
IN-CIRCUIT SERIAL PROGRAMMING FOR PIC16F62X FLASH MCUs .......................................... 3-135
IN-CIRCUIT SERIAL PROGRAMMING FOR PIC16F8X FLASH MCUs ............................................ 3-149
IN-CIRCUIT SERIAL PROGRAMMING FOR PIC16F8XX FLASH MCUs .......................................... 3-165
1.0 PROGRAMMING THE PIC12C5XX

The PIC12C5XX can be programmed using a serial method. Due to this serial programming, the PIC12C5XX can be programmed while in the user's system, increasing design flexibility. This programming specification applies to PIC12C5XX devices in all packages.

1.1 Hardware Requirements

The PIC12C5XX requires two programmable power supplies, one for \( V_{DD} \) (2.0V to 6.5V recommended) and one for \( V_{PP} \) (12V to 14V). Both supplies should have a minimum resolution of 0.25V.

1.2 Programming Mode

The Programming mode for the PIC12C5XX allows programming of user program memory, special locations used for ID, and the configuration word for the PIC12C5XX.
2.0 PROGRAM MODE ENTRY

The Program/Verify Test mode is entered by holding pins DB0 and DB1 low, while raising MCLR pin from VIL to VIHH. Once in this Test mode, the user program memory and the test program memory can be accessed and programmed in a serial fashion. The first selected memory location is the fuses. GP0 and GP1 are Schmitt Trigger inputs in this mode.

Incrementing the PC once (using the increment address command), selects location 0x000 of the regular program memory. Afterwards, all other memory locations from 0x001-01FF (PIC12C508/CE518), 0x001-03FF (PIC12C509/CE519) can be addressed by incrementing the PC.

If the program counter has reached the last user program location and is incremented again, the on-chip special EPROM area will be addressed. (See Figure 2-2 to determine where the special EPROM area is located for the various PIC12C5XX devices.)

2.1 Programming Method

The programming technique is described in the following section. It is designed to ensure good programming margins. It does, however, require a variable power supply for VCC.

2.1.1 PROGRAMMING METHOD DETAILS

Essentially, this technique includes the following steps:

1. Perform blank check at VDD = VDDMIN. Report failure. The device may not be properly erased.
2. Program location with pulses and verify after each pulse at VDD = VDDP:
   where VDDP = VDD range required during programming (4.5V - 5.5V).
   a) Programming condition:
      \[ VPP = 13.0V \text{ to } 13.25V \]
      \[ VDD = VDDP = 4.5V \text{ to } 5.5V \]
      \[ VPP \text{ must be } \geq VDD + 7.25V \text{ to keep } \text{“Programming mode” active.} \]
   b) Verify condition:
      \[ VDD = VDDP \]
      \[ VPP \geq VDD + 7.5V \text{ but not to exceed } 13.25V \]
      If location fails to program after \( N \) pulses (suggested maximum program pulses of 8), then report error as a programming failure.

3. Once location passes “Step 2”, apply 11X over programming (i.e., apply 11 times the number of pulses that were required to program the location). This will insure a solid programming margin. The over programming should be made “software programmable” for easy updates.
4. Program all locations.
5. Verify all locations (using Speed Verify mode) at VDD = VDDMIN.
6. Verify all locations at VDD = VDDMAX. VDDMIN is the minimum operating voltage spec. for the part. VDDMAX is the maximum operating voltage spec. for the part.

2.1.2 SYSTEM REQUIREMENTS

Clearly, to implement this technique, the most stringent requirements will be that of the power supplies:

VPP: VPP can be a fixed 13.0V to 13.25V supply. It must not exceed 14.0V to avoid damage to the pin and should be current limited to approximately 100 mA.

VDD: 2.0V to 6.5V with 0.25V granularity. Since this method calls for verification at different VDD values, a programmable VDD power supply is needed.

Current Requirement: 40 mA maximum

Microchip may release devices in the future with different VDD ranges, which make it necessary to have a programmable VDD.

It is important to verify an EPROM at the voltages specified in this method to remain consistent with Microchip’s test screening. For example, a PIC12C5XX specified for 4.5V to 5.5V should be tested for proper programming from 4.5V to 5.5V.

Note: Any programmer not meeting the programmable VDD requirement and the verify at VDDMAX and VDDMIN requirement, may only be classified as a “prototype” or “development” programmer, but not a production programmer.

2.1.3 SOFTWARE REQUIREMENTS

Certain parameters should be programmable (and therefore, easily modified) for easy upgrade.

a) Pulse width.

b) Maximum number of pulses, present limit 8.

c) Number of over-programming pulses: should be \( A = 11, B = 0 \).

2.2 Programming Pulse Width

Program Memory Cells: When programming one word of EPROM, a programming pulse width (TPW) of 100 \( \mu \text{s} \) is recommended.

The maximum number of programming attempts should be limited to 8 per word.

After the first successful verify, the same location should be over-programmed with 11X over-programming.

Configuration Word: The configuration word for oscillator selection, WDT (Watchdog Timer) disable and code protection, and MCLR enable, requires a programming pulse width (TPWF) of 10 ms. A series of 100 \( \mu \text{s} \) pulses is preferred over a single 10 ms pulse.
FIGURE 2-1: PROGRAMMING METHOD FLOW CHART

Start

Blank Check @ VDD = VDDMIN

Pass?

Yes

Report Possible Erase Failure
Continue Programming at user’s option

No

Report Programming Failure

Program 1 Location @ VPP = 13.0V to 13.25V
VDD = VDDP

Pass?

Yes

Apply 11N additional program pulses

No

N > 8?

Yes

N = N + 1

(N = # of program pulses)

No

Increment PC to point to next location, N = 0

All locations done?

Yes

Verify all locations @ VDD = VDDMIN

Pass?

Yes

Verify all locations @ VDD = VDDMAX

Pass?

Yes

Now program Configuration Word

Done

No

Report verify failure @ VDDMIN

No

Report verify failure @ VDDMAX

No

Verify Configuration Word @ VDDMAX & VDDMIN
FIGURE 2-2: PIC12C5XX SERIES PROGRAM MEMORY MAP IN PROGRAM/VERIFY MODE

<table>
<thead>
<tr>
<th>Address (HEX)</th>
<th>Bit Number</th>
<th>11</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNN</td>
<td>User Program Memory (NNN + 1) x 12 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TTT</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TTT + 1</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TTT + 2</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TTT + 3</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TTT + 3F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(FFF)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For Customer Use
(4 x 4 bit usable)

For Factory Use

Configuration Word 5 bits

NNN Highest normal EPROM memory address. NNN = 0x1FF for PIC12C508/CE518.
NNN = 0x3FF for PIC12C509/CE519.
Note that some versions will have an oscillator calibration value programmed at NNN.

TTT Start address of special EPROM area and ID locations.
2.3 Special Memory Locations

The highest address of program memory space is reserved for the internal RC oscillator calibration value. This location should not be overwritten except when this location is blank, and it should be verified, when programmed, that it is a MOV LW XX instruction.

The ID Locations area is only enabled if the device is in Programming/Verify mode. Thus, in normal operation mode, only the memory location 0x000 to 0xFFFF will be accessed and the Program Counter will just rollover from address 0xFFFF to 0x000 when incremented.

The configuration word can only be accessed immediately after MCLR going from VIL to VHH. The Program Counter will be set to all ‘1’s upon MCLR = VIL. Thus, it has the value “0xFFF” when accessing the configuration EPROM. Incrementing the Program Counter once causes the Program Counter to rollover to all ‘0’s. Incrementing the Program Counter 4K times after RESET (MCLR = VIL) does not allow access to the configuration EPROM.

2.3.1 CUSTOMER ID CODE LOCATIONS

Per definition, the first four words (address TTT to TTT + 3) are reserved for customer use. It is recommended that the customer use only the four lower order bits (bits 0 through 3) of each word and filling the eight higher order bits with ‘0’s.

A user may want to store an identification code (ID) in the ID locations and still be able to read this code after the code protection bit was programmed.

EXAMPLE 2-1: CUSTOMER CODE 0xD1E2

The Customer ID code “0xD1E2” should be stored in the ID locations 0x200-0x203 like this (PIC12C508/509A/CE518):

<table>
<thead>
<tr>
<th>Memory Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>0000 0000 1101</td>
</tr>
<tr>
<td>201</td>
<td>0000 0000 0001</td>
</tr>
<tr>
<td>202</td>
<td>0000 0000 1110</td>
</tr>
<tr>
<td>203</td>
<td>0000 0000 0010</td>
</tr>
</tbody>
</table>

Reading these four memory locations, even with the code protection bit programmed, would still output on GP0 the bit sequence “1101”, “0001”, “1110”, “0010” which is “0xD1E2”.

Note: All other locations in PICmicro® MCU configuration memory are reserved and should not be programmed.

2.4 Program/Verify Mode

The Program/Verify mode is entered by holding pins GP1 and GP0 low, while raising MCLR pin from VIL to VHH (high voltage). Once in this mode, the user program memory and the configuration memory can be accessed and programmed in serial fashion. The mode of operation is serial. GP0 and GP1 are Schmitt Trigger inputs in this mode.

The sequence that enters the device into the Programming/Verify mode places all other logic into the RESET state (the MCLR pin was initially at VIL). This means that all I/O are in the RESET state (High impedance inputs).

Note: The MCLR pin should be raised from VIL to VHH within 9 ms of VDD rise. This is to ensure that the device does not have the PC incremented while in valid operation range.
2.4.1 PROGRAM/VERIFY OPERATION

The GP1 pin is used as a clock input pin, and the GP0 pin is used for entering command bits and data input/output during serial operation. To input a command, the clock pin (GP1) is cycled six times. Each command bit is latched on the falling edge of the clock with the Least Significant bit (LSb) of the command being input first. The data on pin GP0 is required to have a minimum setup and hold time (see AC/DC specs), with respect to the falling edge of the clock. Commands that have data associated with them (read and load) are specified to have a minimum delay of 1 \( \mu s \) between the command and the data. After this delay, the clock pin is cycled 16 times with the first cycle being a START bit and the last cycle being a STOP bit. Data is also input and output LSb first. Therefore, during a read operation, the LSb will be transmitted onto pin GP0 on the rising edge of the second cycle, and during a load operation, the LSb will be latched on the falling edge of the second cycle. A minimum 1 \( \mu s \) delay is also specified between consecutive commands.

All commands are transmitted LSb first. Data words are also transmitted LSb first. The data is transmitted on the rising edge and latched on the falling edge of the clock. To allow for decoding of commands and reversal of data pin configuration, a time separation of at least 1 \( \mu s \) is required between a command and a data word (or another command).

The commands that are available are listed in Table 2-1.

### TABLE 2-1: COMMAND MAPPING

<table>
<thead>
<tr>
<th>Command</th>
<th>Mapping (MSb ... LSb)</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Data</td>
<td>0 0 0 0 1 0</td>
<td>0, data(14), 0</td>
</tr>
<tr>
<td>Read Data</td>
<td>0 0 0 1 0 0</td>
<td>0, data(14), 0</td>
</tr>
<tr>
<td>Increment Address</td>
<td>0 0 0 1 1 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Begin programming</td>
<td>0 0 1 0 0 0</td>
<td>0 0 1 1 0</td>
</tr>
<tr>
<td>End Programming</td>
<td>0 0 1 1 1 0</td>
<td>0 0 1 1 0</td>
</tr>
</tbody>
</table>

**Note:** The clock must be disabled during in-circuit programming.
2.4.1.1 Load Data
After receiving this command, the chip will load in a 14-bit "data word" when 16 cycles are applied, as described previously. Because this is a 12-bit core, the two MSb's of the data word are ignored. A timing diagram for the load data command is shown in Figure 5-1.

2.4.1.2 Read Data
After receiving this command, the chip will transmit data bits out of the memory currently accessed, starting with the second rising edge of the clock input. The GP0 pin will go into Output mode on the second rising clock edge, and it will revert back to Input mode (hi-impedance) after the 16th rising edge. Because this is a 12-bit core, the two MSb's of the data are unused and read as '0'. A timing diagram of this command is shown in Figure 5-2.

2.4.1.3 Increment Address
The PC is incremented when this command is received. A timing diagram of this command is shown in Figure 5-3.

2.4.1.4 Begin Programming
A load data command must be given before every begin programming command. Programming of the appropriate memory (test program memory or user program memory) will begin after this command is received and decoded. Programming should be performed with a series of 100 µs programming pulses. A programming pulse is defined as the time between the begin programming command and the end programming command.

2.4.1.5 End Programming
After receiving this command, the chip stops programming the memory (configuration program memory or user program memory) that it was programming at the time.

2.5 Programming Algorithm Requires Variable VDD
The PIC12C5XX uses an intelligent algorithm. The algorithm calls for program verification at VDDMIN, as well as VDDMAX. Verification at VDDMIN guarantees good "erase margin". Verification at VDDMAX guarantees good "program margin".

The actual programming must be done with VDD in the VDDP range (4.75 - 5.25V). VDDP = VCC range required during programming. VDDMIN = minimum operating VDD spec for the part. VDDMAX = maximum operating VDD spec for the part. Programmers must verify the PIC12C5XX at its specified VDDMAX and VDDMIN levels. Since Microchip may introduce future versions of the PIC12C5XX with a broader VDD range, it is best that these levels are user selectable (defaults are ok).

Note: Any programmer not meeting these requirements may only be classified as a "prototype" or "development" programmer, but not a "production" quality programmer.
3.0 CONFIGURATION WORD

The PIC12C5XX family members have several configuration bits. These bits can be programmed (reads '0'), or left unprogrammed (reads '1'), to select various device configurations. Figure 3-1 provides an overview of configuration bits.

**FIGURE 3-1: CONFIGURATION WORD BIT MAP**

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC12C5XX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MCLRE</td>
<td>CP</td>
<td>WDTE</td>
<td>FOSC1</td>
<td>FOSC0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- bit 11-5: **Reserved**: Write as '0' for PIC12C5XX
- bit 4: **MCLRE**: Master Clear Enable bit
  - 1 = MCLR pin enabled
  - 0 = MCLR internally connected to VDD
- bit 3: **CP**: Code Protect Enable bit
  - 1 = Code memory unprotected
  - 0 = Code memory protected
- bit 2: **WDTE**, WDT Enable bit
  - 1 = WDT enabled
  - 0 = WDT disabled
- bit 1-0: **FOSC<1:0>**, Oscillator Selection Bit
  - 11 = External RC oscillator
  - 10 = Internal RC oscillator
  - 01 = XT oscillator
  - 00 = LP oscillator
4.0 CODE PROTECTION

The program code written into the EPROM can be protected by writing to the CP bit of the configuration word.

In PIC12C5XX, it is still possible to program and read locations 0x000 through 0x03F, after code protection. Once code protection is enabled, all protected segments read ‘0’s (or “garbage values”) and are prevented from further programming. All unprotected segments, including ID locations and configuration word, read normally. These locations can be programmed.

Once code protection is enabled, all code protected locations read 0’s. All unprotected segments, including the internal oscillator calibration value, ID, and configuration word read as normal.

4.1 Embedding Configuration Word and ID Information in the HEX File

To allow portability of code, the programmer is required to read the configuration word and ID locations from the HEX file when loading the HEX file. If configuration word information was not present in the HEX file, then a simple warning message may be issued. Similarly, while saving a HEX file, configuration word and ID information must be included. An option to not include this information may be provided.

Microchip Technology Inc. feels strongly that this feature is important for the benefit of the end customer.

<table>
<thead>
<tr>
<th>TABLE 4-1: CODE PROTECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC12C508</td>
</tr>
<tr>
<td>To code protect:</td>
</tr>
<tr>
<td>• (CP enable pattern: Xxxxxx0xxx)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0xFFF)</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>[0x00:0x3F]</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>[0x40:0x1FF]</td>
<td>Read Disabled (all 0’s), Write Disabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>ID Locations (0x200 : 0x203)</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
</tbody>
</table>

| PIC12C508A                |
| To code protect:         |
| • (CP enable pattern: Xxxxxx0xxx) |

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0xFFF)</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>[0x00:0x3F]</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>[0x40:0x1FE]</td>
<td>Read Disabled (all 0’s), Write Disabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>0x1FF Oscillator Calibration Value</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>ID Locations (0x200 : 0x203)</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
</tbody>
</table>

| PIC12C509                |
| To code protect:         |
| • (CP enable pattern: Xxxxxx0xxx) |

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0xFFF)</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>[0x00:0x3F]</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>[0x40:0x3FF]</td>
<td>Read Disabled (all 0’s), Write Disabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>ID Locations (0x400 : 0x403)</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
</tbody>
</table>
**PIC12C509A**

**To code protect:**
- (CP enable pattern: XXXXXXXX0XXX)

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0xFF)</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>[0x00:0x3F]</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>[0x40:0x3FE]</td>
<td>Read Disabled (all 0’s), Write Disabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>0x3FF Oscillator Calibration Value</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>ID Locations (0x400 : 0x403)</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
</tbody>
</table>

**PIC12CE518**

**To code protect:**
- (CP enable pattern: XXXXXXXX0XXX)

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0xFF)</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>[0x00:0x3F]</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>[0x40:0x1FE]</td>
<td>Read Disabled (all 0’s), Write Disabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>0x1FF Oscillator Calibration Value</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>ID Locations (0x200 : 0x203)</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
</tbody>
</table>

**PIC12CE519**

**To code protect:**
- (CP enable pattern: XXXXXXXX0XXX)

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0xFF)</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>[0x00:0x3F]</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>[0x40:0x3FF]</td>
<td>Read Disabled (all 0’s), Write Disabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
<tr>
<td>ID Locations (0x400 : 0x403)</td>
<td>Read Enabled, Write Enabled</td>
<td>Read Enabled, Write Enabled</td>
</tr>
</tbody>
</table>
4.2 Checksum

4.2.1 CHECKSUM CALCULATIONS

Checksum is calculated by reading the contents of the PIC12C5XX memory locations and adding up the opcodes up to the maximum user addressable location (not including the last location which is reserved for the oscillator calibration value), e.g., 0x1FE for the PIC12C508/CE518. Any carry bits exceeding 16 bits are neglected. Finally, the configuration word (appropriately masked) is added to the checksum. Checksum computation for each member of the PIC12C5XX family is shown in Table 4-2.

The checksum is calculated by summing the following:

- The contents of all program memory locations
- The configuration word, appropriately masked
- Masked ID locations (when applicable)

The Least Significant 16 bits of this sum are the checksum.

The following table describes how to calculate the checksum for each device. Note that the checksum calculation differs depending on the code protect setting. Since the program memory locations read out differently depending on the code protect setting, the table describes how to manipulate the actual program memory values to simulate the values that would be read from a protected device. When calculating a checksum by reading a device, the entire program memory can simply be read and summed. The configuration word and ID locations can always be read.

The oscillator calibration value location is not used in the above checksums.

**TABLE 4-2: CHECKSUM COMPUTATION**

<table>
<thead>
<tr>
<th>Device</th>
<th>Code Protect</th>
<th>Checksum*</th>
<th>Blank Value</th>
<th>0x723 at 0 and Max Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC12C508</td>
<td>OFF</td>
<td>SUM[0x000:0x1FE] + CFGW &amp; 0x01F</td>
<td>EE20</td>
<td>D368</td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>SUM[0x000:0x03F] + CFGW &amp; 0x01F + SUM(IDS)</td>
<td>EDF7</td>
<td>D363</td>
</tr>
<tr>
<td>PIC12C508A</td>
<td>OFF</td>
<td>SUM[0x000:0x1FE] + CFGW &amp; 0x01F</td>
<td>EE20</td>
<td>D368</td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>SUM[0x000:0x03F] + CFGW &amp; 0x01F + SUM(IDS)</td>
<td>EDF7</td>
<td>D363</td>
</tr>
<tr>
<td>PIC12C509</td>
<td>OFF</td>
<td>SUM[0x000:0x3FE] + CFGW &amp; 0x01F</td>
<td>EC20</td>
<td>DA68</td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>SUM[0x000:0x03F] + CFGW &amp; 0x01F + SUM(IDS)</td>
<td>EBF7</td>
<td>D163</td>
</tr>
<tr>
<td>PIC12C509A</td>
<td>OFF</td>
<td>SUM[0x000:0x3FE] + CFGW &amp; 0x01F</td>
<td>EC20</td>
<td>DA68</td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>SUM[0x000:0x03F] + CFGW &amp; 0x01F + SUM(IDS)</td>
<td>EBF7</td>
<td>D163</td>
</tr>
<tr>
<td>PIC12CE518</td>
<td>OFF</td>
<td>SUM[0x000:0x3FE] + CFGW &amp; 0x01F</td>
<td>EC20</td>
<td>DA68</td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>SUM[0x000:0x03F] + CFGW &amp; 0x01F + SUM(IDS)</td>
<td>EBF7</td>
<td>D163</td>
</tr>
<tr>
<td>PIC12CE519</td>
<td>OFF</td>
<td>SUM[0x000:0x3FE] + CFGW &amp; 0x01F</td>
<td>EC20</td>
<td>DA68</td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>SUM[0x000:0x03F] + CFGW &amp; 0x01F + SUM(IDS)</td>
<td>EBF7</td>
<td>D163</td>
</tr>
<tr>
<td>rPIC12C509A</td>
<td>OFF</td>
<td>SUM[0x000:0x3FE] + CFGW &amp; 0x01F</td>
<td>EC20</td>
<td>DA68</td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>SUM[0x000:0x03F] + CFGW &amp; 0x01F + SUM(IDS)</td>
<td>EBF7</td>
<td>D163</td>
</tr>
<tr>
<td>rPIC12C509F</td>
<td>OFF</td>
<td>SUM[0x000:0x3FE] + CFGW &amp; 0x01F</td>
<td>EC20</td>
<td>DA68</td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>SUM[0x000:0x03F] + CFGW &amp; 0x01F + SUM(IDS)</td>
<td>EBF7</td>
<td>D163</td>
</tr>
</tbody>
</table>

Legend:  
CFGW = Configuration Word  
SUM[a:b] = [Sum of locations a through b inclusive]  
SUM_ID = ID locations masked by 0xF then made into a 16-bit value with ID0 as the most significant nibble. For example, ID0 = 0x12, ID1 = 0x37, ID2 = 0x4, ID3 = 0x26, then SUM_ID = 0x2746.  
*Checksum = [Sum of all the individual expressions] MODULO [0xFFFF]  
+ = Addition  
& = Bitwise AND
## 5.0 PROGRAM/VERIFY MODE ELECTRICAL CHARACTERISTICS

### TABLE 5-1: AC/DC CHARACTERISTICS

#### TIMING REQUIREMENTS FOR PROGRAM/VERIFY MODE

<table>
<thead>
<tr>
<th>Parameter No.</th>
<th>Sym.</th>
<th>Characteristic</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Units</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD1</td>
<td>VDDP</td>
<td>Supply voltage during programming</td>
<td>4.75</td>
<td>5.0</td>
<td>5.25</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>PD2</td>
<td>IDDP</td>
<td>Supply current (from VDD) during programming</td>
<td></td>
<td></td>
<td>20</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>PD3</td>
<td>VDDV</td>
<td>Supply voltage during verify</td>
<td>VDDMIN</td>
<td></td>
<td>VDDMAX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD4</td>
<td>VHH1</td>
<td>Voltage on MCLR/VPP during programming</td>
<td>12.75</td>
<td></td>
<td>13.25</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>PD5</td>
<td>VHH2</td>
<td>Voltage on MCLR/VPP during verify</td>
<td>VDD + 4.0</td>
<td></td>
<td>13.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD6</td>
<td>IPP</td>
<td>Programming supply current (from VPP)</td>
<td></td>
<td></td>
<td>50</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>PD9</td>
<td>VIH1</td>
<td>(GP1, GP0) input high level</td>
<td>0.8 VDD</td>
<td></td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
</tr>
<tr>
<td>PD8</td>
<td>VIL1</td>
<td>(GP1, GP0) input low level</td>
<td>0.2 VDD</td>
<td></td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
</tr>
</tbody>
</table>

#### Serial Program Verify

<table>
<thead>
<tr>
<th>Parameter No.</th>
<th>Sym.</th>
<th>Characteristic</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Units</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>TR</td>
<td>MCLR/VPP rise time (VSS to VHH)</td>
<td>8.0</td>
<td></td>
<td></td>
<td>µs</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>TF</td>
<td>MCLR fall time</td>
<td>8.0</td>
<td></td>
<td></td>
<td>µs</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>Tset1</td>
<td>Data in setup time before clock ↓</td>
<td>100</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>Thld1</td>
<td>Data in hold time after clock ↓</td>
<td>100</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>Tdly1</td>
<td>Data input not driven to next clock input (delay required between command/data or command/command)</td>
<td>1.0</td>
<td></td>
<td></td>
<td>µs</td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>Tdly2</td>
<td>Delay between clock ↓ to clock ↑ of next command or data</td>
<td>1.0</td>
<td></td>
<td></td>
<td>µs</td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td>Tdly3</td>
<td>Clock ↑ to date out valid (during read data)</td>
<td>200</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td>Thld0</td>
<td>Hold time after MCLR ↑</td>
<td>2</td>
<td></td>
<td></td>
<td>µs</td>
<td></td>
</tr>
</tbody>
</table>

**Note 1**: Program must be verified at the minimum and maximum VDD limits for the part.

**Note 2**: VHH must be greater than VDD + 4.5V to stay in Programming/Verify mode.
FIGURE 5-1: LOAD DATA COMMAND (PROGRAM/VERIFY)

FIGURE 5-2: READ DATA COMMAND (PROGRAM/VERIFY)

FIGURE 5-3: INCREMENT ADDRESS COMMAND (PROGRAM/VERIFY)
1.0 PROGRAMMING THE PIC12C67X AND PIC12CE67X

The PIC12C67X and PIC12CE67X can be programmed using a serial method. In Serial mode, the PIC12C67X and PIC12CE67X can be programmed while in the users system. This allows for increased design flexibility.

1.1 Hardware Requirements

The PIC12C67X and PIC12CE67X require two programmable power supplies, one for Vdd (2.0V to 6.0V recommended) and one for Vpp (12V to 14V). Both supplies should have a minimum resolution of 0.25V.

1.2 Programming Mode

The Programming mode for the PIC12C67X and PIC12CE67X allows programming of user program memory, special locations used for ID, and the configuration word for the PIC12C67X and PIC12CE67X.

Pin Diagrams:
2.0 PROGRAM MODE ENTRY

2.1 User Program Memory Map

The user memory space extends from 0x0000 to 0x1FFF (8K). Table 2-1 shows actual implementation of program memory in the PIC12C67X family.

**TABLE 2-1: IMPLEMENTATION OF PROGRAM MEMORY IN THE PIC12C67X**

<table>
<thead>
<tr>
<th>Device</th>
<th>Program Memory Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC12C671/</td>
<td>0x000 - 0x3FF (1K)</td>
</tr>
<tr>
<td>PIC12CE673</td>
<td></td>
</tr>
<tr>
<td>PIC12C672/</td>
<td>0x000 - 0x7FF (2K)</td>
</tr>
<tr>
<td>PIC12CE674</td>
<td></td>
</tr>
</tbody>
</table>

When the PC reaches the last location of the implemented program memory, it will wrap around and address a location within the physically implemented memory (see Figure 2-1).

In Programming mode, the program memory space extends from 0x0000 to 0x3FFF, with the first half (0x0000-0x1FFF) being user program memory and the second half (0x2000-0x3FFF) being configuration memory. The PC will increment from 0x0000 to 0x1FFF and wrap to 0x000 or 0x2000 to 0x3FFF and wrap around to 0x2000 (not to 0x0000). Once in configuration memory, the highest bit of the PC stays a ‘1’, thus always pointing to the configuration memory. The only way to point to user program memory is to reset the part and reenter Program/Verify mode, as described in Section 2.2.

The last location of the program memory space holds the factory programmed oscillator calibration value. This location should not be programmed, except when blank (a non-blank value should not cause the device to fail a blank check). If blank, the programmer should program it to a `RETLW XX` statement where “XX” is the calibration value.

In the configuration memory space, 0x2000-0x20FF are utilized. When in configuration memory, as in the user memory, the 0x2000-0x2XFF segment is repeatedly accessed as the PC exceeds 0x2XFF (see Figure 2-1).

A user may store identification information (ID) in four ID locations. The ID locations are mapped in [0x2000:0x2003].

**Note 1:** All other locations in PICmicro® MCU configuration memory are reserved and should not be programmed.

**Note 2:** Due to the secure nature of the on-board EEPROM memory in the PIC12CE673/674, it can be accessed only by the user program.
FIGURE 2-1: PROGRAM MEMORY MAPPING

PIC12C67X and PIC12CE67X

2000  ID Location
2001  ID Location
2002  ID Location
2003  ID Location
2004  Reserved
2005  Reserved
2006  Reserved
2007  Configuration Word

0 1KW 2KW

1FF Implemented Implemented
3FF
400 Implemented
7FF Implemented
800
BFF Reserved
C00 Reserved
FFF Reserved
1000 Reserved
1000
2000 Reserved
2008 Reserved
2100 Reserved
3FFF Reserved
2.2 Program/Verify Mode

The Program/Verify mode is entered by holding pins GP1 and GP0 low, while raising MCLR pin from V\text{IL} to V\text{IH} (high voltage). V\text{DD} is then raised from V\text{IL} to V\text{IH}. Once in this mode, the user program memory and the configuration memory can be accessed and programmed in serial fashion. The mode of operation is serial, and the memory that is accessed is the user program memory. GP1 is a Schmitt Trigger input in this mode.

The sequence that enters the device into the Program/Verify mode places all other logic into the RESET state (the MCLR pin was initially at V\text{IL}). This means that all I/O are in the RESET state (High impedance inputs).

The GP1 pin is used as a clock input pin, and the GP0 pin is used for entering command bits and data input/output during serial operation. To input a command, the clock pin (GP1) is cycled six times. Each command bit is latched on the falling edge of the clock with the least significant bit (LSb) of the command being input first. The data on pin GP0 is required to have a minimum setup and hold time (see AC/DC specs), with respect to the falling edge of the clock. Commands that have data associated with them (read and load) are specified to have a minimum delay of 1\mu s between the command and the data. After this delay, the clock pin is cycled 16 times with the first cycle being a START bit and the last cycle being a STOP bit. Data is also input and output LSB first. Therefore, during a read operation, the LSB will be transmitted onto pin GP0 on the rising edge of the second cycle, and during a load operation, the LSb will be latched on the falling edge of the second cycle. A minimum 1\mu s delay is also specified between consecutive commands.

All commands are transmitted LSB first. Data words are also transmitted LSB first. The data is transmitted on the rising edge and latched on the falling edge of the clock. To allow for decoding of commands and reversal of data pin configuration, a time separation of at least 1\mu s is required between a command and a data word (or another command).

The commands that are available are listed in Table 2-2.

2.2.1 Load Configuration

After receiving this command, the program counter (PC) will be set to 0x2000. By then applying 16 cycles to the clock pin, the chip will load 14-bits, a “data word” as described above, to be programmed into the configuration memory. A description of the memory mapping schemes for normal operation and Configuration mode operation is shown in Figure 2-1. After the configuration memory is entered, the only way to get back to the user program memory is to exit the Program/Verify Test mode by taking MCLR low (V\text{IL}).

<table>
<thead>
<tr>
<th>TABLE 2-2: COMMAND MAPPING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
</tr>
<tr>
<td>Load Configuration</td>
</tr>
<tr>
<td>Load Data</td>
</tr>
<tr>
<td>Read Data</td>
</tr>
<tr>
<td>Increment Address</td>
</tr>
<tr>
<td>Begin programming</td>
</tr>
<tr>
<td>End Programming</td>
</tr>
</tbody>
</table>

Note 1: The MCLR pin must be raised from V\text{IL} to V\text{IH} before V\text{DD} is applied. This is to ensure that the device does not have the PC incremented while in valid operation range.

2: Do not power GP2, GP4 or GP5 before V\text{DD} is applied.
Start

Set VPP = Vih1

Set VDD = VDDP*

N = 0

Program Cycle

Read Data Command

Data Correct?

Yes

Apply 3N Additional Program Cycles

All Locations Done?

Yes

Verified all Locations @ VDDMIN*

VPP = VihH1

Data Correct?

Yes

Verified all Locations @ VDDMAX

VPP = VihH1

Data Correct?

Yes

Done

No

No

No

No

N = N + 1

N = # of Program Cycles

Yes

N > 25

Report Programming Failure

Increment Address Command

Begin Programming Command

Wait 100 µs

End Programming Command

Report Verify @ VDDMIN Error

Report Verify @ VDDMAX Error

N = 0

All Locations Done?

VDDP = VDD range for programming (typically 4.75V - 5.25V).

VDDMIN = Minimum VDD for device operation.

VDDMAX = Maximum VDD for device operation.
FIGURE 2-3: PROGRAM FLOW CHART - PIC12C67X AND PIC12CE67X CONFIGURATION
WORD & ID LOCATIONS

Start

Set VPP = Vih1

Load Configuration Command

N = 0

Program ID Loc?

Yes

Program Cycle

N = N + 1

N = # of Program Cycles

No

Data Correct?

Yes

Apply 3N Program Cycles

Data Correct?

Yes

Read Data Command

N > 25

Yes

ID/Configuration Error

N = 0

Data Correct?

Yes

Read Data Command

N = N + 1

No

Increment Address Command

Address = 2004

Yes

Increment Address Command

No

Increment Address Command

Increment Address Command

Increment Address Command

Increment Address Command

Program Cycle 100 Cycles

Read Data Command

Data Correct?

Yes

Set VDD = VDDMIN

Report Program ID/Config. Error

Yes

Read Data Command

Set VPP = VIH2

No

Data Correct?

Yes

Read Data Command

Set VPP = VIH2

No

Data Correct?

Yes

Read Data Command

Set VPP = VIH2

Done
2.2.1.2 Load Data
After receiving this command, the chip will load in a 14-bit "data word" when 16 cycles are applied, as described previously. A timing diagram for the load data command is shown in Figure 5-1.

2.2.1.3 Read Data
After receiving this command, the chip will transmit data bits out of the memory currently accessed starting with the second rising edge of the clock input. The GP0 pin will go into Output mode on the second rising clock edge, and it will revert back to Input mode (hi-impedance) after the 16th rising edge. A timing diagram of this command is shown in Figure 5-2.

2.2.1.4 Increment Address
The PC is incremented when this command is received. A timing diagram of this command is shown in Figure 5-3.

2.2.1.5 Begin Programming
A load command (load configuration or load data) must be given before every begin programming command. Programming of the appropriate memory (test program memory or user program memory) will begin after this command is received and decoded. Programming should be performed with a series of 100\(\mu\)s programming pulses. A programming pulse is defined as the time between the begin programming command and the end programming command.

2.2.1.6 End Programming
After receiving this command, the chip stops programming the memory (configuration program memory or user program memory) that it was programming at the time.

2.3 Programming Algorithm Requires Variable VDD
The PIC12C67X and PIC12CE67X uses an intelligent algorithm. The algorithm calls for program verification at \(V_{DDMIN}\) as well as \(V_{DDMAX}\). Verification at \(V_{DDMIN}\) guarantees good "erase margin". Verification at \(V_{DDMAX}\) guarantees good "program margin".

The actual programming must be done with VDD in the \(V_{DDP}\) range (4.75 - 5.25V).
\[ V_{DDP} = V_{CC} \text{ range required during programming.} \]
\[ V_{DDMIN} = \text{minimum operating VDD spec for the part.} \]
\[ V_{DDMAX} = \text{maximum operating VDD spec for the part.} \]

Programmers must verify the PIC12C67X and PIC12CE67X at its specified \(V_{DDmax}\) and \(V_{DDmin}\) levels. Since Microchip may introduce future versions of the PIC12C67X and PIC12CE67X with a broader VDD range, it is best that these levels are user selectable (defaults are ok).

**Note:** Any programmer not meeting these requirements may only be classified as "prototype" or "development" programmer, but not a "production" quality programmer.
3.0 CONFIGURATION WORD

The PIC12C67X and PIC12CE67X family members have several configuration bits. These bits can be programmed (reads '0'), or left unprogrammed (reads '1'), to select various device configurations. Figure 3-1 provides an overview of configuration bits.

**FIGURE 3-1: CONFIGURATION WORD**

<table>
<thead>
<tr>
<th>Bit Number:</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP1</td>
<td>CP0</td>
<td>CP0</td>
<td>CP1</td>
<td>CP0</td>
<td>CP0</td>
<td>MCLRE</td>
<td>CP1</td>
<td>CP0</td>
<td>PWRTE</td>
<td>WDTE</td>
<td>FOSC2</td>
<td>FOSC1</td>
<td>FOSC0</td>
<td></td>
</tr>
<tr>
<td>Register: CONFIG</td>
<td>Address: 2007h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

bits 13-8, **CP1:CP0**: Code Protection bits\(^{(1)}\)\(^{(2)}\)
- 11 = Code protection off
- 10 = 0400h-07FFh code protected
- 01 = 0200h-07FFh code protected
- 00 = 0000h-07FFh code protected

bit 7 **MCLRE**: GP3/MCLR Pin Function Select
- 1 = GP3/MCLR pin function is MCLR
- 0 = GP3/MCLR pin function is digital I/O, MCLR internally tied to VDD

bit 4 **PWRTE**: Power-up Timer Enable bit\(^{(1)}\)
- 1 = PWRT disabled
- 0 = PWRT enabled

bit 3 **WDTE**: Watchdog Timer Enable bit
- 1 = WDT enabled
- 0 = WDT disabled

bit 2-0 **FOSC2:FOSC0**: Oscillator Selection bits
- 111 = EXTRC oscillator/CLKOUT function on GP4/OSC2/CLKOUT pin
- 110 = EXTRC oscillator/GP4 function on GP4/OSC2/CLKOUT pin
- 101 = INTRC oscillator/CLKOUT function on GP4/OSC2/CLKOUT pin
- 100 = INTRC oscillator/GP4 function on GP4/OSC2/CLKOUT pin
- 011 = invalid selection
- 010 = HS oscillator
- 001 = XT oscillator
- 000 = LP oscillator

**Note 1**: All of the CP1:CP0 pairs have to be given the same value to enable the code protection scheme listed.

**Note 2**: 07FFh is always uncode protected on the PIC12C672 and 03FFh is always uncode protected on the PIC12C671. This location contains the `RETLW 0xXX` calibration instruction for the INTRC.
4.0 CODE PROTECTION

The program code written into the EPROM can be protected by writing to the CP0 and CP1 bits of the configuration word.

For PIC12C67X and PIC12CE67X devices, once code protection is enabled, all protected segments read '0's (or "garbage values") and are prevented from further programming. All unprotected segments, including ID and configuration word locations, and calibration word location read normally and can be programmed.

4.1 Embedding Configuration Word and ID Information in the HEX File

To allow portability of code, the programmer is required to read the configuration word and ID locations from the HEX file when loading the HEX file. If configuration word information was not present in the HEX file then a simple warning message may be issued. Similarly, while saving a HEX file, configuration word and ID information must be included. An option to not include this information may be provided.

Microchip Technology Inc. feels strongly that this feature is important for the benefit of the end customer.

<table>
<thead>
<tr>
<th>TABLE 4-1: CONFIGURATION WORD</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th></th>
<th>PIC12C671, PIC12CE673</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>To code protect:</strong></td>
<td></td>
</tr>
<tr>
<td>• Protect all memory</td>
<td>00 0000 X00X XXXX</td>
</tr>
<tr>
<td>• Protect 0200h-07FFh</td>
<td>01 0101 X01X XXXX</td>
</tr>
<tr>
<td>• No code protection</td>
<td>11 1111 X11X XXXX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0x2007)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>Unprotected Memory Segment</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>Protected Memory Segment</td>
<td>Read All 0's, Write Disabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>ID Locations (0x2000 : 0x2003)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>INTRC Calibration Word (0X3FF)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>PIC12C672, PIC12CE674</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>To code protect:</strong></td>
<td></td>
</tr>
<tr>
<td>• Protect all memory</td>
<td>00 0000 X00X XXXX</td>
</tr>
<tr>
<td>• Protect 0200h-07FFh</td>
<td>01 0101 X01X XXXX</td>
</tr>
<tr>
<td>• Protect 0400h-07FFh</td>
<td>10 1010 X10X XXXX</td>
</tr>
<tr>
<td>• No code protection</td>
<td>11 1111 X11X XXXX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0x2007)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>Unprotected Memory Segment</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>Protected Memory Segment</td>
<td>Read All 0's, Write Disabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>ID Locations (0x2000 : 0x2003)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>INTRC Calibration Word (0X7FF)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
</tbody>
</table>
4.2 Checksum

4.2.1 CHECKSUM CALCULATIONS

Checksum is calculated by reading the contents of the PIC12C67X and PIC12CE67X memory locations and adding the opcodes up to the maximum user addressable location, excluding the oscillator calibration location in the last address, e.g., 0x3FE for the PIC12C671/CE673. Any carry bits exceeding 16-bits are neglected. Finally, the configuration word (appropriately masked) is added to the checksum. Checksum computation for each member of the PIC12C67X and PIC12CE67X devices is shown in Table 4-2.

The checksum is calculated by summing the following:
- The contents of all program memory locations
- The configuration word, appropriately masked
- Masked ID locations (when applicable)

The least significant 16 bits of this sum is the checksum.

The following table describes how to calculate the checksum for each device. Note that the checksum calculation differs depending on the code protect setting. Since the program memory locations read out differently depending on the code protect setting, the table describes how to manipulate the actual program memory values to simulate the values that would be read from a protected device. When calculating a checksum by reading a device, the entire program memory can simply be read and summed. The configuration word and ID locations can always be read.

Note that some older devices have an additional value added in the checksum. This is to maintain compatibility with older device programmer checksums.

### TABLE 4-2: CHECKSUM COMPUTATION

<table>
<thead>
<tr>
<th>Device</th>
<th>Code Protect</th>
<th>Checksum*</th>
<th>Blank Value</th>
<th>Ox25E6 at 0 and max address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC12C671</td>
<td>OFF</td>
<td>SUM[0x000:0x3FE] + CFGW &amp; 0x3FFF</td>
<td>FC00</td>
<td>C7CE</td>
</tr>
<tr>
<td>PIC12C673</td>
<td>1/2</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3FFF + SUM_ID</td>
<td>0FBF</td>
<td>C174</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3FFF + SUM_ID</td>
<td>FC9F</td>
<td>C86D</td>
</tr>
<tr>
<td>PIC12C672</td>
<td>OFF</td>
<td>SUM[0x000:0x7FE] + CFGW &amp; 0x3FFF</td>
<td>F800</td>
<td>C3CE</td>
</tr>
<tr>
<td>PIC12CE674</td>
<td>1/2</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3FFF + SUM_ID</td>
<td>1EDF</td>
<td>D094</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3FFF + SUM_ID</td>
<td>0BBF</td>
<td>BD74</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3FFF + SUM_ID</td>
<td>F89F</td>
<td>C46D</td>
</tr>
</tbody>
</table>

Legend:  
CFGW = Configuration Word  
SUM[a:b] = [Sum of locations a through b inclusive]  
SUM_ID = ID locations masked by 0xF then made into a 16-bit value with ID0 as the most significant nibble. For example, ID0 = 0x12, ID1 = 0x37, ID2 = 0x4, ID3 = 0x26, then SUM_ID = 0x2746.  
*Checksum = [Sum of all the individual expressions] MODULO [0xFFFF]  
+ = Addition  
& = Bitwise AND
# 5.0 PROGRAM/VERIFY MODE ELECTRICAL CHARACTERISTICS

## TABLE 5-1: AC/DC CHARACTERISTICS

### TIMING REQUIREMENTS FOR PROGRAM/VERIFY TEST MODE

<table>
<thead>
<tr>
<th>Parameter No.</th>
<th>Sym.</th>
<th>Characteristic</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Units</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>General</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD1</td>
<td>VDDP</td>
<td>Supply voltage during programming</td>
<td>4.75</td>
<td>5.0</td>
<td>5.25</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>PD2</td>
<td>IDD</td>
<td>Supply current (from VDD) during programming</td>
<td></td>
<td>20</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>PD3</td>
<td>VDDV</td>
<td>Supply voltage during verify</td>
<td>VDDMIN</td>
<td></td>
<td>VDDMAX</td>
<td>V</td>
<td>(Note 1)</td>
</tr>
<tr>
<td>PD4</td>
<td>VHH1</td>
<td>Voltage on MCLR/VPP during programming</td>
<td>12.75</td>
<td></td>
<td>13.25</td>
<td>V</td>
<td>(Note 2)</td>
</tr>
<tr>
<td>PD5</td>
<td>VHH2</td>
<td>Voltage on MCLR/VPP during verify</td>
<td>VDD + 4.0</td>
<td></td>
<td>13.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD6</td>
<td>IPP</td>
<td>Programming supply current (from VPP)</td>
<td></td>
<td>50</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>PD9</td>
<td>VIH1</td>
<td>(GP0, GP1) input high level</td>
<td>0.8 VDD</td>
<td></td>
<td></td>
<td>V</td>
<td>Schmitt Trigger input</td>
</tr>
<tr>
<td>PD8</td>
<td>VIL1</td>
<td>(GP0, GP1) input low level</td>
<td>0.2 VDD</td>
<td></td>
<td></td>
<td>V</td>
<td>Schmitt Trigger input</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter No.</th>
<th>Sym.</th>
<th>Characteristic</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Units</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Serial Program Verify</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>TR</td>
<td>MCLR/VPP rise time (VSS to VHH) for Test mode entry</td>
<td>8.0</td>
<td></td>
<td></td>
<td>µs</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>TF</td>
<td>MCLR Fall time</td>
<td>8.0</td>
<td></td>
<td></td>
<td>µs</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>TSET1</td>
<td>Data in setup time before clock ↓</td>
<td>100</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>THLD1</td>
<td>Data in hold time after clock ↓</td>
<td>100</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>TDL1</td>
<td>Data input not driven to next clock input (delay required between command/data or command/command)</td>
<td>1.0</td>
<td></td>
<td></td>
<td>µs</td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>TDL2</td>
<td>Delay between clock ↓ to clock ↑ of next command or data</td>
<td>1.0</td>
<td></td>
<td></td>
<td>µs</td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td>TDL3</td>
<td>Clock ↑ to data out valid (during read data)</td>
<td>200</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td>THLD0</td>
<td>Hold time after VDD↑</td>
<td>2</td>
<td></td>
<td></td>
<td>µs</td>
<td></td>
</tr>
<tr>
<td>P9</td>
<td>TPEP</td>
<td>Hold time after VPP↑</td>
<td>5</td>
<td></td>
<td></td>
<td>µs</td>
<td></td>
</tr>
</tbody>
</table>

**Note 1:** Program must be verified at the minimum and maximum VDD limits for the part.

**Note 2:** VHH must be greater than VDD + 4.5V to stay in Programming/Verify mode.
FIGURE 5-1: LOAD DATA COMMAND (PROGRAM/VERIFY)

FIGURE 5-2: READ DATA COMMAND (PROGRAM/VERIFY)

FIGURE 5-3: INCREMENT ADDRESS COMMAND (PROGRAM/VERIFY)
This document includes the programming specifications for the following devices:

- PIC14000

1.0 PROGRAMMING THE PIC14000

The PIC14000 can be programmed using a serial method. In serial mode the PIC14000 can be programmed while in the user's system. This allows for increased design flexibility. This programming specification applies to PIC14000 devices in all packages.

1.1 Hardware Requirements

The PIC14000 requires two programmable power supplies, one for VDD (2.0V to 6.5V recommended) and one for VPP (12V to 14V).

1.2 Programming Mode

The programming mode for the PIC14000 allows programming of user program memory, configuration word, and calibration memory.
2.0 PROGRAM MODE ENTRY

2.1 User Program Memory Map

The program and calibration memory space extends from 0x000 to 0xFFF (4096 words). Table 2-1 shows actual implementation of program memory in the PIC14000.

**TABLE 2-1: IMPLEMENTATION OF PROGRAM AND CALIBRATION MEMORY IN THE PIC14000**

<table>
<thead>
<tr>
<th>Area</th>
<th>Memory Space</th>
<th>Access to Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>0x000-0xFFB</td>
<td>PC&lt;12:0&gt;</td>
</tr>
<tr>
<td>Calibration</td>
<td>0xFC0 -0xFFF</td>
<td>PC&lt;12:0&gt;</td>
</tr>
</tbody>
</table>

When the PC reaches address 0xFFF, it will wrap around and address a location within the physically implemented memory (see Figure 2-1).

In programming mode the program memory space extends from 0x0000 to 0x3FFF, with the first half (0x0000-0x1FFF) being user program memory and the second half (0x2000-0x3FFF) being configuration memory. The PC will increment from 0x0000 to 0x1FFF and wrap to 0x0000, or 0x2000 to 0x3FFF and wrap around to 0x2000 (not to 0x0000). Once in configuration memory, the highest bit of the PC stays a "1", thus always pointing to the configuration memory. The only way to point to user program memory is to reset the part and reenter program/verify mode, as described in Section 2.2.

In the configuration memory space, 0x2000-0x20FF are utilized. When in configuration memory, as in the user memory, the 0x2000-0x2XF segment is repeatedly accessed as PC exceeds 0x2XF (Figure 2-1).

A user may store identification information (ID) in four ID locations. The ID locations are mapped in [0x2000:0x2003]. All other locations are reserved and should not be programmed.

The ID locations read out normally, even after code protection. To understand how the devices behave, refer to Table 4-1.

To understand the scrambling mechanism after code protection, refer to Section 4.1.
FIGURE 2-1: PROGRAM MEMORY MAPPING

- ID Location
- ID Location
- ID Location
- ID Location
- Reserved
- Reserved
- Reserved
- Configuration Word
- Program
- Calibration
- Reserved
- Test
- Reserved
2.2 Program/Verify Mode

The program/verify mode is entered by holding pins RC6 and RC7 low while raising MCLR pin from VIL to VIH (high voltage). Once in this mode the user program memory and the configuration memory can be accessed and programmed in serial fashion. The mode of operation is serial, and the memory that is accessed is the user program memory. RC6 and RC7 are both Schmitt Trigger inputs in this mode.

The sequence that enters the device into the programming/verify mode places all other logic into the reset state (the MCLR pin was initially at VIL). This means that all I/O are in the reset state (High impedance inputs).

The RB6 pin is used as a clock input pin, and the RB7 pin is used for entering command bits and data input/output during serial operation. To input a command, the clock pin (RC6) is cycled six times. Each command bit is latched on the falling edge of the clock with the least significant bit (LSB) of the command being input first. The data on pin RC7 is required to have a minimum setup and hold time (see AC/DC specs) with respect to the falling edge of the clock. Commands that have data associated with them (read and load) are specified to have a minimum delay of 1μs between the command and the data. After this delay the clock pin is cycled 16 times with the first cycle being a start bit and the last cycle being a stop bit. Data is also input and output LSB first. Therefore, during a read operation the LSB will be transmitted onto pin RC7 on the rising edge of the second cycle, and during a load operation the LSB will be latched on the falling edge of the second cycle. A minimum 1μs delay is also specified between consecutive commands.

All commands are transmitted LSB first. Data words are also transmitted LSB first. The data is transmitted on the rising edge and latched on the falling edge of the clock. To allow for decoding of commands and reversal of data pin configuration, a time separation of at least 1μs is required between a command and a data word (or another command).

The commands that are available are listed in Table 2-1.

### 2.2.1 PROGRAM/VERIFY OPERATION

After receiving this command, the program counter (PC) will be set to 0x2000. By then applying 16 cycles to the clock pin, the chip will load 14-bits a “data word” as described above, to be programmed into the configuration memory. A description of the memory mapping schemes for normal operation and configuration mode operation is shown in Figure 2-1. After the configuration memory is entered, the only way to get back to the user program memory is to exit the program/verify test mode by taking MCLR low (VIL).

### TABLE 2-1: COMMAND MAPPING

<table>
<thead>
<tr>
<th>Command</th>
<th>Mapping (MSB ... LSB)</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Configuration</td>
<td>0 0 0 0 0 0</td>
<td>0, data(14), 0</td>
</tr>
<tr>
<td>Load Data</td>
<td>0 0 0 0 1 0</td>
<td>0, data(14), 0</td>
</tr>
<tr>
<td>Read Data</td>
<td>0 0 0 1 0 0</td>
<td>0, data(14), 0</td>
</tr>
<tr>
<td>Increment Address</td>
<td>0 0 0 1 1 0</td>
<td></td>
</tr>
<tr>
<td>Begin programming</td>
<td>0 0 1 0 0 0</td>
<td></td>
</tr>
<tr>
<td>End Programming</td>
<td>0 0 1 1 1 0</td>
<td></td>
</tr>
</tbody>
</table>

Note: The CPU clock must be disabled during in-circuit programming (to avoid incrementing the PC).
* V_{DDP} = V_{DD} range for programming (typically 4.75V - 5.25V).
V_{DDmin} = Minimum V_{DD} for device operation.
V_{DDmax} = Maximum V_{DD} for device operation.
2.2.1.2 LOAD DATA

After receiving this command, the chip will load in a 14-bit "data word" when 16 cycles are applied, as described previously. A timing diagram for the load data command is shown in Figure 5-1.

2.2.1.3 READ DATA

After receiving this command, the chip will transmit data bits out of the memory currently accessed starting with the second rising edge of the clock input. The RC7 pin will go into output mode on the second rising clock edge, and it will revert back to input mode (hi-impedance) after the 16th rising edge. A timing diagram of this command is shown in Figure 5-2.

2.2.1.4 INCREMENT ADDRESS

The PC is incremented when this command is received. A timing diagram of this command is shown in Figure 5-3.

2.2.1.5 BEGIN PROGRAMMING

A load command (load configuration or load data) must be given before every begin programming command. Programming of the appropriate memory (test program memory or user program memory) will begin after this command is received and decoded. Programming should be performed with a series of 100µs programming pulses. A programming pulse is defined as the time between the begin programming command and the end programming command.

2.2.1.6 END PROGRAMMING

After receiving this command, the chip stops programming the memory (configuration program memory or user program memory) that it was programming at the time.

2.3 Programming Algorithm Requires Variable Vdd

The PIC14000 uses an intelligent algorithm. The algorithm calls for program verification at VDDmin as well as VDDmax. Verification at VDDmin guarantees good "erase margin". Verification at VDDmax guarantees good "program margin".

The actual programming must be done with VDD in the VDDP range (4.75 - 5.25V).

VDDP = VCC range required during programming.
VDDmin = minimum operating VDD spec for the part.
VDDmax = maximum operating VDD spec for the part.

Programmers must verify the PIC14000 at its specified VDDmax and VDDmin levels. Since Microchip may introduce future versions of the PIC14000 with a broader VDD range, it is best that these levels are user selectable (defaults are ok).

Note: Any programmer not meeting these requirements may only be classified as "prototype" or "development" programmer but not a "production" quality programmer.
3.0 CONFIGURATION WORD

The PIC14000 has several configuration bits. These bits can be programmed (reads '0') or left unprogrammed (reads '1') to select various device configurations. Figure 3-1 provides an overview of configuration bits.

FIGURE 3-1: CONFIGURATION WORD BIT MAP

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC14000</td>
<td>CPC</td>
<td>CPP1</td>
<td>CPP0</td>
<td>CPP0</td>
<td>CPP1</td>
<td>CPC</td>
<td>CPC</td>
<td>F</td>
<td>CPP1</td>
<td>CPP0</td>
<td>PWRTE</td>
<td>WDTE</td>
<td>F</td>
<td>FOSC</td>
</tr>
</tbody>
</table>

**CPP<1:0>**
- 11: All Unprotected
- 10: N/A
- 01: N/A
- 00: All Protected

bit 1,6: F Internal trim, factory programmed. DO NOT CHANGE! Program as '1'. Note 1.

bit 3: **PWRTE**, Power Up Timer Enable Bit
- 0 = Power up timer enabled
- 1 = Power up timer disabled (unprogrammed)

bit 2: **WDTE**, WDT Enable Bit
- 0 = WDT disabled
- 1 = WDT enabled (unprogrammed)

bit 0: **FOSC<1:0>**, Oscillator Selection Bit
- 0: HS oscillator (crystal/resonator)
- 1: Internal RC oscillator (unprogrammed)

**Note 1:** See Section 4.1.2 for cautions.
4.0 CODE PROTECTION

The memory space in the PIC14000 is divided into two areas: program space (0-0xFBF) and calibration space (0xFC0-0xFFF).

For program space or user space, once code protection is enabled, all protected segments read '0's (or "garbage values") and are prevented from further programming. All unprotected segments, including ID locations and configuration word, read normally. These locations can be programmed.

4.1 Calibration Space

The calibration space can contain factory-generated and programmed values. For non-JW devices, the CPC bits in the configuration word are set to '0' at the factory, and the calibration data values are write-protected; they may still be read out, but not programmed. JW devices contain the factory values, but DO NOT have the CPC bits set.

Microchip does not recommend setting code protect bits in windowed devices to '0'. Once code-protected, the device cannot be reprogrammed.

4.1.1 CALIBRATION SPACE CHECKSUM

The data in the calibration space has its own checksum. When properly programmed, the calibration memory will always checksum to 0x0000. When this checksum is 0x0000, and the checksum of memory [0x0000:0xFBF] is 0x2FBF, the part is effectively blank, and the programmer should indicate such.

If the CPC bits are set to '1', but the checksum of the calibration memory is 0x0000, the programmer should NOT program locations in the calibration memory space, even if requested to do so by the operator. This would be the case for a new JW device.

If the CPC bits are set to '1', and the checksum of the calibration memory is NOT 0x0000, the programmer is allowed to program the calibration space as directed by the operator.

The calibration space contains specially coded data values used for device parameter calibration. The programmer may wish to read these values and display them for the operator's convenience. For further information on these values and their coding, refer to AN621 (DS00621B).

4.1.2 REPROGRAMMING CALIBRATION SPACE

The operator should be allowed to read and store the data in the calibration space, for future reprogramming of the device. This procedure is necessary for reprogramming a windowed device, since the calibration data will be erased along with the rest of the memory. When saving this data, Configuration Word <1,6> must also be saved, and restored when the calibration data is reloaded.

4.2 Embedding Configuration Word and ID Information in the Hex File

To allow portability of code, the programmer is required to read the configuration word and ID locations from the hex file when loading the hex file. If configuration word information was not present in the hex file then a simple warning message may be issued. Similarly, while saving a hex file, configuration word and ID information must be included. An option to not include this information may be provided.

Microchip Technology Inc. feels strongly that this feature is important for the benefit of the end customer.

<table>
<thead>
<tr>
<th>TABLE 4-1: CODE PROTECT OPTIONS</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0x2007)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>Unprotected memory segment</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>Protected memory segment</td>
<td>Read All 0's, Write Disabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>Protected calibration memory</td>
<td>Read Unscrambled, Write Disabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>ID Locations (0x2000 : 0x2003)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
</tbody>
</table>

Legend: X = Don't care
4.3 Checksum

4.3.1 CHECKSUM CALCULATIONS

Checksum is calculated by reading the contents of the PIC14000 memory locations and adding up the opcodes up to the maximum user addressable location, 0xFB. Any carry bits exceeding 16-bits are neglected. Finally, the configuration word (appropriately masked) is added to the checksum. Checksum computation for the PIC14000 device is shown in Table 4-2:

The checksum is calculated by summing the following:
- The contents of all program memory locations
- The configuration word, appropriately masked
- Masked ID locations (when applicable)

The least significant 16 bits of this sum is the checksum.

The following table describes how to calculate the checksum for each device. Note that the checksum calculation differs depending on the code protect setting. Since the program memory locations read out differently depending on the code protect setting, the table describes how to manipulate the actual program memory values to simulate the values that would be read from a protected device. When calculating a checksum by reading a device, the entire program memory can simply be read and summed. The configuration word and ID locations can always be read.

Note that some older devices have an additional value added in the checksum. This is to maintain compatibility with older device programmer checksums.

<table>
<thead>
<tr>
<th>Code Protect</th>
<th>Checksum*</th>
<th>Blank Value</th>
<th>0x25E6 at 0 and max address</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>SUM[0000:0FBF] + CFGW &amp; 0x3FBD</td>
<td>0x2FFD</td>
<td>0xFBCB</td>
</tr>
<tr>
<td>OFF OTP</td>
<td>SUM[0000:0FBF] + CFGW &amp; 0x3FBD</td>
<td>0x0E7D</td>
<td>0xDA4B</td>
</tr>
<tr>
<td>ON</td>
<td>CFGW &amp; 0x3FBD + SUM(IDs)</td>
<td>0x300A</td>
<td>0xFBD8</td>
</tr>
</tbody>
</table>

Legend: CFGW = Configuration Word
SUM[A:B] = [Sum of locations a through b inclusive]
SUM(ID) = ID locations masked by 0x7F then made into a 28-bit value with ID0 as the most significant byte
*Checksum = [Sum of all the individual expressions] MODULO [0xFFFF]
+ = Addition
& = Bitwise AND
# 5.0 PROGRAM/VERIFY Mode Electrical Characteristics

## Table 5-1: AC/DC Characteristics

### AC/DC Timing Requirements for Program/Verify Mode

<table>
<thead>
<tr>
<th>Parameter No.</th>
<th>Sym.</th>
<th>Characteristic</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Units</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD1</td>
<td>VDDP</td>
<td>Supply voltage during programming</td>
<td>4.75</td>
<td>5.0</td>
<td>5.25</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>PD2</td>
<td>IDD</td>
<td>Supply current (from VDD) during programming</td>
<td>–</td>
<td>–</td>
<td>20</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>PD3</td>
<td>VDDV</td>
<td>Supply voltage during verify</td>
<td>VDDmin</td>
<td>VDDmax</td>
<td>V</td>
<td>Note 1</td>
<td></td>
</tr>
<tr>
<td>PD4</td>
<td>VHH1</td>
<td>Voltage on MCLR/VPP during programming</td>
<td>12.75</td>
<td>–</td>
<td>13.25</td>
<td>V</td>
<td>Note 2</td>
</tr>
<tr>
<td>PD5</td>
<td>VHH2</td>
<td>Voltage on MCLR/VPP during verify</td>
<td>VDD + 4.0</td>
<td>–</td>
<td>13.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD6</td>
<td>IPP</td>
<td>Programming supply current (from VPP)</td>
<td>–</td>
<td>–</td>
<td>50</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>PD9</td>
<td>VH1</td>
<td>(RC6, RC7) input high level</td>
<td>0.8 VDD</td>
<td>–</td>
<td>–</td>
<td>V</td>
<td>Schmitt Trigger input</td>
</tr>
<tr>
<td>PD8</td>
<td>VL1</td>
<td>(RC6, RC7) input low level</td>
<td>0.2 VDD</td>
<td>–</td>
<td>–</td>
<td>V</td>
<td>Schmitt Trigger input</td>
</tr>
</tbody>
</table>

### Serial Program Verify

<table>
<thead>
<tr>
<th>Parameter No.</th>
<th>Sym.</th>
<th>Characteristic</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Units</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>TR</td>
<td>MCLR/VPP rise time (VSS to VHH) for test mode entry</td>
<td>–</td>
<td>–</td>
<td>8.0</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>Tf</td>
<td>MCLR Fall time</td>
<td>–</td>
<td>–</td>
<td>8.0</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>Tset1</td>
<td>Data in setup time before clock Ø</td>
<td>100</td>
<td>–</td>
<td>–</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>Thld1</td>
<td>Data in hold time after clock Ø</td>
<td>100</td>
<td>–</td>
<td>–</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>Tdly1</td>
<td>Data input not driven to next clock input (delay required between command/data or command/command)</td>
<td>1.0</td>
<td>–</td>
<td>–</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>Tdly2</td>
<td>Delay between clock Ø to clock ‖ of next command or data</td>
<td>1.0</td>
<td>–</td>
<td>–</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td>Tdly3</td>
<td>Clock ‖ to date out valid (during read data)</td>
<td>200</td>
<td>–</td>
<td>–</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td>Thld0</td>
<td>Hold time after MCLR ‖</td>
<td>2</td>
<td>–</td>
<td>–</td>
<td>ms</td>
<td></td>
</tr>
</tbody>
</table>

Note 1: Program must be verified at the minimum and maximum VDD limits for the part.
Note 2: VHH must be greater than VDD + 4.5V to stay in programming/verify mode.
FIGURE 5-1: LOAD DATA COMMAND (PROGRAM/VERIFY)

FIGURE 5-2: READ DATA COMMAND (PROGRAM/VERIFY)

FIGURE 5-3: INCREMENT ADDRESS COMMAND (PROGRAM/VERIFY)
This document includes the programming specifications for the following devices:

- PIC16C554
- PIC16C556
- PIC16C558

1.0 PROGRAMMING THE PIC16C55X

The PIC16C55X can be programmed using a serial method. In serial mode the PIC16C55X can be programmed while in the users system. This allows for increased design flexibility.

1.1 Hardware Requirements

The PIC16C55X requires two programmable power supplies, one for VDD (2.0V to 6.5V recommended) and one for VPP (12V to 14V). Both supplies should have a minimum resolution of 0.25V.

1.2 Programming Mode

The programming mode for the PIC16C55X allows programming of user program memory, special locations used for ID, and the configuration word for the PIC16C55X.
2.0 PROGRAM MODE ENTRY

2.1 User Program Memory Map

The user memory space extends from 0x0000 to 0x1FFF (8K). Table 2-1 shows actual implementation of program memory in the PIC16C55X family.

TABLE 2-1: IMPLEMENTATION OF PROGRAM MEMORY IN THE PIC16C55X

<table>
<thead>
<tr>
<th>Device</th>
<th>Program Memory Size</th>
<th>Access to Program Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C554</td>
<td>0x000 - 0x1FF (0.5K)</td>
<td>PC&lt;8:0&gt;</td>
</tr>
<tr>
<td>PIC16C556</td>
<td>0x000 - 0x3FF (1K)</td>
<td>PC&lt;9:0&gt;</td>
</tr>
<tr>
<td>PIC16C558</td>
<td>0x000 - 0x7FF (2K)</td>
<td>PC&lt;10:0&gt;</td>
</tr>
</tbody>
</table>

When the PC reaches the last location of the implemented program memory, it will wrap around and address a location within the physically implemented memory (see Figure 2-1).

In programming mode the program memory space extends from 0x0000 to 0x3FFF, with the first half (0x0000-0x1FFF) being user program memory and the second half (0x2000-0x3FFF) being configuration memory. The PC will increment from 0x0000 to 0x1FFF and wrap to 0x000 or 0x2000 to 0x3FFF and wrap around to 0x2000 (not to 0x0000). Once in configuration memory, the highest bit of the PC stays a ‘1’, thus always pointing to the configuration memory. The only way to point to user program memory is to reset the part and reenter program/verify mode, as described in Section 2.2.

In the configuration memory space, 0x2000-0x20FF are utilized. When in a configuration memory, as in the user memory, the 0x2000-0x2XFF segment is repeatedly accessed as the PC exceeds 0x2XFF (see Figure 2-1).

A user may store identification information (ID) in four ID locations. The ID locations are mapped in [0x2000:0x2003]. It is recommended that the user use only the four least significant bits of each ID location. In some devices, the ID locations read-out in a scrambled fashion after code protection is enabled. For these devices, it is recommended that ID location is written as “11 1111 1000 bbbb” where ‘bbbb’ is ID information.

Note: All other locations are reserved and should not be programmed.

In other devices, the ID locations read out normally, even after code protection. To understand how the devices behave, refer to Table 4-1.

To understand the scrambling mechanism after code protection, refer to Section 4.1.
FIGURE 2-1: PROGRAM MEMORY MAPPING

<table>
<thead>
<tr>
<th>ID Location</th>
<th>ID Location</th>
<th>ID Location</th>
<th>ID Location</th>
<th>Reserved</th>
<th>Reserved</th>
<th>Reserved</th>
<th>Configuration Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1FF</td>
<td>1FF</td>
<td>1FF</td>
<td>2000</td>
<td>2001</td>
<td>2002</td>
<td></td>
</tr>
<tr>
<td>0.5KW</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>1KW</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>2KW</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

ID Location:
- ID 0: Configuration Word
- ID 1: Configuration Word
- ID 2: Configuration Word
- ID 3: Configuration Word
- ID 4: Configuration Word
- ID 5: Configuration Word
- ID 6: Configuration Word
- ID 7: Configuration Word

Reserved: All memory locations not used for specific functions.
2.2 Program/Verify Mode

The program/verify mode is entered by holding pins RB6 and RB7 low while raising MCLR pin from VIL to VIH (high voltage). Once in this mode the user program memory and the configuration memory can be accessed and programmed in serial fashion. The mode of operation is serial, and the memory that is accessed is the user program and configuration memory. RB6 is a Schmitt Trigger input in this mode.

The sequence that enters the device into the programming/verify mode places all other logic into the reset state (the MCLR pin was initially at VIL). This means that all I/O are in the reset state (High impedance inputs).

Note: The MCLR pin should be raised as quickly as possible from VIL to VIH. This is to ensure that the device does not have the PC incremented while in valid operation range.

2.2.1 PROGRAM/VERIFY OPERATION

The RB6 pin is used as a clock input pin, and the RB7 pin is used for entering command bits and data input/output during serial operation. To input a command, the clock pin (RB6) is cycled six times. Each command bit is latched on the falling edge of the clock with the least significant bit (LSB) of the command being input first. The data on pin RB7 is required to have a minimum setup and hold time (see AC/DC specs) with respect to the falling edge of the clock. Commands that have data associated with them (read and load) are specified to have a minimum delay of 1ms between the command and the data. After this delay the clock pin is cycled 16 times with the first cycle being a start bit and the last cycle being a stop bit. Data is also input and output LSB first. Therefore, during a read operation the LSB will be transmitted onto pin RB7 on the rising edge of the second cycle, and during a load operation the LSB will be latched on the falling edge of the second cycle. A minimum 1ms delay is also specified between consecutive commands.

The commands that are available are listed in Table 2-1.

2.2.1.1 LOAD CONFIGURATION

After receiving this command, the program counter (PC) will be set to 0x2000. By then applying 16 cycles to the clock pin, the chip will load 14-bits a “data word” as described above, to be programmed into the configuration memory. A description of the memory mapping schemes for normal operation and configuration mode operation is shown in Figure 2-1. After the configuration memory is entered, the only way to get back to the user program memory is to exit the program/verify test mode by taking MCLR low (VIL).

<table>
<thead>
<tr>
<th>TABLE 2-1: COMMAND MAPPING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
</tr>
<tr>
<td>Load Configuration</td>
</tr>
<tr>
<td>Load Data</td>
</tr>
<tr>
<td>Read Data</td>
</tr>
<tr>
<td>Increment Address</td>
</tr>
<tr>
<td>Begin programming</td>
</tr>
<tr>
<td>End Programming</td>
</tr>
</tbody>
</table>

Note: The CPU clock must be disabled during in-circuit programming.
* $V_{DDP} = V_{DD}$ range for programming (typically 4.75V - 5.25V).

$V_{DD}^{\text{min}} = \text{Minimum } V_{DD} \text{ for device operation.}$

$V_{DD}^{\text{max}} = \text{Maximum } V_{DD} \text{ for device operation.}$
FIGURE 2-3: PROGRAM FLOW CHART - PIC16C55X CONFIGURATION WORD & ID LOCATIONS

Start → Load Configuration Command → N = 0 → Program ID Loc? → Yes → Program Cycle → Read Data Command → Data Correct? → Yes → Set VDD = VDDmin

No → Address = 2004 → Yes → Increment Address Command

No → Address = 2004 → No → Increment Address Command

No → Address = 2004 → No → Increment Address Command

Program Cycle 100 Cycles → Read Data Command → Data Correct? → Yes → Set VDD = VDDmax

Read Data Command → Data Correct? → Yes → Read Data Command

Set VPP = VIH+2

2.2.1.2 LOAD DATA

After receiving this command, the chip will load in a 14-bit “data word” when 16 cycles are applied, as described previously. A timing diagram for the load data command is shown in Figure 5-1.

2.2.1.3 READ DATA

After receiving this command, the chip will transmit data bits out of the memory currently accessed starting with the second rising edge of the clock input. The RB7 pin will go into output mode on the second rising clock edge, and it will revert back to input mode (hi-impedance) after the 16th rising edge. A timing diagram of this command is shown in Figure 5-2.

2.2.1.4 INCREMENT ADDRESS

The PC is incremented when this command is received. A timing diagram of this command is shown in Figure 5-3.

2.2.1.5 BEGIN PROGRAMMING

A load command (load configuration or load data) must be given before every begin programming command. Programming of the appropriate memory (test program memory or user program memory) will begin after this command is received and decoded. Programming should be performed with a series of 100ms programming pulses. A programming pulse is defined as the time between the begin programming command and the end programming command.

2.2.1.6 END PROGRAMMING

After receiving this command, the chip stops programming the memory (configuration program memory or user program memory) that it was programming at the time.

2.3 Programming Algorithm Requires Variable Vdd

The PIC16C55X uses an intelligent algorithm. The algorithm calls for program verification at Vddmin as well as Vddmax. Verification at Vddmin guarantees good “erase margin”. Verification at Vddmax guarantees good “program margin”.

The actual programming must be done with Vdd in the Vddp range (4.75 - 5.25V).

Vddp = Vcc range required during programming.

Vdd min. = minimum operating Vdd spec for the part.

Vdd max. = maximum operating Vdd spec for the part.

Programmers must verify the PIC16C55X at its specified Vddmax and Vddmin levels. Since Microchip may introduce future versions of the PIC16C55X with a broader Vdd range, it is best that these levels are user selectable (defaults are ok).

Note: Any programmer not meeting these requirements may only be classified as “prototype” or “development” programmer but not a “production” quality programmer.
3.0 CONFIGURATION WORD

The PIC16C55X family members have several configuration bits. These bits can be programmed (reads '0') or left unprogrammed (reads '1') to select various device configurations. Figure 3-1 provides an overview of configuration bits.

FIGURE 3-1: CONFIGURATION WORD BIT MAP

<table>
<thead>
<tr>
<th>Bit Number:</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C554/556/558</td>
<td>CP1</td>
<td>CP0</td>
<td>CP1</td>
<td>CP0</td>
<td>CP1</td>
<td>CP0</td>
<td>—</td>
<td>0</td>
<td>CP1</td>
<td>CP0</td>
<td>PWRTE</td>
<td>WDTE</td>
<td>FOSC1</td>
<td>FOSC0</td>
</tr>
</tbody>
</table>

bit 7: Reserved for future use
bit 6: Set to 0
bit 5-4: CP1:CP0, Code Protect
bit 8-13

<table>
<thead>
<tr>
<th>Device</th>
<th>CP1</th>
<th>CP0</th>
<th>Code Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C554</td>
<td>0</td>
<td>0</td>
<td>All memory protected</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>Do not use</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>Do not use</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>Code protection off</td>
</tr>
<tr>
<td>PIC16C556</td>
<td>0</td>
<td>0</td>
<td>All memory protected</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>Upper 1/2 memory protected</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>Do not use</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>Code protection off</td>
</tr>
<tr>
<td>PIC16C558</td>
<td>0</td>
<td>0</td>
<td>All memory protected</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>Upper 3/4 memory protected</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>Upper 1/2 memory protected</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>Code protection off</td>
</tr>
</tbody>
</table>

bit 3: **PWRTE**, Power Up Timer Enable Bit
PIC16C554/556/558:
0 = Power up timer enabled
1 = Power up timer disabled

bit 2: **WDTE**, WDT Enable Bit
1 = WDT enabled
0 = WDT disabled

bit 1-0: **FOSC<1:0>**, Oscillator Selection Bit
11: RC oscillator
10: HS oscillator
01: XT oscillator
00: LP oscillator
4.0 CODE PROTECTION

The program code written into the EPROM can be protected by writing to the CP0 & CP1 bits of the configuration word.

4.1 Programming Locations 0x0000 to 0x03F after Code Protection

For PIC16C55X devices, once code protection is enabled, all protected segments read '0's (or "garbage values") and are prevented from further programming. All unprotected segments, including ID locations and configuration word, read normally. These locations can be programmed.

4.2 Embedding Configuration Word and ID Information in the Hex File

To allow portability of code, the programmer is required to read the configuration word and ID locations from the hex file when loading the hex file. If configuration word information was not present in the hex file then a simple warning message may be issued. Similarly, while saving a hex file, configuration word and ID information must be included. An option to not include this information may be provided.

Microchip Technology Inc. feels strongly that this feature is important for the benefit of the end customer.

| TABLE 4-1: CONFIGURATION WORD |
| PIC16C554 |

To code protect:
- Protect all memory 0000001000XXXX
- No code protection 1111111011XXXX

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0x2007)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>Protected memory segment</td>
<td>Read All 0's, Write Disabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>ID Locations (0x2000 : 0x2003)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
</tbody>
</table>

| PIC16C556 |

To code protect:
- Protect all memory 0000001000XXXX
- Protect upper 1/2 memory 0101011001XXXX
- No code protection 1111111011XXXX

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0x2007)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>Protected memory segment</td>
<td>Read All 0's, Write Disabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>ID Locations (0x2000 : 0x2003)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
</tbody>
</table>

| PIC16C558 |

To code protect:
- Protect all memory 0000001000XXXX
- Protect upper 3/4 memory 0101011001XXXX
- Protect upper 1/2 memory 1010101010XXXX
- No code protection 1111111011XXXX

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0x2007)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>Protected memory segment</td>
<td>Read All 0's, Write Disabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>ID Locations (0x2000 : 0x2003)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
</tbody>
</table>
4.3 Checksum

4.3.1 CHECKSUM CALCULATIONS

Checksum is calculated by reading the contents of the PIC16C55X memory locations and adding up the opcodes up to the maximum user addressable location, e.g., 0x1FF for the PIC16C74. Any carry bits exceeding 16-bits are neglected. Finally, the configuration word (appropriately masked) is added to the checksum. Checksum computation for each member of the PIC16C55X devices is shown in Table.

The checksum is calculated by summing the following:
- The contents of all program memory locations
- The configuration word, appropriately masked
- Masked ID locations (when applicable)

The least significant 16 bits of this sum is the checksum.

The following table describes how to calculate the checksum for each device. Note that the checksum calculation differs depending on the code protect setting. Since the program memory locations read out differently depending on the code protect setting, the table describes how to manipulate the actual program memory values to simulate the values that would be read from a protected device. When calculating a checksum by reading a device, the entire program memory can simply be read and summed. The configuration word and ID locations can always be read.

Note that some older devices have an additional value added in the checksum. This is to maintain compatibility with older device programmer checksums.

**TABLE 4-2: CHECKSUM COMPUTATION**

<table>
<thead>
<tr>
<th>Device</th>
<th>Code Protect</th>
<th>Checksum*</th>
<th>Blank Value</th>
<th>0x25E6 at 0 and max address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C554</td>
<td>OFF</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F3F</td>
<td>3D3F</td>
<td>090D</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>SUM_ID + CFGW &amp; 0x3F3F</td>
<td>3D4E</td>
<td>091C</td>
</tr>
<tr>
<td>PIC16C556</td>
<td>OFF</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F3F</td>
<td>3B3F</td>
<td>070D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F3F + SUM_ID</td>
<td>4E5E</td>
<td>0013</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F3F + SUM_ID</td>
<td>3B4E</td>
<td>071C</td>
</tr>
<tr>
<td>PIC16C558</td>
<td>OFF</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F3F</td>
<td>373F</td>
<td>030D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F3F + SUM_ID</td>
<td>5D6E</td>
<td>0F23</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F3F + SUM_ID</td>
<td>4A5E</td>
<td>FC13</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F3F + SUM_ID</td>
<td>374E</td>
<td>031C</td>
</tr>
</tbody>
</table>

Legend: CFGW = Configuration Word
SUM[a:b] = [Sum of locations a through b inclusive]
SUM_ID = ID locations masked by 0xF then made into a 16-bit value with ID0 as the most significant nibble.
For example,
ID0 = 0x12, ID1 = 0x37, ID2 = 0x4, ID3 = 0x26, then SUM_ID = 0x2746.

*Checksum = [Sum of all the individual expressions] MODULO [0xFFFF]
+ = Addition
& = Bitwise AND
# 5.0 PROGRAM/VERIFY MODE ELECTRICAL CHARACTERISTICS

## TABLE 5-1: AC/DC CHARACTERISTICS

<table>
<thead>
<tr>
<th>Timing Requirements for Program/Verify Test Mode</th>
</tr>
</thead>
</table>

### Standard Operating Conditions

**Operating Temperature:** +10°C ≤ T ≤ +40°C, unless otherwise stated, (25°C is recommended)

**Operating Voltage:** 4.5V ≤ VDD ≤ 5.5V, unless otherwise stated.

### Parameter No. | Sym. | Characteristic | Min. | Typ. | Max. | Units | Conditions |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PD1</td>
<td>VDDP</td>
<td>Supply voltage during programming</td>
<td>4.75</td>
<td>5.0</td>
<td>5.25</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>PD2</td>
<td>IDD</td>
<td>Supply current (from VDD) during programming</td>
<td>-</td>
<td>-</td>
<td>20</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>PD3</td>
<td>VDDV</td>
<td>Supply voltage during verify</td>
<td>VDDmin</td>
<td>-</td>
<td>VDDmax</td>
<td>V</td>
<td>Note 1</td>
</tr>
<tr>
<td>PD4</td>
<td>VHH1</td>
<td>Voltage on MCLR/VPP during programming</td>
<td>12.75</td>
<td>-</td>
<td>13.25</td>
<td>V</td>
<td>Note 2</td>
</tr>
<tr>
<td>PD5</td>
<td>VHH2</td>
<td>Voltage on MCLR/VPP during verify</td>
<td>VDD + 4.0</td>
<td>-</td>
<td>13.5</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>PD6</td>
<td>IPP</td>
<td>Programming supply current (from VPP)</td>
<td>-</td>
<td>-</td>
<td>50</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>PD9</td>
<td>Vih1</td>
<td>(RB6, RB7) input high level</td>
<td>0.8 VDD</td>
<td>-</td>
<td>-</td>
<td>V</td>
<td>Schmitt Trigger input</td>
</tr>
<tr>
<td>PD8</td>
<td>Vil1</td>
<td>(RB6, RB7) input low level</td>
<td>0.2 VDD</td>
<td>-</td>
<td>-</td>
<td>V</td>
<td>Schmitt Trigger input</td>
</tr>
</tbody>
</table>

### Serial Program Verify

<table>
<thead>
<tr>
<th>Parameter No.</th>
<th>Sym.</th>
<th>Characteristic</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Units</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>TR</td>
<td>MCLR/VPP rise time (VSS to VHH) for test mode entry</td>
<td>-</td>
<td>-</td>
<td>8.0</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>Tf</td>
<td>MCLR Fall time</td>
<td>-</td>
<td>-</td>
<td>8.0</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>Tset1</td>
<td>Data in setup time before clock Ø</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>Thld1</td>
<td>Data in hold time after clock Ø</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>Tdy1</td>
<td>Data input not driven to next clock input (delay required between command/data or command/command)</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>Tdy2</td>
<td>Delay between clock Ø to clock † of next command or data</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td>Tdy3</td>
<td>Clock † to date out valid (during read data)</td>
<td>200</td>
<td>-</td>
<td>-</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td>Thld0</td>
<td>Hold time after MCLR †</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Tpw</td>
<td>Programming Pulse Width</td>
<td>10</td>
<td>100</td>
<td>1000</td>
<td>ms</td>
<td></td>
</tr>
</tbody>
</table>

**Note 1:** Program must be verified at the minimum and maximum VDD limits for the part.

**2:** VihH1 must be greater than VDD + 4.5V to stay in program/verify mode.
FIGURE 5-1: LOAD DATA COMMAND (PROGRAM/VERIFY)

FIGURE 5-2: READ DATA COMMAND (PROGRAM/VERIFY)

FIGURE 5-3: INCREMENT ADDRESS COMMAND (PROGRAM/VERIFY)
This document includes the programming specifications for the following devices:

- PIC16C61
- PIC16C72A
- PIC16CE623
- PIC16C62
- PIC16C73
- PIC16CE624
- PIC16C62A
- PIC16C73A
- PIC16CE625
- PIC16C62B
- PIC16C73B
- PIC16C710
- PIC16C63
- PIC16C74
- PIC16C711
- PIC16C63A
- PIC16C74A
- PIC16C712
- PIC16C64
- PIC16C74B
- PIC16C716
- PIC16C64A
- PIC16C76
- PIC16C745
- PIC16C65
- PIC16C77
- PIC16C765
- PIC16C65A
- PIC16C620
- PIC16C773
- PIC16C65B
- PIC16C620A
- PIC16C774
- PIC16C66
- PIC16C621
- PIC16C923
- PIC16C67
- PIC16C621A
- PIC16C924
- PIC16C69
- PIC16C622
- PIC16C925
- PIC16C72
- PIC16C622A
- PIC16C926

1.0 PROGRAMMING THE PIC16C6XX/7XX/9XX

The PIC16C6XX/7XX/9XX family can be programmed using a serial method. In Serial mode, the PIC16C6XX/7XX/9XX can be programmed while in the user's system. This allows for increased design flexibility. This programming specification applies to PIC16C6XX/7XX/9XX devices in all packages.

1.1 Hardware Requirements

The PIC16C6XX/7XX/9XX requires two programmable power supplies, one for VDD (2.0V to 6.5V recommended) and one for VPP (12V to 14V). Both supplies should have a minimum resolution of 0.25V.

1.2 Programming Mode

The Programming mode for the PIC16C6XX/7XX/9XX allows programming of user program memory, special locations used for ID, and the configuration word for the PIC16C6XX/7XX/9XX.
Pin Diagrams (Con’t)

**PDIP, SOIC, Windowed CERDIP**

**300 mil. SDIP, SOIC, Windowed CERDIP, SSOP**

**18-pin PDIP, SOIC, Windowed CERDIP**

**20-pin SSOP**
PIC16C6XX/7XX/9XX

Pin Diagrams (Con’t)

PLCC, CLCC

PIC16C92X
2.0 PROGRAM MODE ENTRY

2.1 User Program Memory Map

The user memory space extends from 0x0000 to 0x1FFF (8K). Table 2-1 shows actual implementation of program memory in the PIC16C6XX/7XX/9XX family.

<table>
<thead>
<tr>
<th>Device</th>
<th>Program Memory Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C61</td>
<td>0x000 – 0x3FF (1K)</td>
</tr>
<tr>
<td>PIC16C620/620A</td>
<td>0x000 – 0x1FF (0.5K)</td>
</tr>
<tr>
<td>PIC16C621/621A</td>
<td>0x000 – 0x3FF (1K)</td>
</tr>
<tr>
<td>PIC16C622/622A</td>
<td>0x000 – 0x7FF (2K)</td>
</tr>
<tr>
<td>PIC16C626/626A/62B</td>
<td>0x000 – 0x7FF (2K)</td>
</tr>
<tr>
<td>PIC16C63/63A</td>
<td>0x000 – 0xFFF (4K)</td>
</tr>
<tr>
<td>PIC16C64/64A</td>
<td>0x000 – 0xFFF (4K)</td>
</tr>
<tr>
<td>PIC16C65/65A/65B</td>
<td>0x000 – 0xFFF (4K)</td>
</tr>
<tr>
<td>PIC16C6623</td>
<td>0x000 – 0xFFFF (8K)</td>
</tr>
<tr>
<td>PIC16C6624</td>
<td>0x000 – 0xFFFF (8K)</td>
</tr>
<tr>
<td>PIC16C67</td>
<td>0x000 – 0xFFFF (8K)</td>
</tr>
<tr>
<td>PIC16C76</td>
<td>0x000 – 0xFFFF (8K)</td>
</tr>
<tr>
<td>PIC16C77</td>
<td>0x000 – 0xFFFF (8K)</td>
</tr>
<tr>
<td>PIC16C745</td>
<td>0x000 – 0xFFFF (8K)</td>
</tr>
<tr>
<td>PIC16C765</td>
<td>0x000 – 0xFFFF (8K)</td>
</tr>
<tr>
<td>PIC16C773</td>
<td>0x000 – 0xFFFF (8K)</td>
</tr>
<tr>
<td>PIC16C774</td>
<td>0x000 – 0xFFFF (8K)</td>
</tr>
<tr>
<td>PIC16C923/924/925</td>
<td>0x000 – 0xFFFF (8K)</td>
</tr>
<tr>
<td>PIC16C926</td>
<td>0x000 – 0xFFFF (8K)</td>
</tr>
</tbody>
</table>

When the PC reaches the last location of the implemented program memory, it will wrap around and address a location within the physically implemented memory (see Figure 2-1).

Once in configuration memory, the highest bit of the PC stays a ‘1’, thus, always pointing to the configuration memory. The only way to point to user program memory is to reset the part and re-enter Program/Verify mode, as described in Section 2.2.

A user may store identification information (ID) in four ID locations. The ID locations are mapped in [0x2000:0x2003]. It is recommended that the user use only the four Least Significant bits of each ID location. In some devices, the ID locations read-out in a scrambled fashion after code protection is enabled. For these devices, it is recommended that ID location is written as "11111bbb bbbb", where ‘bbb’ is ID information.

In other devices, the ID locations read out normally, even after code protection. To understand how the devices behave, refer to Table 4-1.

To understand the scrambling mechanism after code protection, refer to Section 3.1.
FIGURE 2-1: PROGRAM MEMORY MAPPING

<table>
<thead>
<tr>
<th>ID Location</th>
<th>ID Location</th>
<th>ID Location</th>
<th>ID Location</th>
<th>ID Location</th>
<th>ID Location</th>
<th>ID Location</th>
<th>ID Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>2001h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>2002h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>2003h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>2004h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>2005h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>2006h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>2007h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>Configuration Word</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
2.2 Program/Verify Mode

The Program/Verify mode is entered by holding pins RB6 and RB7 low, while raising MCLR pin from VSS to the appropriate V\textsubscript{IHH} (high voltage). Once in this mode, the user program memory and the configuration memory can be accessed and programmed in serial fashion. The mode of operation is serial, and the memory that is accessed is the user program memory. RB6 is a Schmitt Trigger input in this mode.

The sequence that enters the device into the Programming/Verify mode places all other logic into the RESET state (the MCLR pin was initially at VSS). This means that all I/O are in the RESET state (high impedance inputs).

The RB6 pin is used as a clock input pin, and the RB7 pin is used for entering command bits and data input/output during serial operation. To input a command, the clock pin (RB6) is cycled six times. Each command bit is latched on the falling edge of the clock with the Least Significant bit (LSb) of the command being input first. The data on pin RB7 is required to have a minimum setup and hold time (see AC/DC specs), with respect to the falling edge of the clock.

Commands that have data associated with them (read and load) are specified to have a minimum delay of 1 $\mu$s between the command and the data. After this delay, the clock pin is cycled 16 times, with the first cycle being a START bit and the last cycle being a STOP bit. Data is also input and output LSb first. Therefore, during a read operation, the LSb will be transmitted onto pin RB7 on the rising edge of the second cycle, and during a load operation, the LSb will be latched on the falling edge of the second cycle. A minimum 1 $\mu$s delay is also specified between consecutive commands.

All commands are transmitted LSb first. Data words are also transmitted LSb first. The data is transmitted on the rising edge and latched on the falling edge of the clock. To allow for decoding of commands and reversal of data pin configuration, a time separation of at least 1 $\mu$s is required between a command and a data word (or another command).

The commands that are available are listed in Table 2-2.

2.2.1 PROGRAM/VERIFY OPERATION

After receiving this command, the program counter (PC) will be set to 0x2000. By then applying 16 cycles to the clock pin, the chip will load 14-bits, a “data word” as described above, to be programmed into the configuration memory. A description of the memory mapping schemes for normal operation and Configuration mode operation is shown in Figure 2-1. After the configuration memory is entered, the only way to get back to the user program memory is to exit the Program/Verify test mode by taking MCLR low (VIL).

### TABLE 2-2: COMMAND MAPPING

<table>
<thead>
<tr>
<th>Command</th>
<th>Mapping (MSb ... LSb)</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Configuration</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0, data(14), 0</td>
</tr>
<tr>
<td>Load Data</td>
<td>0 0 0 0 1 0 0 0</td>
<td>0, data(14), 0</td>
</tr>
<tr>
<td>Read Data</td>
<td>0 0 0 0 1 0 0 0</td>
<td>0, data(14), 0</td>
</tr>
<tr>
<td>Increment Address</td>
<td>0 0 0 1 1 0 0 0</td>
<td></td>
</tr>
<tr>
<td>Begin programming</td>
<td>0 0 1 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>End Programming</td>
<td>0 0 1 1 1 0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The clock must be disabled during In-Circuit Serial Programming™.
FIGURE 2-2: PROGRAM FLOW CHART - PIC16C6XX/7XX/9XX PROGRAM MEMORY

Start

Set VDD = VDDP*

Set VPP = Vih1

N = 1

Program Cycle

Read Data Command

Data correct?

Yes 

Apply 3N Additional Program Cycles

N = N + 1    N = # of Program Cycles

No 

No 

All locations done?

Yes 

Verify all locations @ VDDMIN* VPP = Vih1/2

Data correct?

No 

Report verify @ VDDMIN Error

Yes 

Verify all locations @ VDDMAX* VPP = Vih1/2

Data correct?

No 

Report verify @ VDDMAX Error

Yes 

Done

Program Cycle

Load Data Command

Begin Programming Command

Wait 100 µs

End Programming Command

*NDDP = VDD range for programming (typically 4.75V - 5.25V). 
VDDMIN = Minimum Vdd for device operation. 
VDDMAX = Maximum Vdd for device operation.
FIGURE 2-3: PROGRAM FLOW CHART - PIC16C6XX/7XX/9XX CONFIGURATION WORD AND ID LOCATIONS

Start
- Set Vdd = Vddp*
- Set Vpp = Vihh1
- Load Configuration Command
- N = 1

Program ID Loc?
- Yes
  - Program Cycle
  - N = N + 1
  - N = # of Program Cycles
  - Data Correct?
    - No
    - Apply 3N Program Cycles
  - Data Correct?
    - Yes
    - Report ID Configuration Error
      - Yes
      - N > 25
      - Set Vdd = Vddmax
      - Read Data Command
      - Data Correct?
        - No
        - Set Vdd = Vddmin
        - Read Data Command
        - Data Correct?
          - No
          - Set Vpp = Vihh1
          - N = N + 1
          - N = # of Program Cycles
          - Data Correct?
            - Yes
            - Report Program ID/Config. Error
              - Yes
              - Data Correct?
                - No
                - Read Data Command
                - Data Correct?
                  - Yes
                  - Set Vdd = Vddmax
                  - Set Vpp = Vihh2
                  - Yes
                  - Done

Increment Address Command
- Address = 2004
- Yes
- Increment Address Command
- No

Increment Address Command
- Increment Address Command
- Increment Address Command

Set Vdd = Vddp*
Vddp = Vdd range for programming (typically 4.75V - 5.25V).
Vddmin = Minimum Vdd for device operation.
Vddmax = Maximum Vdd for device operation.
2.2.1.2 Load Data

After receiving this command, the chip will load in a 14-bit "data word" when 16 cycles are applied, as described previously. A timing diagram for the load data command is shown in Figure 4-1.

2.2.1.3 Read Data

After receiving this command, the chip will transmit data bits out of the memory currently accessed, starting with the second rising edge of the clock input. The RB7 pin will go into output mode on the second rising clock edge, and it will revert back to input mode (hi-impedance) after the 16th rising edge. A timing diagram of this command is shown in Figure 4-2.

2.2.1.4 Increment Address

The PC is incremented when this command is received. A timing diagram of this command is shown in Figure 4-3.

2.2.1.5 Begin Programming

A load command (load configuration or load data) must be given before every begin programming command. Programming of the appropriate memory (test program memory or user program memory) will begin after this command is received and decoded. Programming should be performed with a series of 100μs programming pulses. A programming pulse is defined as the time between the begin programming command and the end programming command.

2.2.1.6 End Programming

After receiving this command, the chip stops programming the memory (configuration program memory or user program memory) that it was programming at the time.

2.3 Programming Algorithm Requires Variable VDD

The PIC16C6XX/7XX/9XX family uses an intelligent algorithm. The algorithm calls for program verification at VDDMIN as well as VDDMAX. Verification at VDDMIN guarantees a good “erase margin”. Verification at VDDMAX guarantees a good “program margin”.

The actual programming must be done with VDD in the VDDP range (4.75 - 5.25V):

\[
\begin{align*}
V_{DDP} & = V_{CC} \\
V_{DDMIN} & = \text{minimum operating VDD spec for the part.} \\
V_{DDMAX} & = \text{maximum operating VDD spec for the part}
\end{align*}
\]

Programmers must verify the PIC16C6XX/7XX/9XX at its specified VDDMAX and VDDMIN levels. Since Microchip may introduce future versions of the PIC16C6XX/7XX/9XX with a broader VDD range, it is best that these levels are user selectable (defaults are OK).

Note: Any programmer not meeting these requirements may only be classified as “prototype” or “development” programmer, but not a “production” quality programmer.
3.0 CONFIGURATION WORD

The PIC16C6XX/7XX/9XX family members have several configuration bits. For all devices, these are part of the Configuration Word, located at address 2007h. These bits can be programmed (reads '0'), or left unprogrammed (reads '1'), to select various device configurations.

Because the PIC16C6XX/7XX/9XX family spans so many devices, there are a number of different bit configurations possible for the Configuration Word. Registers 3-1 through 3-7 provide details for each of the seven distinct groups. Table 3-1 provides a cross-index of a particular device name to its appropriate Configuration Word listing.

TABLE 3-1: PIC16C6XX/7XX/9XX DEVICES AND THEIR CONFIGURATION WORD REGISTERS

<table>
<thead>
<tr>
<th>Device</th>
<th>Register</th>
<th>Page</th>
<th>Device</th>
<th>Register</th>
<th>Page</th>
<th>Device</th>
<th>Register</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C61</td>
<td>3-1</td>
<td>61</td>
<td>PIC16C72A</td>
<td>3-3</td>
<td>62</td>
<td>PIC16CE623</td>
<td>3-3</td>
<td>62</td>
</tr>
<tr>
<td>PIC16C62</td>
<td>3-2</td>
<td>61</td>
<td>PIC16C73</td>
<td>3-2</td>
<td>61</td>
<td>PIC16CE624</td>
<td>3-3</td>
<td>62</td>
</tr>
<tr>
<td>PIC16C62A</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C73A</td>
<td>3-3</td>
<td>62</td>
<td>PIC16CE625</td>
<td>3-3</td>
<td>62</td>
</tr>
<tr>
<td>PIC16C62B</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C73B</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C710</td>
<td>3-4</td>
<td>63</td>
</tr>
<tr>
<td>PIC16C63</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C74</td>
<td>3-2</td>
<td>61</td>
<td>PIC16C711</td>
<td>3-4</td>
<td>63</td>
</tr>
<tr>
<td>PIC16C63A</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C74A</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C712</td>
<td>3-3</td>
<td>62</td>
</tr>
<tr>
<td>PIC16C64</td>
<td>3-2</td>
<td>61</td>
<td>PIC16C74B</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C716</td>
<td>3-3</td>
<td>62</td>
</tr>
<tr>
<td>PIC16C64A</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C76</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C745</td>
<td>3-6</td>
<td>65</td>
</tr>
<tr>
<td>PIC16C65</td>
<td>3-2</td>
<td>61</td>
<td>PIC16C77</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C765</td>
<td>3-6</td>
<td>65</td>
</tr>
<tr>
<td>PIC16C65A</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C620</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C773</td>
<td>3-5</td>
<td>64</td>
</tr>
<tr>
<td>PIC16C65B</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C620A</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C774</td>
<td>3-5</td>
<td>64</td>
</tr>
<tr>
<td>PIC16C66</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C621</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C923</td>
<td>3-6</td>
<td>65</td>
</tr>
<tr>
<td>PIC16C67</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C621A</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C924</td>
<td>3-6</td>
<td>65</td>
</tr>
<tr>
<td>PIC16C71</td>
<td>3-1</td>
<td>61</td>
<td>PIC16C622</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C925</td>
<td>3-7</td>
<td>66</td>
</tr>
<tr>
<td>PIC16C72</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C622A</td>
<td>3-3</td>
<td>62</td>
<td>PIC16C926</td>
<td>3-7</td>
<td>66</td>
</tr>
</tbody>
</table>

Note: Throughout the PIC16C6XX/7XX/9XX family, two different implementations of the Power-up Timer Enable bit are used. PWRTEN (timer enabled when bit is set to ‘1’) is used on some earlier PIC16C6X and PIC16C7X devices. PWRTEN (timer enabled when bit is set to ‘0’) is used for all other devices. Please carefully note the distinction between these two versions.
REGISTER 3-1: CONFIGURATION WORD FOR PIC16C61/71 (ADDRESS 2007h)

<table>
<thead>
<tr>
<th>bit13—bit0</th>
<th>CP0</th>
<th>PWTREN</th>
<th>WDTEN</th>
<th>F0SC1</th>
<th>F0SC0</th>
</tr>
</thead>
</table>

bit 13-5  Unimplemented: Read as ‘1’

bit 4  **CP0**: Code Protection bit

1 = Code protection off
0 = All memory code protected

bit 3  **PWTREN**: Power-up Timer Enable bit

1 = PWRT enabled
0 = PWRT disabled

bit 2  **WDTEN**: Watchdog Timer Enable bit

1 = WDT enabled
0 = WDT disabled

bit 1-0  **FOSC1:FOSC0**: Oscillator Selection bits

11 = RC oscillator
10 = HS oscillator
01 = XT oscillator
00 = LP oscillator

Note 1: Enabling Brown-out Reset automatically enables Power-up Timer (PWRT), regardless of the value of bit PWTREN. Ensure the Power-up Timer is enabled any time Brown-out Reset is enabled.

REGISTER 3-2: CONFIGURATION WORD FOR PIC16C62/64/65/73/74 (ADDRESS 2007h)

<table>
<thead>
<tr>
<th>bit13—bit0</th>
<th>CP1</th>
<th>CP0</th>
<th>PWTREN</th>
<th>WDTEN</th>
<th>F0SC1</th>
<th>F0SC0</th>
</tr>
</thead>
</table>

bit 13-6  Unimplemented: Read as ‘1’

bit 5-4  **CP<1:0>**: Code Protection bits

11 = Code protection off
10 = Upper 1/2 memory code protected
01 = Upper 3/4 memory code protected
00 = All memory is protected

bit 3  **PWTREN**: Power-up Timer Enable bit(2)

1 = PWRT enabled
0 = PWRT disabled

bit 2  **WDTEN**: Watchdog Timer Enable bit

1 = WDT enabled
0 = WDT disabled

bit 1-0  **FOSC1:FOSC0**: Oscillator Selection bits

11 = RC oscillator
10 = HS oscillator
01 = XT oscillator
00 = LP oscillator

Note 1: Enabling Brown-out Reset automatically enables Power-up Timer (PWRT), regardless of the value of bit PWTREN. Ensure the Power-up Timer is enabled any time Brown-out Reset is enabled.
**REGISTER 3-3: CONFIGURATION WORD FOR:** PIC16C62A/62B/62C/63/63A/64A/65A/65B/66/67
PIC16C72/72A/73A/73B/74A/74B/76/77
PIC16C620/620A/621A/621/622/622A/712/716
PIC16CE623/624/625
(ADDRESS 2007h)

<table>
<thead>
<tr>
<th>CP1</th>
<th>CP0</th>
<th>CP1</th>
<th>CP0</th>
<th>—</th>
<th>BOREN</th>
<th>CP1</th>
<th>CP0</th>
<th>PWTREN</th>
<th>WDTEN</th>
<th>F0SC1</th>
<th>F0SC0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

bit 13-8  **CP<1:0>:** Code Protection bits\(^{(1)}\)
bit 5-4  For all devices EXCEPT PIC16C620, PIC16C621, PIC16CE623 and PIC16CE624:
\[11 = \text{Code protection off}\]
\[10 = \text{Upper 1/2 of program memory code protected}\]
\[01 = \text{Upper 3/4 of program memory code protected}\]
\[00 = \text{All memory is protected}\]
For the PIC16C621 and PIC16CE624:
\[1x = \text{Code protection off}\]
\[01 = \text{Upper 1/2 of program memory code protected}\]
\[00 = \text{All program memory is code protected}\]
For the PIC16C620 and PIC16CE623:
\[1x, 01 = \text{Code protection off}\]
\[00 = \text{All program memory is code protected}\]

bit 7  **Unimplemented:** Read as '1'

bit 6  **BOREN:** Brown-out Reset Enable bit\(^{(2)}\)
\[1 = \text{BOR enabled}\]
\[0 = \text{BOR disabled}\]

bit 3  **PWTREN:** Power-up Timer Enable bit\(^{(2)}\)
\[1 = \text{PWRT disabled}\]
\[0 = \text{PWRT enabled}\]

bit 2  **WDTEN:** Watchdog Timer Enable bit
\[1 = \text{WDT enabled}\]
\[0 = \text{WDT disabled}\]

bit 1-0  **F0SC1:F0SC0:** Oscillator Selection bits
\[11 = \text{RC oscillator}\]
\[10 = \text{HS oscillator}\]
\[01 = \text{XT oscillator}\]
\[00 = \text{LP oscillator}\]

**Note 1:** All of the CP<1:0> bit pairs have to be given the same value to enable the code protection scheme listed.

**2:** Enabling Brown-out Reset automatically enables Power-up Timer (PWRT), regardless of the value of bit PWTREN. Ensure the Power-up Timer is enabled any time Brown-out Reset is enabled.
## REGISTER 3-4: CONFIGURATION WORD, PIC16C710/711 (ADDRESS 2007h)

<table>
<thead>
<tr>
<th>Bit 13-7</th>
<th>Bit 5-4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP0</td>
<td>CP0</td>
<td>BOREN</td>
<td>PWTREN</td>
<td>WDTEN</td>
</tr>
<tr>
<td>F0SC1</td>
<td>F0SC0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**bit 13-7 CP0:** Code Protection bits
- 1 = Code protection off
- 0 = All program memory is code protected, but 00h - 3Fh is writable

**bit 5-4 BOREN:** Brown-out Reset Enable bit
- 1 = BOR enabled
- 0 = BOR disabled

**bit 3 PWTREN:** Power-up Timer Enable bit
- 1 = PWRT disabled
- 0 = PWRT enabled

**bit 2 WDTEN:** Watchdog Timer Enable bit
- 1 = WDT enabled
- 0 = WDT disabled

**bit 1-0 FOSC1:FOSC0:** Oscillator Selection bits
- 11 = RC oscillator
- 10 = HS oscillator
- 01 = XT oscillator
- 00 = LP oscillator

**Note 1:** All of the CP0 bits have to be given the same value to enable the code protection scheme listed.

**Note 2:** Enabling Brown-out Reset automatically enables Power-up Timer (PWRT), regardless of the value of bit PWTREN. Ensure the Power-up Timer is enabled any time Brown-out Reset is enabled.
REGISTER 3-5:  CONFIGURATION WORD, PIC16C773/774 (ADDRESS 2007h)

<table>
<thead>
<tr>
<th>CP1</th>
<th>CP0</th>
<th>BORV1</th>
<th>BORV0</th>
<th>CP1</th>
<th>CP0</th>
<th>---</th>
<th>BOREN</th>
<th>CP1</th>
<th>CP0</th>
<th>PWTRREN</th>
<th>WDTEN</th>
<th>FOSC1</th>
<th>FOSC0</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit13</td>
<td>bit0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **bit 13-7**: $CP<1:0>$: Code Protection bits$^{(1)}$
  - 11 = Code protection off
  - 10 = Upper 1/2 of program memory code protected
  - 01 = Upper 3/4 of program memory code protected
  - 00 = All program memory is code protected

- **bit 9-8**: BORV $<1:0>$: Brown-out Reset Voltage bits
  - 11 = $V_{BOR}$ set to 2.5V
  - 10 = $V_{BOR}$ set to 2.7V
  - 01 = $V_{BOR}$ set to 4.2V
  - 00 = $V_{BOR}$ set to 4.5V

- **bit 6**: BOREN: Brown-out Reset Enable bit$^{(2)}$
  - 1 = BOR enabled
  - 0 = BOR disabled

- **bit 3**: PWTRREN: Power-up Timer Enable bit$^{(2)}$
  - 1 = PWRT disabled
  - 0 = PWRT enabled

- **bit 2**: WDTEN: Watchdog Timer Enable bit
  - 1 = WDT enabled
  - 0 = WDT disabled

- **bit 1-0**: FOSC1:FOSC0: Oscillator Selection bits
  - 11 = RC oscillator
  - 10 = HS oscillator
  - 01 = XT oscillator
  - 00 = LP oscillator

**Note 1**: All of the CP$<1:0>$ bits pairs have to be given the same value to enable the code protection scheme listed.

**Note 2**: Enabling Brown-out Reset automatically enables Power-up Timer (PWRT), regardless of the value of bit PWTRREN. Ensure the Power-up Timer is enabled any time Brown-out Reset is enabled.
## REGISTER 3-6: CONFIGURATION WORD FOR: PIC16C745/765/923/924

(ADDRESS 2007h)

<table>
<thead>
<tr>
<th>Bit</th>
<th>CP&lt;1:0&gt;</th>
<th>PWTREN</th>
<th>WDTEN</th>
<th>FOSC1</th>
<th>FOSC0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 13</td>
<td>Code Protection bits&lt;sup&gt;(1)&lt;/sup&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit 5-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit 7-6</td>
<td>Unimplemented: Read as ‘1’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit 3</td>
<td>PWTREN: Power-up Timer Enable bit&lt;sup&gt;(2)&lt;/sup&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit 2</td>
<td>WDTEN: Watchdog Timer Enable bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit 1-0</td>
<td>FOSC1:FOSC0: Oscillator Selection bits</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**For PIC16C745/765:**
- 11 = E external clock with 4K PLL
- 10 = H HS oscillator with 4K PL enabled
- 01 = EC external clock with CLKOUT on OSC2
- 00 = HS oscillator

**For PIC16C923/924:**
- 11 = RC oscillator
- 10 = HS oscillator
- 01 = XT oscillator
- 00 = LP oscillator

**Note 1:** All of the CP<1:0> bits pairs have to be given the same value to enable the code protection scheme listed.
## REGISTER 3-7: CONFIGURATION WORD FOR PIC16C925/926 (ADDRESS 2007h)

<table>
<thead>
<tr>
<th>bit13</th>
<th>BOREN</th>
<th>CP1</th>
<th>CP0</th>
<th>PWRTEN</th>
<th>WDTEN</th>
<th>F0SC1</th>
<th>F0SC0</th>
<th>bit0</th>
</tr>
</thead>
</table>

**bit 13-7**  Unimplemented: Read as ‘1’

**bit 6**  BOREN: Brown-out Reset Enable bit\(^{(1)}\)

- 1 = BOR enabled
- 0 = BOR disabled

**bit 5-4**  CP<1:0>: Program Memory Code Protection bits

- **For PIC16C926:**
  - 11 = Code protection off
  - 10 = Lower 1/2 of program memory code protected (0000h-0FFFh)
  - 01 = All but last 256 bytes of program memory code protected (0000h-1EFFh)
  - 00 = All memory is protected
- **For PIC16C925:**
  - 11 = Code protection off
  - 10 = Lower 1/2 of program memory code protected (0000h-07FFh)
  - 01 = All but last 256 bytes of program memory code protected (0000h-0EFFh)
  - 00 = All program memory is protected

**Note:**  For PIC16C925, address values of 1000h to 1FFFh wrap around to 0000h to 0FFh.

**bit 3**  PWRTEN: Power-up Timer Enable bit\(^{(1)}\)

- 1 = PWRT disabled
- 0 = PWRT enabled

**bit 2**  WDTEN: Watchdog Timer Enable bit

- 1 = WDT enabled
- 0 = WDT disabled

**bit 1-0**  FOSC1:FOSC0: Oscillator Selection bits

- 11 = RC oscillator
- 10 = HS oscillator
- 01 = XT oscillator
- 00 = LP oscillator

**Note 1:**  Enabling Brown-out Reset automatically enables Power-up Timer (PWRT), regardless of the value of bit PWRTEN. Ensure the Power-up Timer is enabled any time Brown-out Reset is enabled.
3.1 Embedding Configuration Word and ID Information in the HEX File

To allow portability of code, the programmer is required to read the configuration word and ID locations from the HEX file when loading the HEX file. If configuration word information was not present in the HEX file, then a simple warning message may be issued. Similarly, while saving a HEX file, configuration word and ID information must be included. An option to not include this information may be provided.
Microchip Technology Inc. feels strongly that this feature is beneficial to the end customer.

3.2 Checksum

3.2.1 CHECKSUM CALCULATIONS

Checksum is calculated by reading the contents of the PIC16C6XX/7XX/9XX memory locations and adding up the opcodes, up to the maximum user addressable location, e.g., 0x1FF for the PIC16C74. Any carry bits exceeding 16-bits are neglected. Finally, the configuration word (appropriately masked) is added to the checksum. Checksum computation for each member of the PIC16C6XX/7XX/9XX devices is shown in Table 3-2.

The checksum is calculated by summing the following:
- The contents of all program memory locations
- The configuration word, appropriately masked
- Masked ID locations (when applicable)

The Least Significant 16 bits of this sum is the checksum.

The following table describes how to calculate the checksum for each device. Note that the checksum calculation differs depending on the code protect setting. Since the program memory locations read out differently depending on the code protect setting, the table describes how to manipulate the actual program memory values to simulate the values that would be read from a protected device. When calculating a checksum by reading a device, the entire program memory can simply be read and summed. The configuration word and ID locations can always be read.

Note that some older devices have an additional value added in the checksum. This is to maintain compatibility with older device programmer checksums.

Table 3-2: CHECKSUM COMPUTATION

<table>
<thead>
<tr>
<th>Device</th>
<th>Code Protect</th>
<th>Checksum*</th>
<th>Blank Value</th>
<th>0x25E6 at 0 and Max Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C61</td>
<td>OFF</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x001F + 0x3FE0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>SUM_XNOR[7] + [0x000:0x3FF] + (CFGW &amp; 0x001F</td>
<td>0x0060)</td>
<td>0x3BFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0xFC6F</td>
</tr>
<tr>
<td>PIC16C620</td>
<td>OFF</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>SUM_ID + CFGW &amp; 0x3F7F</td>
<td>0x3D7F</td>
<td>0x25E6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0xFC6E</td>
</tr>
<tr>
<td>PIC16C620A</td>
<td>OFF</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>SUM_ID + CFGW &amp; 0x3F7F</td>
<td>0x3D7F</td>
<td>0x25E6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0xFC6E</td>
</tr>
<tr>
<td>PIC16C621</td>
<td>OFF</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x3B7F</td>
<td>0x25E6</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x4EDE</td>
<td>0x0903</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x3BCE</td>
</tr>
<tr>
<td>PIC16C621A</td>
<td>OFF</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x3B7F</td>
<td>0x25E6</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x4EDE</td>
<td>0x0903</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x3BCE</td>
</tr>
</tbody>
</table>

Legend: CFGW = Configuration Word
SUM[a:b] = [Sum of locations a through b inclusive]
SUM_XNOR[7][a:b] = XNOR of the seven high order bits of memory location with the seven low order bits summed over locations a through b inclusive. For example, XNOR(0x3C31) = 0x78 XOR 0xC31 = 0x0036.
SUM_ID = ID locations masked by 0xF then made into a 16-bit value with ID0 as the most significant nibble.

For example:
ID0 = 0x12, ID1 = 0x37, ID2 = 0x4, ID3 = 0x26, then SUM_ID = 0x2746.

*Checksum = [Sum of all the individual expressions] MODULO [0xFFFF]
= Addition
& = Bitwise AND
| = Bitwise OR
### TABLE 3-2: CHECKSUM COMPUTATION (CONTINUED)

<table>
<thead>
<tr>
<th>Device</th>
<th>Code Protect</th>
<th>Checksum*</th>
<th>Blank Value</th>
<th>0x25E6 at 0 and Max Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C622</td>
<td>OFF</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F</td>
<td>0x377F</td>
<td>0x034D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x5DEE</td>
<td>0x0FA3</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x4ADE</td>
<td>0xCF93</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x37CE</td>
<td>0x039C</td>
</tr>
<tr>
<td>PIC16C622A</td>
<td>OFF</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F</td>
<td>0x377F</td>
<td>0x034D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x5DEE</td>
<td>0x0FA3</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x4ADE</td>
<td>0xCF93</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x37CE</td>
<td>0x039C</td>
</tr>
<tr>
<td>PIC16CE623</td>
<td>OFF</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F</td>
<td>0x37DF</td>
<td>0x094D</td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>SUM_ID + CFGW &amp; 0x3F7F</td>
<td>0x3DCE</td>
<td>0x099C</td>
</tr>
<tr>
<td>PIC16CE624</td>
<td>OFF</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F</td>
<td>0x37BF</td>
<td>0x074D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x4ADE</td>
<td>0xF093</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x37CE</td>
<td>0x079C</td>
</tr>
<tr>
<td>PIC16CE625</td>
<td>OFF</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F</td>
<td>0x377F</td>
<td>0x034D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x5DEE</td>
<td>0x0FA3</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x4ADE</td>
<td>0xCF93</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x37CE</td>
<td>0x039C</td>
</tr>
<tr>
<td>PIC16C62</td>
<td>OFF</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x37BF</td>
<td>0x038D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x3FF] + SUM_XNOR7[0x400:0x7FF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x5DEE</td>
<td>0x0FA3</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x1FF] + SUM_XNOR7[0x200:0x7FF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x4ADE</td>
<td>0xCF93</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>SUM_XNOR7[0x000:0x7FF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x37BF</td>
<td>0x0375</td>
</tr>
<tr>
<td>PIC16C62A</td>
<td>OFF</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x003F</td>
<td>0x377F</td>
<td>0x034D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x003F + SUM_ID</td>
<td>0x5DEE</td>
<td>0x0FA3</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x003F + SUM_ID</td>
<td>0x4ADE</td>
<td>0xCF93</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x003F + SUM_ID</td>
<td>0x37CE</td>
<td>0x039C</td>
</tr>
<tr>
<td>PIC16C62B</td>
<td>OFF</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F</td>
<td>0x377F</td>
<td>0x034D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x5DEE</td>
<td>0x0FA3</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x4ADE</td>
<td>0xCF93</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x37CE</td>
<td>0x039C</td>
</tr>
<tr>
<td>PIC16C63</td>
<td>OFF</td>
<td>SUM[0x000:0xFFFF] + CFGW &amp; 0x3F7F</td>
<td>0x2F7F</td>
<td>0xFB4D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x51EE</td>
<td>0x03A3</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x40DE</td>
<td>0xFD93</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x2FCE</td>
<td>0xFB9C</td>
</tr>
<tr>
<td>PIC16C63A</td>
<td>OFF</td>
<td>SUM[0x000:0xFFFF] + CFGW &amp; 0x3F7F</td>
<td>0x2F7F</td>
<td>0xFB4D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x51EE</td>
<td>0x03A3</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x40DE</td>
<td>0xFD93</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x2FCE</td>
<td>0xFB9C</td>
</tr>
<tr>
<td>PIC16C64</td>
<td>OFF</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x37BF</td>
<td>0x038D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x3FF] + SUM_XNOR7[0x400:0x7FF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x37AF</td>
<td>0x1D69</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x1FF] + SUM_XNOR7[0x200:0x7FF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x37BF</td>
<td>0x1D59</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x003F + SUM_ID</td>
<td>0x37BF</td>
<td>0x0375</td>
</tr>
</tbody>
</table>

Legend:  
- CFGW = Configuration Word  
- SUM[a:b] = [Sum of locations a through b inclusive]  
- SUM_XNOR7[a:b] = XNOR of the seven high order bits of memory location with the seven low order bits summed over locations a through b inclusive. For example, XNOR(0x3C31)=0x78 XNOR 0c31 = 0x0036.  
- SUM_ID = ID locations masked by 0xF then made into a 16-bit value with ID0 as the most significant nibble.  
- For example, ID0 = 0x12, ID1 = 0x37, ID2 = 0x4, ID3 = 0x26, then SUM_ID = 0x2746.  
- *Checksum = [Sum of all the individual expressions] MODULO [0xFFFF]  
  + = Addition  
  & = Bitwise AND  
  | = Bitwise OR
### TABLE 3-2: CHECKSUM COMPUTATION (CONTINUED)

<table>
<thead>
<tr>
<th>Device</th>
<th>Code Protect</th>
<th>Checksum*</th>
<th>Blank Value</th>
<th>0x25E6 at 0 and Max Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C64A</td>
<td>OFF</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F</td>
<td>0x377F</td>
<td>0x034D</td>
</tr>
<tr>
<td>1/2</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x5DEE</td>
<td>0x0FA3</td>
<td></td>
</tr>
<tr>
<td>3/4</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x49DE</td>
<td>0xFD93</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x37CE</td>
<td>0x039C</td>
<td></td>
</tr>
<tr>
<td>PIC16C65</td>
<td>OFF</td>
<td>SUM[0x000:0xFFF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x2FBf</td>
<td>0xFB8D</td>
</tr>
<tr>
<td>1/2</td>
<td>SUM[0x000:0x1FF] + SUM_XNOR7[0x800:FFF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x2DFA</td>
<td>0x1569</td>
<td></td>
</tr>
<tr>
<td>3/4</td>
<td>SUM[0x000:0x3FF] + SUM_XNOR7[0x400:FFF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x2F9F</td>
<td>0x1559</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>SUM_XNOR7[0x000:0xFFF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x2F8F</td>
<td>0x2F35</td>
<td></td>
</tr>
<tr>
<td>PIC16C65A</td>
<td>OFF</td>
<td>SUM[0x000:0xFFF] + CFGW &amp; 0x3F7F</td>
<td>0x2F7F</td>
<td>0xFB4D</td>
</tr>
<tr>
<td>1/2</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x51EE</td>
<td>0x03A3</td>
<td></td>
</tr>
<tr>
<td>3/4</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x40DE</td>
<td>0xF293</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x2FCE</td>
<td>0xFB9C</td>
<td></td>
</tr>
<tr>
<td>PIC16C66</td>
<td>OFF</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F</td>
<td>0x1F7F</td>
<td>0xEB4D</td>
</tr>
<tr>
<td>1/2</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x39EE</td>
<td>0x0EA3</td>
<td></td>
</tr>
<tr>
<td>3/4</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x2CDE</td>
<td>0xD93E</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x1FCE</td>
<td>0xEB9C</td>
<td></td>
</tr>
<tr>
<td>PIC16C67</td>
<td>OFF</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F</td>
<td>0x1F7F</td>
<td>0xEB4D</td>
</tr>
<tr>
<td>1/2</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x39EE</td>
<td>0x0EA3</td>
<td></td>
</tr>
<tr>
<td>3/4</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x2CDE</td>
<td>0xD93E</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x1FCE</td>
<td>0xEB9C</td>
<td></td>
</tr>
<tr>
<td>PIC16C710</td>
<td>OFF</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F</td>
<td>0x1F7F</td>
<td>0xEB4D</td>
</tr>
<tr>
<td>ON</td>
<td>SUM[0x000:0x03FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x39EE</td>
<td>0x0EA3</td>
<td></td>
</tr>
<tr>
<td>ON</td>
<td>SUM[0x000:0x01FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x2CDE</td>
<td>0xD93E</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x1FCE</td>
<td>0xEB9C</td>
<td></td>
</tr>
<tr>
<td>PIC16C71</td>
<td>OFF</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x001F + 0x3FE0</td>
<td>0x3BFF</td>
<td>0x07CD</td>
</tr>
<tr>
<td>ON</td>
<td>SUM_XNOR7[0x000:0x3FF] + (CFGW &amp; 0x001F</td>
<td>0x0060)</td>
<td>0xFC6F</td>
<td>0xFC15</td>
</tr>
<tr>
<td>ON</td>
<td>SUM[0x000:0x03FF] + CFGW &amp; 0x3F7F</td>
<td>0x3BFF</td>
<td>0x07CD</td>
<td></td>
</tr>
<tr>
<td>PIC16C711</td>
<td>OFF</td>
<td>SUM[0x000:0x07FF] + CFGW &amp; 0x3F7F</td>
<td>0x37FF</td>
<td>0x034D</td>
</tr>
<tr>
<td>1/2</td>
<td>SUM[0x000:0x03FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x3C0E</td>
<td>0xEDC3</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x37CE</td>
<td>0x039C</td>
<td></td>
</tr>
<tr>
<td>PIC16C716</td>
<td>OFF</td>
<td>SUM[0x000:0x07FF] + CFGW &amp; 0x3F7F</td>
<td>0x37FF</td>
<td>0x034D</td>
</tr>
<tr>
<td>1/2</td>
<td>SUM[0x000:0x03FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x3C0E</td>
<td>0xEDC3</td>
<td></td>
</tr>
<tr>
<td>3/4</td>
<td>SUM[0x000:0x01FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x37CE</td>
<td>0x039C</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x37CE</td>
<td>0x039C</td>
<td></td>
</tr>
<tr>
<td>PIC16C72</td>
<td>OFF</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F</td>
<td>0x37FF</td>
<td>0x034D</td>
</tr>
<tr>
<td>1/2</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x3C0E</td>
<td>0xEDC3</td>
<td></td>
</tr>
<tr>
<td>3/4</td>
<td>SUM[0x000:0x01FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x37CE</td>
<td>0x039C</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x37CE</td>
<td>0x039C</td>
<td></td>
</tr>
</tbody>
</table>

**Legend:**
- CFGW = Configuration Word
- SUM[a:b] = [Sum of locations a through b inclusive]
- SUM_XNOR7[a:b] = XNOR of the seven high order bits of memory location with the seven low order bits summed over locations a through b inclusive. For example, XNOR(0x3C31)=0x78 XNOR 0c31 = 0x0036.
- SUM_ID = ID locations masked by 0xF then made into a 16-bit value with ID0 as the most significant nibble.

For example,
- ID0 = 0x12, ID1 = 0x37, ID2 = 0x4, ID3 = 0x26, then SUM_ID = 0x2746.

*Checksum = [Sum of all the individual expressions] MODULO [0xFFFF]*

+ = Addition
& = Bitwise AND
| = Bitwise OR

© 2003 Microchip Technology Inc.

DS30228K-page 3-69
## Table 3-2: Checksum Computation (Continued)

<table>
<thead>
<tr>
<th>Device</th>
<th>Code Protect</th>
<th>Checksum*</th>
<th>Blank Value</th>
<th>$0x25E6$ at 0 and Max Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C72A</td>
<td>OFF</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F</td>
<td>0x377F</td>
<td>0x034D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x5DEE</td>
<td>0x0FAD</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x4ADE</td>
<td>0xFD93</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x37CE</td>
<td>0x039C</td>
</tr>
<tr>
<td>PIC16C73</td>
<td>OFF</td>
<td>SUM[0x000:0xFFFF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x2FBF</td>
<td>0xFBB8D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x7FF] + SUM_XNOR[0x800:FFF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x2FAD</td>
<td>0x1566</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x3FF] + SUM_XNOR[0x400:FFF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x2F9F</td>
<td>0x1559</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>SUM_XNOR[0x000:0xFFFF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x2F8F</td>
<td>0x2F35</td>
</tr>
<tr>
<td>PIC16C73A</td>
<td>OFF</td>
<td>SUM[0x000:0xFFFF] + CFGW &amp; 0x3F7F</td>
<td>0x2F7F</td>
<td>0xFB4D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x51EE</td>
<td>0x03A3</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x40DE</td>
<td>0xF293</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x2FCE</td>
<td>0xFB9C</td>
</tr>
<tr>
<td>PIC16C73B</td>
<td>OFF</td>
<td>SUM[0x000:0xFFFF] + CFGW &amp; 0x3F7F</td>
<td>0x2F7F</td>
<td>0xFB4D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x51EE</td>
<td>0x03A3</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x40DE</td>
<td>0xF293</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x2FCE</td>
<td>0xFB9C</td>
</tr>
<tr>
<td>PIC16C74</td>
<td>OFF</td>
<td>SUM[0x000:0xFFFF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x2FBF</td>
<td>0xFB8D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x7FF] + SUM_XNOR[0x800:FFF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x2FAD</td>
<td>0x1566</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x3FF] + SUM_XNOR[0x400:FFF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x2F9F</td>
<td>0x1559</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>SUM_XNOR[0x000:0xFFFF] + CFGW &amp; 0x003F + 0x3F80</td>
<td>0x2F8F</td>
<td>0x2F35</td>
</tr>
<tr>
<td>PIC16C74A</td>
<td>OFF</td>
<td>SUM[0x000:0xFFFF] + CFGW &amp; 0x3F7F</td>
<td>0x2F7F</td>
<td>0xFB4D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x51EE</td>
<td>0x03A3</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x40DE</td>
<td>0xF293</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x2FCE</td>
<td>0xFB9C</td>
</tr>
<tr>
<td>PIC16C74B</td>
<td>OFF</td>
<td>SUM[0x000:0xFFFF] + CFGW &amp; 0x3F7F</td>
<td>0x2F7F</td>
<td>0xFB4D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x51EE</td>
<td>0x03A3</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x40DE</td>
<td>0xF293</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x2FCE</td>
<td>0xFB9C</td>
</tr>
<tr>
<td>PIC16C76</td>
<td>OFF</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F</td>
<td>0x1F7F</td>
<td>0xEB4D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x0FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x39EE</td>
<td>0xEA43</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x07FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x2CDE</td>
<td>0xDE93</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x1FCE</td>
<td>0xEB9C</td>
</tr>
<tr>
<td>PIC16C77</td>
<td>OFF</td>
<td>SUM[0x000:0x1FF] + CFGW &amp; 0x3F7F</td>
<td>0x1F7F</td>
<td>0xEB4D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x0FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x39EE</td>
<td>0xEA43</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x07FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x2CDE</td>
<td>0xDE93</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x1FCE</td>
<td>0xEB9C</td>
</tr>
<tr>
<td>PIC16C773</td>
<td>OFF</td>
<td>SUM[0x000:0x0FF] + CFGW &amp; 0x3F7F</td>
<td>0x2F7F</td>
<td>0xFB4D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x07FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x55EE</td>
<td>0x07A3</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x03FF] + CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x48DE</td>
<td>0xFA93</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F7F + SUM_ID</td>
<td>0x3BCE</td>
<td>0x079C</td>
</tr>
</tbody>
</table>

Legend: CFGW = Configuration Word
SUM[a:b] = [Sum of locations a through b inclusive]
SUM_XNOR[7:a:b] = XOR of the seven high order bits of memory location with the seven low order bits summed over locations a through b inclusive. For example, XNOR(0x3C31)=0x78 XNOR [0c31] = 0x0036.

SUM_ID = ID locations masked by 0xF then made into a 16-bit value with ID0 as the most significant nibble.

For example,

ID0 = 0x12, ID1 = 0x37, ID2 = 0x4, ID3 = 0x26, then SUM_ID = 0x2F746.

*Checksum = [Sum of all the individual expressions] MODULO [0xFFFF]
+ = Addition
&& = Bitwise AND
| = Bitwise OR
TABLE 3-2: CHECKSUM COMPUTATION (CONTINUED)

<table>
<thead>
<tr>
<th>Device</th>
<th>Code Protect</th>
<th>Checksum*</th>
<th>Blank Value</th>
<th>0x25E6 at 0 and Max Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16C923</td>
<td>OFF</td>
<td>SUM[0x000:0xFFF] + CFGW &amp; 0x3F3F</td>
<td>0x2F3F</td>
<td>0xFB0D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F3F + SUM_ID</td>
<td>0x516E</td>
<td>0x0323</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F3F + SUM_ID</td>
<td>0x405E</td>
<td>0xF213</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F3F + SUM_ID</td>
<td>0x2F4E</td>
<td>0xFB1C</td>
</tr>
<tr>
<td>PIC16C924</td>
<td>OFF</td>
<td>SUM[0x000:0xFFF] + CFGW &amp; 0x3F3F</td>
<td>0x2F3F</td>
<td>0xFB0D</td>
</tr>
<tr>
<td></td>
<td>1/2</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x3F3F + SUM_ID</td>
<td>0x516E</td>
<td>0x0323</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>SUM[0x000:0x3FF] + CFGW &amp; 0x3F3F + SUM_ID</td>
<td>0x405E</td>
<td>0xF213</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3F3F + SUM_ID</td>
<td>0x2F4E</td>
<td>0xFB1C</td>
</tr>
<tr>
<td>PIC16C745</td>
<td>OFF</td>
<td>SUM(0000:1FFF) + CFGW &amp; 0x3F3F</td>
<td>0x1F3F</td>
<td>0xEB0D</td>
</tr>
<tr>
<td></td>
<td>1000:1FFF</td>
<td>SUM(0000:0FFF) + CFGW &amp; 0x3F3F+SUM_ID</td>
<td>0x396E</td>
<td>0xEB23</td>
</tr>
<tr>
<td></td>
<td>800:1FFF</td>
<td>SUM(0000:07FF) + CFGW &amp; 0x3F3F + SUM_ID</td>
<td>0x2C5E</td>
<td>0xDE13</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW * 0x3F3F + SUM_ID</td>
<td>0x1F4E</td>
<td>0xEB1C</td>
</tr>
<tr>
<td>PIC16C765</td>
<td>OFF</td>
<td>SUM(0000:1FFF) + CFGW &amp; 0x3F3F</td>
<td>0x1F3F</td>
<td>0xEB0D</td>
</tr>
<tr>
<td></td>
<td>1000:1FFF</td>
<td>SUM(0000:0FFF) + CFGW &amp; 0x3F3F+SUM_ID</td>
<td>0x396E</td>
<td>0xEB23</td>
</tr>
<tr>
<td></td>
<td>800:1FFF</td>
<td>SUM(0000:07FF) + CFGW &amp; 0x3F3F + SUM_ID</td>
<td>0x2C5E</td>
<td>0xDE13</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW * 0x3F3F + SUM_ID</td>
<td>0x1F4E</td>
<td>0xEB1C</td>
</tr>
</tbody>
</table>

Legend: CFGW = Configuration Word
SUM[a:b] = [Sum of locations a through b inclusive]
SUM_XNOR7[a:b] = XNOR of the seven high order bits of memory location with the seven low order bits summed over
locations a through b inclusive. For example, XNOR(0x3C31)=0x78 XOR 0c31 = 0x0036.
SUM_ID = ID locations masked by 0xF then made into a 16-bit value with ID0 as the most significant nibble.
For example, ID0 = 0x12, ID1 = 0x37, ID2 = 0x4, ID3 = 0x26, then SUM_ID = 0x2746.
*Checksum = [Sum of all the individual expressions] MODULO [0xFFFF]
+ = Addition
& = Bitwise AND
| = Bitwise OR
4.0 PROGRAM/VERIFY MODE

TABLE 4-1: AC/DC CHARACTERISTICS
TIMING REQUIREMENTS FOR PROGRAM/VERIFY TEST MODE

<table>
<thead>
<tr>
<th>Parameter No.</th>
<th>Sym.</th>
<th>Characteristic</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Units</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD1</td>
<td>VDDP</td>
<td>Supply voltage during programming</td>
<td>4.75</td>
<td>5.0</td>
<td>5.25</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>PD2</td>
<td>IDD</td>
<td>Supply current (from VDD) during programming</td>
<td>–</td>
<td>–</td>
<td>20</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>PD3</td>
<td>VDDV</td>
<td>Supply voltage during verify</td>
<td>VDDMIN</td>
<td>–</td>
<td>VDDMAX</td>
<td>V</td>
<td>(Note 1)</td>
</tr>
<tr>
<td>PD4</td>
<td>VIHH1</td>
<td>Voltage on MCLR/VPP during programming</td>
<td>12.75</td>
<td>–</td>
<td>13.25</td>
<td>V</td>
<td>(Note 2)</td>
</tr>
<tr>
<td>PD5</td>
<td>VIHH2</td>
<td>Voltage on MCLR/VPP during verify</td>
<td>VDD + 4.5</td>
<td>–</td>
<td>13.25</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>PD6</td>
<td>IPP</td>
<td>Programming supply current (from VPP)</td>
<td>–</td>
<td>–</td>
<td>50</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>PD9</td>
<td>VIH</td>
<td>(RB6, RB7) input high level</td>
<td>0.8 VDD</td>
<td>–</td>
<td>–</td>
<td>V</td>
<td>Schmitt Trigger input</td>
</tr>
<tr>
<td>PD8</td>
<td>VIL</td>
<td>(RB6, RB7) input low level</td>
<td>0.2 VDD</td>
<td>–</td>
<td>–</td>
<td>V</td>
<td>Schmitt Trigger input</td>
</tr>
<tr>
<td><strong>Serial Program Verify</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>TR</td>
<td>MCLR/VPP rise time (VSS to VIHH) for Test mode entry</td>
<td>–</td>
<td>–</td>
<td>8.0</td>
<td>µs</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>TF</td>
<td>MCLR fall time</td>
<td>–</td>
<td>–</td>
<td>8.0</td>
<td>µs</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>Tset1</td>
<td>Data in setup time before clock ↓</td>
<td>100</td>
<td>–</td>
<td>–</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>Thld1</td>
<td>Data in hold time after clock ↓</td>
<td>100</td>
<td>–</td>
<td>–</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>Tdly1</td>
<td>Data input not driven to next clock input (delay required between command/data or command/command)</td>
<td>1.0</td>
<td>–</td>
<td>–</td>
<td>µs</td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>Tdly2</td>
<td>Delay between clock ↓ to clock ↑ of next command or data</td>
<td>1.0</td>
<td>–</td>
<td>–</td>
<td>µs</td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td>Tdly3</td>
<td>Clock ↑ to data out valid (during read data)</td>
<td>200</td>
<td>–</td>
<td>–</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td>Thld0</td>
<td>Hold time after MCLR ↑</td>
<td>2</td>
<td>–</td>
<td>–</td>
<td>µs</td>
<td></td>
</tr>
</tbody>
</table>

**Note 1:** Program must be verified at the minimum and maximum VDD limits for the part.
**Note 2:** VIHH must be greater than VDD + 4.5V to stay in Programming/Verify mode.
FIGURE 4-1: LOAD DATA COMMAND (PROGRAM/VERIFY)

FIGURE 4-2: READ DATA COMMAND (PROGRAM/VERIFY)

FIGURE 4-3: INCREMENT ADDRESS COMMAND (PROGRAM/VERIFY)
This document includes the programming specifications for the following devices:

- PIC17C752
- PIC17C756
- PIC17C756A
- PIC17C762
- PIC17C766

1.0 PROGRAMMING THE PIC17C7XX

The PIC17C7XX is programmed using the TABLWT instruction. The table pointer points to the internal EPROM location start. Therefore, a user can program an EPROM location while executing code (even from internal EPROM). This programming specification applies to PIC17C7XX devices in all packages.

For the convenience of a programmer developer, a “program & verify” routine is provided in the on-chip test program memory space. The program resides in ROM and not EPROM, therefore, it is not erasable. The “program/verify” routine allows the user to load any address, program a location, verify a location or increment to the next location. It allows variable programming pulse width.

The PIC17C7XX group of the High End Family has added a feature that allows the serial programming of the device. This is very useful in applications where it is desirable to program the device after it has been manufactured into the users system (In-circuit Serial Programming (ISP)). This allows the product to be shipped just before final test as opposed to before board manufacture. Devices may be serialized to make the product unique, “special” variants of the product may be offered, and code updates are possible. This allows for increased design flexibility.

1.1 Hardware Requirements

Since the PIC17C7XX under programming is actually executing code from “boot ROM,” a clock must be provided to the part. Furthermore, the PIC17C7XX under programming may have any oscillator configuration (EC, XT, LF or RC). Therefore, the external clock driver must be able to overdrive pulldown in RC mode. CMOS drivers are required since the OSC1 input has a Schmitt trigger input with levels (typically) of 0.2 VDD and 0.8 VDD. See the PIC17C7XX data sheet (DS30289) for exact specifications.

The PIC17C7XX requires two programmable power supplies, one for VDD (3.0V to 5.5V recommended) and one for VPP (13 ± 0.25V). Both supplies should have a minimum resolution of 0.25V.

The PIC17C7XX uses an intelligent algorithm. The algorithm calls for program verification at VDDmin as well as VDDmax. Verification at VDDmin guarantees good “erase margin”. Verification at VDDmax guarantees good “program margin.” Three times (3X) additional pulses will increase program margin beyond VDDmax and insure safe operation in user system.

The actual programming must be done with VDD in the VDDP range (Parameter PD1).

\[ VDDP = VDD \text{ range required during programming.} \]
\[ VDDmin. = \text{minimum operating VDD spec. for the part.} \]
\[ VDDmax. = \text{maximum operating VCC spec for the part.} \]

Programmers must verify the PIC17C7XX at its specified VDDmax and VDDmin levels. Since Microchip may introduce future versions of the PIC17C7XX with a broader VDD range, it is best that these levels are user selectable (defaults are ok). Blank checks should be performed at VDDMIN.

Note: Any programmer not meeting these requirements may only be classified as “prototype” or “development” programmer but not a “production” quality programmer.
**FIGURE 1-1: PIC17C752/756/756A/762/766 LCC**

**TABLE 1-1: PIN DESCRIPTIONS (DURING PROGRAMMING IN PARALLEL MODE): PIC17C7XX**

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Pin Name</th>
<th>Pin Type</th>
<th>Pin Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA4:RA0</td>
<td>RA4:RA0</td>
<td>I</td>
<td>Necessary in programming mode</td>
</tr>
<tr>
<td>TEST</td>
<td>TEST</td>
<td>I</td>
<td>Must be set to “high” to enter programming mode</td>
</tr>
<tr>
<td>PORTB&lt;7:0&gt;</td>
<td>DAD15:DAD8</td>
<td>I/O</td>
<td>Address &amp; data: high byte</td>
</tr>
<tr>
<td>PORTC&lt;7:0&gt;</td>
<td>DAD7:DAD0</td>
<td>I/O</td>
<td>Address &amp; data: low byte</td>
</tr>
<tr>
<td>MCLR/VPP</td>
<td>VPP</td>
<td>P</td>
<td>Programming Power</td>
</tr>
<tr>
<td>VDD</td>
<td>VDD</td>
<td>P</td>
<td>Power Supply</td>
</tr>
<tr>
<td>VSS</td>
<td>VSS</td>
<td>P</td>
<td>Ground</td>
</tr>
</tbody>
</table>

Legend: I = Input, O = Output, P = Power
2.0 PARALLEL MODE PROGRAM ENTRY

To execute the programming routine, the user must hold TEST pin high, RA2, RA3 must be low and RA4 must be high (after power-up) while keeping MCLR low and then raise MCLR pin from VIL to VDD or VPP. This will force FFE0h in the program counter and execution will begin at that location (the beginning of the boot code) following reset.

**Note:** The Oscillator must not have 72 OSC clocks while the device MCLR is between VIL and VIH.

All unused pins during programming are in hi-impedance state.

PORTB (RB pins) has internal weak pull-ups which are active during the programming mode. When the TEST pin is high, the Power-up timer (PWRT) and Oscillator Start-up Timers (OST) are disabled.

2.1 Program/Verify Mode

The program/verify mode is intended for full-feature programmers. This mode offers the following capabilities:

a) Load any arbitrary 16-bit address to start program and/or verify at that location.

b) Increment address to program/verify the next location.

c) Allows arbitrary length programming pulse width.

d) Following a "verify" allows option to program the same location or increment and verify the next location.

e) Following a "program" allows options to program the same location again, verify the same location or to increment and verify the next location.

**FIGURE 2-1: PROGRAMMING/VERIFY STATE DIAGRAM**
2.1.1 LOADING NEW ADDRESS

The program allows new address to be loaded right out of reset. A 16-bit address is presented on ports B (high byte) and C (low byte) and the RA1 is pulsed (0Æ1, then 1Æ0). The address is latched on the rising edge of RA1. See timing diagrams for details. After loading an address, the program automatically goes into a “verify cycle.” To load a new address at any time, the PIC17C7XX must be reset and the programming mode re-entered.

2.1.2 VERIFY (OR READ) MODE

“Verify mode” can be entered from “Load address” mode, “program mode” or “verify mode.” In verify mode pulsing RA1 will turn on PORTB and PORTC output drivers and output the 16-bit value from the current location. Pulsing RA1 again will increment location count and be ready for the next verify cycle. Pulsing RA0 will begin a program cycle.

2.1.3 PROGRAM CYCLE

“Program cycle” is entered from “verify cycle” or program cycle” itself. After a verify, pulsing RA0 will begin a program cycle. 16-bit data must be presented on PORTB (high byte) and PORTC (low byte) before RA0 is raised.

The data is sampled 3 Tcy cycles after the rising edge of RA0. Programming continues for the duration of RA0 pulse.

At the end of programming, the user can choose one of three different routes. If RA1 is kept low and RA0 is pulsed again, the same location will be programmed again. This is useful for applying over programming pulses. If RA1 is raised before RA0 falling edge, then a verify cycle is started without address increment. Raising RA1 after RA0 goes low will increment address and begin verify cycle on the next address.

FIGURE 2-2: PIC17C7XX PROGRAM MEMORY MAP
3.0 PARALLEL MODE PROGRAMMING SPECIFICATIONS

FIGURE 3-1: PROGRAMMING ROUTINE FLOWCHART

1. **RESET**
   - RA2 = 0
   - RA3 = 0
   - RA4 = 1

2. MCLR = 1
   - Bport = 0xE1 (hold for 10 Tcy)

3. Present address on ports RB, RC hold Tcy after RA1 changes to 1
   - NO: RA1 = 0
   - YES: RA1 = 1

4. Stop driving address on ports
   - NO: RA1 = 0
   - YES: RA1 = 1

5. RA0 = 1
   - NO: Program 16-bit data
   - YES: Increment Address

6. If programming is desired force portB = MSB of data
   - YES: force portC = LSB of data
   (hold 10 Tcy after RA0 is raised)

7. B and C ports not driven by part
   - YES: RA0 = 1
   - NO: RA1 = 1

8. B port is forced by the part
   - B port = MSB of Data
   - C port = LSB of Data

9. B port tristate, should be forced by user
   - Min RA + high or low = 10 Tcy
FIGURE 3-2: RECOMMENDED PROGRAMMING ALGORITHM FOR USER EPROM

Start

Load new address
Pulse-count = 0

Set VDD = VDDMIN

Verify blank

Pulse Blank Check?

NO

Issue “Blank check fail” error message

YES

Load new data

Set VDD to VDDP

Program using 100 µs pulse increment pulse-count

Verify location for correct date

Pass?

YES

Set VDD = VDDMIN

verify location(s)

NO

Pass?

Set VDD = VDDMIN

verify location

YES

Apply (3x Pulse-count) more 100 µs programming pulses for margin (Over programming)

NO

Pulse-Count >25

Location fails programming issue error message “Unable to programming location”
FIGURE 3-3: RECOMMENDED PROGRAMMING ALGORITHM FOR CONFIGURATION WORDS

1. Start
2. Load new address
   - Pulse-count = 0
3. Set V\text{DD} = V\text{DD}\text{min}
4. Verify blank
5. Pass: Blank check?
   - NO: Issue "blank check fail" error message
   - YES: Load new data
6. Set V\text{DD} = V\text{DD}\text{P}
7. Program using 100 ms pulse increment pulse-count
8. Pulse-count < 100?
   - YES: Set V\text{DD} = V\text{DD}\text{M}
   - NO: Verify location for correct data
9. Pass?
   - NO: Location fails programming, issue error message "Unable to program location"
   - YES: Issue error message "Fail verify @ V\text{DD}\text{min/max}"
10. Set V\text{DD} = V\text{DD}\text{min}
11. Verify location(s)
12. Pass?
   - NO: Programming error: Issue error message "Fail verify @ V\text{DD}\text{min/max}"
   - YES: Start
4.0 SERIAL MODE PROGRAM ENTRY

4.1 Hardware Requirements

Certain design criteria must be taken into account for ISP. Seven pins are required for the interface. These are shown in Table 4-1.

4.2 Serial Program Mode Entry

To place the device into the serial programming test mode, two pins will need to be placed at \( V_{IH} \). These are the TEST pin and the MCLR/V\( PP \) pins. Also, the following sequence of events must occur:

1. The TEST pin is placed at \( V_{IH} \).
2. The MCLR/V\( PP \) pin is placed at \( V_{IH} \).

There is a setup time between step 1 and step 2 that must be meet (See “Electrical Specifications for Serial Programming Mode” on page 97.)

After this sequence the Program Counter is pointing to Program Memory Address 0xFF60. This location is in the Boot ROM. The code initializes the USART/SCI so that it can receive commands. For this the device must be clocked. The device clock source in this mode is the RA1/T0CKI pin. Once the USART/SCI has been initialized, commands may be received. The flow is show in these 3 steps:

1. The device clock source starts.
2. Wait 80 device clocks for Boot ROM code to configure the USART/SCI.
3. Commands may be sent now.

### TABLE 4-1: ISP Interface Pins

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA4/RX/DT</td>
<td>DT</td>
<td>I/O</td>
<td>Serial Data</td>
</tr>
<tr>
<td>RA5/TX/CK</td>
<td>CK</td>
<td>I</td>
<td>Serial Clock</td>
</tr>
<tr>
<td>RA1/T0CKI</td>
<td>OSCI</td>
<td>I</td>
<td>Device Clock Source</td>
</tr>
<tr>
<td>TEST</td>
<td>TEST</td>
<td>I</td>
<td>Test mode selection control input. Force to ( V_{IH} ),</td>
</tr>
<tr>
<td>MCLR/V( PP )</td>
<td>MCLR/V( PP )</td>
<td>P</td>
<td>Programming Power</td>
</tr>
<tr>
<td>V( DD )</td>
<td>V( DD )</td>
<td>P</td>
<td>Power Supply</td>
</tr>
<tr>
<td>V( SS )</td>
<td>V( SS )</td>
<td>P</td>
<td>Ground</td>
</tr>
</tbody>
</table>
### 4.3 Software Commands

This feature is similar to that of the PIC16CXXX mid-range family, but the programming commands have been implemented in the device Boot ROM. The Boot ROM is located in the program memory from 0xFF60 to 0xFFFF. The ISP mode is entered when the TEST pin has a V_{IH} voltage applied. Once in ISP mode, the USART/SCI module is configured as a synchronous slave receiver, and the device waits for a command to be received. The ISP firmware recognizes eight commands. These are shown in Table 4-2.

#### Table 4-2: ISP Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET PROGRAM MEMORY POINTER</td>
<td>0000</td>
</tr>
<tr>
<td>LOAD DATA</td>
<td>0000</td>
</tr>
<tr>
<td>READ DATA</td>
<td>0000</td>
</tr>
<tr>
<td>INCREMENT ADDRESS</td>
<td>0000</td>
</tr>
<tr>
<td>BEGIN PROGRAMMING</td>
<td>0000</td>
</tr>
<tr>
<td>LOAD ADDRESS</td>
<td>0000</td>
</tr>
<tr>
<td>READ ADDRESS</td>
<td>0000</td>
</tr>
<tr>
<td>END PROGRAMMING</td>
<td>0000</td>
</tr>
</tbody>
</table>

#### Figure 4-1: Reset Address Pointer Command (Program/Verify)

#### Figure 4-2: Increment Address Command (Program/Verify)
4.3.3 LOAD ADDRESS

This is used to load the address pointer to the Program Memory with a specific 16-bit value. This is useful when a specific range of locations are to be accessed.

4.3.4 READ ADDRESS

This is used so that the current address in the Program Memory pointer can be determined. This can be used to increase the robustness of the ISP programming (ensure that the Program Memory pointers are still in sync).

FIGURE 4-3: LOAD ADDRESS COMMAND

FIGURE 4-4: READ ADDRESS COMMAND
4.3.5  LOAD DATA

This is used to load the 16-bit data that is to be programmed into the Program Memory location. The Program Memory address may be modified after the data is loaded. This data will not be programmed until a BEGIN PROGRAMMING command is executed.

4.3.6  READ DATA

This is used to read the data in Program Memory that is pointed to by the current address pointer. This is useful for doing a verify of the programming cycle and can be used to determine the number for programming cycles that are required for the 3X overprogramming.

FIGURE 4-5: LOAD DATA COMMAND

FIGURE 4-6: READ DATA COMMAND
4.3.7 BEGIN PROGRAMMING

This is used to program the current 16-bit data (last data sent with LOAD DATA Command) into the Program Memory at the address specified by the current address pointer. The programming cycle time is specified by specification P10. After this time has elapsed, any command must be sent, which wakes the processor from the Long Write cycle. This command will be the next executed command.

4.3.8 3X OVERPROGRAMMING

Once a location has been both programmed and verified over a range of voltages, 3X overprogramming should be applied. In other words, apply three times the number of programming pulses that were required to program a location in memory, to ensure a solid programming margin.

This means that every location will be programmed a minimum of 4 times (1 + 3X overprogramming).
FIGURE 4-8: RECOMMENDED PROGRAMMING FLOWCHART

START

TEST = Vihh

MCLR = Vihh

TEST = Vihh

MCLR = Vihh

Start Device Clock (on RA0),
Wait 80 Device Clocks

ISP Command
RESET ADDRESS

N = 1

ISP Command
LOAD ADDRESS

ISP Command
LOAD DATA

ISP Command
BEGIN PROGRAMMING

Wait approx 100 ms

ISP Command
READ DATA

Data Correct?

No

Yes

N = N + 1

N = 3N

ISP Command
BEGIN PROGRAMMING

Wait approx 100 ms

N = N - 1

N = 0?

No

Programmed all
required locations?

Yes

Data Correct?

No

Yes

Verify all Locations
@ Vddmin

Verify all Locations
@ Vddmax

Yes

No

Report Verify
Error
@ Vddmin

Report Verify
Error
@ Vddmax

DONE

4.75V < Vdd < 5.25V

Start Device Clock (on RA0),
TEST = MCLR = RA4 = RA5 = Vss

Verify all Locations
@ Vddmin

Verify all Locations
@ Vddmax

Report Programming
Failure

Test

N > 25?

Yes

No
5.0 CONFIGURATION WORD

Configuration bits are mapped into program memory. Each bit is assigned one memory location. In erased condition, a bit will read as ‘1’. To program a bit, the user needs to write to the memory address. The data is immaterial; the very act of writing will program the bit. The configuration word locations are shown in Table 5-3. The programmer should not program the reserved locations to avoid unpredictable results and to be compatible with future variations of the PIC17C7XX. It is also mandatory that configuration locations are programmed in the strict order starting from the first location (0xFE00) and ending with the last (0xFE0F). Unpredictable results may occur if the sequence is violated.

5.1 Reading Configuration Word

The PIC17C7XX has seven configuration locations (Table 5-1). These locations can be programmed (read as ‘0’) or left unprogrammed (read as ‘1’) to select various device configurations. Any write to a configuration location, regardless of the data, will program that configuration bit. Reading any configuration location between 0xFE00 and 0xFE07 will place the low byte of the configuration word (Table 5-2) into DAD<7:0> (PORTC). DAD<15:8> (PORTD) will be set to 0xFF. Reading a configuration location between 0xFE08 and 0xFE0F will place the high byte of the configuration word into DAD<7:0> (PORTC). DAD<15:8> (PORTD) will be set to 0xFF.

<table>
<thead>
<tr>
<th>TABLE 5-1: CONFIGURATION BIT PROGRAMMING LOCATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit Address</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>FOSC0</td>
</tr>
<tr>
<td>FOSC1</td>
</tr>
<tr>
<td>WDTPS0</td>
</tr>
<tr>
<td>WDTPS1</td>
</tr>
<tr>
<td>PM0</td>
</tr>
<tr>
<td>PM1</td>
</tr>
<tr>
<td>PM2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE 5-2: READ MAPPING OF CONFIGURATION BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>PM2 BODEN PM2 PM2 PM2 PM2 PM2 PM2 PM2 PM2 PM2</td>
</tr>
</tbody>
</table>

—=Unused

PM<2:0>, Processor Mode Select bits
111 = Microprocessor mode
110 = Microcontroller mode
101 = Extended Microcontroller mode
000 = Code protected microcontroller mode

BODEN, Brown-out Detect Enable
1 = Brown-out Detect Circuitry enabled
0 = Brown-out Detect Circuitry disabled

WDTPS1:WDTPS0, WDT Prescaler Select bits.
11 = WDT enabled, postscaler = 1
10 = WDT enabled, postscaler = 256
01 = WDT enabled, postscaler = 64
00 = WDT disabled, 16-bit overflow timer

FOSC1:FOSC0, Oscillator Select bits
11 = EC oscillator
10 = XT oscillator
01 = RC oscillator
00 = LF oscillator
5.2 Embedding Configuration Word Information in the Hex File

To allow portability of code, a PIC17C7XX programmer is required to read the configuration word locations from the hex file when loading the hex file. If the configuration word information was not present in the hex file, then a simple warning message may be issued. Similarly, while saving a hex file, all configuration word information must be included. An option to not include the configuration word information may be provided. When embedding configuration word information in the hex file, it should be to address FE00h.

Microchip Technology Inc. feels strongly that this feature is important for the benefit of the end customer.

5.3 Reading From and Writing To a Code Protected Device

When a device is code-protected, writing to program memory is disabled. If program memory is read, the value returned is the XNOR8 result of the actual program memory word. The XNOR8 result is the upper eight bits of the program memory word XNOR’d with the lower eight bits of the same word. This 8-bit result is then duplicated into both the upper and lower 8-bits of the read value. The configuration word can always be read and written.
5.4 CHECKSUM COMPUTATION

The checksum is calculated by summing the following:
- The contents of all program memory locations
- The configuration word, appropriately masked
- Masked ID locations (when applicable)

The least significant 16 bits of this sum is the checksum.

Table 5-3 describes how to calculate the checksum for each device. Note that the checksum calculation differs depending on the code protect setting. Since the program memory locations read out differently, depending on the code protect setting, the table describes how to manipulate the actual program memory values to simulate the values that would be read from a protected device. When calculating a checksum by reading a device, the entire program memory can simply be read and summed. The configuration word and ID locations can always be read.

**Note:** Some older devices have an additional value added in the checksum. This is to maintain compatibility with older device programmer checksums.

### TABLE 5-3: CHECKSUM COMPUTATION

<table>
<thead>
<tr>
<th>Device</th>
<th>Code Protect</th>
<th>Checksum*</th>
<th>Blank Value</th>
<th>0xC0DE at 0 and max address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC17C752</td>
<td>MP mode</td>
<td>SUM[0x0000:0x1FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0xA05F</td>
<td>0x221D</td>
</tr>
<tr>
<td></td>
<td>MC mode</td>
<td>SUM[0x0000:0x1FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0xA04F</td>
<td>0x220D</td>
</tr>
<tr>
<td></td>
<td>EMC mode</td>
<td>SUM[0x0000:0x1FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0xA01F</td>
<td>0x211D</td>
</tr>
<tr>
<td></td>
<td>PMC mode</td>
<td>SUM_XNOR8[0x0000:0x1FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0x200F</td>
<td>0xE3D3</td>
</tr>
<tr>
<td>PIC17C756</td>
<td>MP mode</td>
<td>SUM[0x0000:0x3FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0xA05F</td>
<td>0x180D</td>
</tr>
<tr>
<td></td>
<td>MC mode</td>
<td>SUM[0x0000:0x3FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0xA04F</td>
<td>0x180D</td>
</tr>
<tr>
<td></td>
<td>EMC mode</td>
<td>SUM[0x0000:0x3FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0xA01F</td>
<td>0x180D</td>
</tr>
<tr>
<td></td>
<td>PMC mode</td>
<td>SUM_XNOR8[0x0000:0x3FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0x180F</td>
<td>0xC3D3</td>
</tr>
<tr>
<td>PIC17C756A</td>
<td>MP mode</td>
<td>SUM[0x0000:0x3FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0xA05F</td>
<td>0x180D</td>
</tr>
<tr>
<td></td>
<td>MC mode</td>
<td>SUM[0x0000:0x3FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0xA04F</td>
<td>0x180D</td>
</tr>
<tr>
<td></td>
<td>EMC mode</td>
<td>SUM[0x0000:0x3FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0xA01F</td>
<td>0x180D</td>
</tr>
<tr>
<td></td>
<td>PMC mode</td>
<td>SUM_XNOR8[0x0000:0x3FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0x180F</td>
<td>0xC3D3</td>
</tr>
<tr>
<td>PIC17C762</td>
<td>MP mode</td>
<td>SUM[0x0000:0x1FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0xA05F</td>
<td>0x221D</td>
</tr>
<tr>
<td></td>
<td>MC mode</td>
<td>SUM[0x0000:0x1FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0xA04F</td>
<td>0x220D</td>
</tr>
<tr>
<td></td>
<td>EMC mode</td>
<td>SUM[0x0000:0x1FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0xA01F</td>
<td>0x211D</td>
</tr>
<tr>
<td></td>
<td>PMC mode</td>
<td>SUM_XNOR8[0x0000:0x1FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0x200F</td>
<td>0xE3D3</td>
</tr>
<tr>
<td>PIC17C766</td>
<td>MP mode</td>
<td>SUM[0x0000:0x3FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0xA05F</td>
<td>0x221D</td>
</tr>
<tr>
<td></td>
<td>MC mode</td>
<td>SUM[0x0000:0x3FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0xA04F</td>
<td>0x220D</td>
</tr>
<tr>
<td></td>
<td>EMC mode</td>
<td>SUM[0x0000:0x3FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0xA01F</td>
<td>0x211D</td>
</tr>
<tr>
<td></td>
<td>PMC mode</td>
<td>SUM_XNOR8[0x0000:0x3FFF] + (CONFIG &amp; 0xC05F)</td>
<td>0x200F</td>
<td>0xE3D3</td>
</tr>
</tbody>
</table>

Legend: CFGW = Configuration Word
SUM[a:b] = [Sum of locations a to b inclusive]
SUM_XNOR8(a:b) = [Sum of 8-bit wide XOR copied into upper and lower byte, of locations a to b inclusive]
*Checksum = [Sum of all the individual expressions] MODULO [0xFFFF]
+ = Addition
& = Bitwise AND
5.5 Device ID Register

Program memory location FDFFh is preprogrammed during the fabrication process with information on the device and revision information. These bits are accessed by a TABLR0 instruction, and are access when the TEST pin is high. As a result, the device ID bits can be read when the part is code protected.

TABLE 5-4: DEVICE ID REGISTER DECODE

<table>
<thead>
<tr>
<th>Device</th>
<th>Device ID Value</th>
<th>Resultant Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC17C766</td>
<td>0000 0010 001</td>
<td>X XXXX</td>
</tr>
<tr>
<td>PIC17C762</td>
<td>0000 0010 101</td>
<td>X XXXX</td>
</tr>
<tr>
<td>PIC17C756</td>
<td>0000 0010 001</td>
<td>X XXXX</td>
</tr>
<tr>
<td>PIC17C756A</td>
<td>0000 0010 001</td>
<td>X XXXX</td>
</tr>
<tr>
<td>PIC17C752</td>
<td>0000 0010 101</td>
<td>X XXXX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device</th>
<th>Device ID Value</th>
<th>Resultant Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC17C766</td>
<td>0000 0001 001</td>
<td>X XXXX</td>
</tr>
<tr>
<td>PIC17C762</td>
<td>0000 0001 101</td>
<td>X XXXX</td>
</tr>
<tr>
<td>PIC17C756</td>
<td>0000 0000 001</td>
<td>X XXXX</td>
</tr>
<tr>
<td>PIC17C756A</td>
<td>0000 0010 001</td>
<td>X XXXX</td>
</tr>
<tr>
<td>PIC17C752</td>
<td>0000 0010 101</td>
<td>X XXXX</td>
</tr>
</tbody>
</table>
# 6.0 PARALLEL MODE AC/DC CHARACTERISTICS AND TIMING

## REQUIREMENTS FOR PROGRAM/VERIFY TEST MODE

### Standard Operating Conditions

- **Operating Temperature:** \(+10\times C \leq T \leq +70\times C\), unless otherwise stated, (25\times C is recommended)
- **Operating Voltage:** \(4.5V \leq \text{VDD} \leq 5.25V\), unless otherwise stated.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PD1</td>
<td>VDDP</td>
<td>Supply voltage during programming</td>
<td>4.75</td>
<td>5.0</td>
<td>5.25</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>PD2</td>
<td>IDDP</td>
<td>Supply current during programming</td>
<td>—</td>
<td>—</td>
<td>50</td>
<td>mA</td>
<td>Freq = 10MHz, VDD = 5.5V</td>
</tr>
<tr>
<td>PD3</td>
<td>VDDV</td>
<td>Supply voltage during verify</td>
<td>VDD min.</td>
<td>—</td>
<td>VDD max.</td>
<td>V</td>
<td>Note 2</td>
</tr>
<tr>
<td>PD4</td>
<td>VPP</td>
<td>Voltage on VPP/MCLR pin during programming</td>
<td>12.75</td>
<td>—</td>
<td>13.25</td>
<td>V</td>
<td>Note 1</td>
</tr>
<tr>
<td>PD6</td>
<td>IPP</td>
<td>Programming current on VPP/MCLR pin</td>
<td>—</td>
<td>25</td>
<td>50</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>FOSCP</td>
<td>Osc/clockin frequency during programming</td>
<td>4</td>
<td>—</td>
<td>10</td>
<td>MHz</td>
<td>TY = 4/FOSCP</td>
</tr>
<tr>
<td>P2</td>
<td>TCY</td>
<td>Instruction cycle</td>
<td>1</td>
<td>—</td>
<td>0.4</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>TIRV2TSH</td>
<td>RA0, RA1, RA2, RA3, RA4 setup before TEST;</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>TSH2mCH</td>
<td>TEST; to MCL;</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>TCVC2V2nCH</td>
<td>RC7:RC0, RB7:RB0 valid to RA1 or RA0; Address/Data input setup time</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>TIRH2BCL</td>
<td>RA1 or RA0; to RB7:RB0, RC7:RC0 invalid; Address data hold time;</td>
<td>10 TCY</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td>T0CKI2BCZ</td>
<td>RTØ to RB7:RB0, RC7:RC0 invalid; Address data hold time;</td>
<td>—</td>
<td>—</td>
<td>8TCY</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td>T0CKH2BCZ</td>
<td>RA1; to data out valid</td>
<td>—</td>
<td>—</td>
<td>10 TCY</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P9</td>
<td>TPROG</td>
<td>Programming pulse width</td>
<td>100</td>
<td>—</td>
<td>1000</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P10</td>
<td>TIRH2IRL</td>
<td>RA0, RA1 high pulse width</td>
<td>10 TCY</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P11</td>
<td>TIRL2IRH</td>
<td>RA0, RA1 low pulse width</td>
<td>10 TCY</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P12</td>
<td>T0CKI2INL</td>
<td>RA1; before INTØ (to go from prog cycle to verify w/o increment)</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P13</td>
<td>TIN2RTL</td>
<td>RA1 valid after RA0 (to select increment or no increment going from program to verify cycle)</td>
<td>10 TCY</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P14</td>
<td>TVPSS</td>
<td>VPP setup time before RA0;</td>
<td>100</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td>Note 1</td>
</tr>
<tr>
<td>P15</td>
<td>TVPPh</td>
<td>VPP hold time after INTØ</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td>Note 1</td>
</tr>
<tr>
<td>P16</td>
<td>TVDV2TSH</td>
<td>VDD stable to TEST;</td>
<td>10</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P17</td>
<td>TRBV2mCH</td>
<td>RB input (E1h) valid to VPP/MCLR;</td>
<td>0</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P18</td>
<td>TMCH2RBH</td>
<td>RB input (E1h) hold after VPP/MCLR;</td>
<td>10TCY</td>
<td>—</td>
<td>—</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P19</td>
<td>TVP2LD</td>
<td>VDD power down after VPP power down</td>
<td>10</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td></td>
</tr>
</tbody>
</table>

**Note 1:** VPP/MCLR pin must only be equal to or greater than VDD at times other than programming.

**2:** Program must be verified at the minimum and maximum VDD limits for the part.
Figure 6-1: Parallel Mode Programming and Verify Timings I

- **Test MCLR**: CELK
- **RA1**: 13V
- **RA0**: 5V
- **RB<7:0>**: INC ADDR
- **RC<7:0>**: E1H ADDR_HI DATA_HI OUT DATA_HI OUT DDATA_HI OUT
- **P4**: ADDR_LO DATA_LO OUT DATA_LO OUT DATA_LO OUT
- **P15**: ADDR_HI OUT
- **P14**: DATA_HI OUT
- **P9**: DATA_LO OUT
- **P5**: DATA_LO OUT
- **P6**: DATA_HI OUT
- **P7**: DATA_HI OUT
- **Jump Address**: Load Address X
- **Programming Mode Entry**: Higher RC<7:0> by pulsing RA1
- **Program location X + 1**: Increment Address to X + 1
- **Verify location X + 1**: Program location X + 1
- **Verify location X + 1**: Do not increment PC by raising RA1 before RA0

**Note:**
- RA2 = 0
- RA3 = 0
- RA4 = 1
FIGURE 6-2: PARALLEL MODE PROGRAMMING AND VERIFY TIMINGS II

Test

13V

Vpp-MCLR

RA1

RA0

RB<7:0>

RC<7:0>

E1H ADDR_HI DATA_HI OUT DATA_HI_IN DATA_HI_IN

ADDR_LO DATA_LO OUT DATA_LO_IN DATA_LO_IN

Jump Address

Input

Programming mode entry

Load address X

Note:
RA2 = 0
RA3 = 0
RA4 = 1
FIGURE 6-3: PARALLEL MODE PROGRAMMING AND VERIFY TIMINGS III

Note:
Device in PGM mode
Test = +6V
VPP/MCLR = VPP
RA2 = 0
RA3 = 0
RA4 = 1

RA1 RA0 RB<7:0> RC<7:0> INC PC DATA_HI IN DATA_HI OUT DATA_LO IN DATA_LO OUT
Verify location X
Do not increment PC before raising RA0 to do this
Program location X
Raise RA1 after RA0 to increment location X + 1
Verify location X + 1
Pulse RA1 to increment address to X + 2
Verify location X + 2
FIGURE 6-4: POWER-UP/DOWN SEQUENCE FOR PROGRAMMING
# 7.0 ELECTRICAL SPECIFICATIONS FOR SERIAL PROGRAMMING MODE

All parameters apply across the specified operating ranges unless otherwise noted. Vcc = 2.5V to 5.5V

<table>
<thead>
<tr>
<th>Parameter No.</th>
<th>Sym</th>
<th>Characteristic</th>
<th>Min</th>
<th>Typ†</th>
<th>Max</th>
<th>Units</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Vihh</td>
<td>Programming Voltage on VPP/ MCLR pin and TEST pin.</td>
<td>12.75</td>
<td>—</td>
<td>13.75</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IPP</td>
<td>Programming current on MCLR pin</td>
<td>—</td>
<td>25</td>
<td>50</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fosc</td>
<td>Input OSC frequency on RA1</td>
<td>—</td>
<td>—</td>
<td>8</td>
<td>MHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tcy</td>
<td>Instruction Cycle Time</td>
<td>—</td>
<td>4/Fosc</td>
<td>—</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS1</td>
<td>Tvih2vih</td>
<td>Setup time between TEST = Vihh and MCLR = Vihh</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>PS2</td>
<td>Tser</td>
<td>Serial setup time</td>
<td>20</td>
<td>—</td>
<td>—</td>
<td>Tcy</td>
<td></td>
</tr>
<tr>
<td>PS3</td>
<td>Tsclk</td>
<td>Serial Clock period</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>Tcy</td>
<td></td>
</tr>
<tr>
<td>PS4</td>
<td>Tset1</td>
<td>Input Data Setup Time to serial clock Ø</td>
<td>15</td>
<td>—</td>
<td>—</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>PS5</td>
<td>Thld1</td>
<td>Input Data Hold Time from serial clock Ø</td>
<td>15</td>
<td>—</td>
<td>—</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>PS6</td>
<td>Tdly1</td>
<td>Delay between last clock Ø to first clock † of next command</td>
<td>20</td>
<td>—</td>
<td>—</td>
<td>Tcy</td>
<td></td>
</tr>
<tr>
<td>PS7</td>
<td>Tdly2</td>
<td>Delay between last clock Ø of command byte to first clock † of read of data word</td>
<td>20</td>
<td>—</td>
<td>—</td>
<td>Tcy</td>
<td></td>
</tr>
<tr>
<td>PS8</td>
<td>Tdly3</td>
<td>Delay between last clock Ø of command byte to first clock † of write of data word</td>
<td>30</td>
<td>—</td>
<td>—</td>
<td>Tcy</td>
<td></td>
</tr>
<tr>
<td>PS9</td>
<td>Tdly4</td>
<td>Data input not driven to next clock input</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>Tcy</td>
<td></td>
</tr>
<tr>
<td>PS10</td>
<td>Tdly5</td>
<td>Delay between last begin programming clock Ø to last clock Ø of next command (minimum programming time)</td>
<td>100</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td></td>
</tr>
</tbody>
</table>

* These parameters are characterized but not tested.
† Data in "Typ" column is at 5V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.
FIGURE 7-1: RESET ADDRESS POINTER COMMAND (PROGRAM/VERIFY)

FIGURE 7-2: INCREMENT ADDRESS COMMAND (PROGRAM/VERIFY)
FIGURE 7-3: LOAD ADDRESS COMMAND

FIGURE 7-4: READ ADDRESS COMMAND
FIGURE 7-5: LOAD DATA COMMAND

RA1/T0CKI
Test

MCLR/Vpp

RA5 (Clock)

RA4 (Data)

Reset

RA4 = Input

Program/Verify Test Mode

FIGURE 7-6: READ DATA COMMAND

RA1/T0CKI
Test

MCLR/Vpp

RA5 (Clock)

RA4 (Data)

Reset

RA4 = Input

RA4 = Output

Program/Verify Test Mode

FIGURE 7-7: BEGIN PROGRAMMING COMMAND (PROGRAM)

RA1/T0CKI
Test

MCLR/Vpp

RA5 (Clock)

RA4 (Data)

Reset

RA4 = Input

Program/Verify Test Mode
This document includes the programming specifications for the following devices:

- PIC18C242
- PIC18C252
- PIC18C442
- PIC18C452
- PIC18C601
- PIC18C801
- PIC18C658
- PIC18C858

### 1.0 PROGRAMMING THE PIC18CXXX

The PIC18CXXX can be programmed using a serial method while in users’ system, allowing increased design flexibility. This programming specification applies to PIC18CXXX devices in all package types.

#### 1.1 Hardware Requirements

The PIC18CXXX requires two programmable power supplies, one for \( V_{DD} \) and one for \( V_{PP} \). Both supplies should have a minimum resolution of 0.25V.

#### 1.2 Programming Mode

The Programming mode for the PIC18CXXX allows programming of user program memory (except for the PIC18C601/801 ROMless devices), special locations used for ID, and the configuration words for the PIC18CXXX.

### Pin Diagrams

The pin diagrams for the PIC18CXX2 family are shown below in Figure 1-1 through Figure 1-3. Pin diagrams for the PIC18CXXX family are provided in Figure 1-4 through Figure 1-7. Pin diagrams for the PIC18C601/801 family are provided in Figure 1-8 through Figure 1-11.

#### FIGURE 1-1: PIC18CXX2 FAMILY PIN DIAGRAM

| Pin Name | During Programming |
|----------|-------------------|---|
| MCLR/VPP | VPP | P | Programming Power |
| VDD | VDD | P | Power Supply |
| VSS | VSS | P | Ground |
| RB6 | RB6 | I | Serial Clock |
| RB7 | RB7 | I/O | Serial Data |

Legend: I = Input, O = Output, P = Power
* RB3 is the alternate pin for the CCP2 pin multiplexing.

Note: Pin compatible with 44-pin PIC16C7X devices.
* RB3 is the alternate pin for the CCP2 pin multiplexing.

Note: Pin compatible with 28-pin PIC16C7X devices.
FIGURE 1-4: PIC18C658 64-PIN TQFP DIAGRAM

Note: All PIC18C658 and PIC18C858 package outlines are compatible with PIC17C7XX.
FIGURE 1-5: PIC18C658 68-PIN PLCC DIAGRAM

Note: All PIC18C658 and PIC18C858 package outlines are compatible with PIC17C7XX.
FIGURE 1-6: PIC18C858 80-PIN TQFP DIAGRAM

Note: All PIC18C658 and PIC18C858 package outlines are compatible with PIC17C7XX.
FIGURE 1-7: PIC18C858 84-PIN PLCC DIAGRAM

Note: All PIC18C658 and PIC18C858 package outlines are compatible with PIC17C7XX.
FIGURE 1-8: PIC18C601 64-PIN TQFP DIAGRAM

 PIC18C601

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
RE1/AD9 RE0/AD8 RG0/ALE RG1/OE RG2/WRL RG3/WRH MCLR/Vpp RG4/BA0 Vss Vdd
17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
RF7/UB RF6/LB RF5/CST RF4/A16 RF3/CSI0 RF2/AN7
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
RB0/INT0 RB1/INT1 RB2/INT2 RB3/CCP2 RB4 RB5 RB6 Vss OSC2/CLKO OSC1/CLKI Vdd RB7 RC5/SDO RC4/SDI/SDA RC3/SCK/SCL RC2/CCP1
49
FIGURE 1-9: PIC18C601 68-PIN PLCC DIAGRAM
FIGURE 1-10: PIC18C801 80-PIN TQFP DIAGRAM
2.0 IN-CIRCUIT SERIAL PROGRAMMING™ (ICSP™) MODE

2.1 Introduction

Serial Programming mode is entered by asserting MCLR/VPP = VᵢH and RB6, RB7 = 0V.

Instructions are fed into the CPU serially on RB7, and are shifted on the rising edge, and latched on the falling edge of the serial clock presented on RB6. RB7 serves as data out, as well. Programming and verification are performed by executing TBLRD and TBLWT instructions. The address pointer to the program memory is simply the table pointer. The address pointer can be incremented and decremented by executing table reads and writes with auto-decrement and auto-increment.

2.2 ICSP Operation

In ICSP mode, instruction execution takes place through a serial interface using RB6 and RB7. RB7 is used to shift in instructions and shift out data from the TABLAT register. RB6 is used as the serial shift clock and the CPU execution clock. Instructions and data are shifted LSb first.

In this mode, all instructions are shifted serially, loaded into the instruction register, and executed. No program fetching occurs from internal or external program memory. 8-bit data bytes are read from the TABLAT register via the same serial interface.

2.2.1 4-BIT SERIAL INSTRUCTIONS

A set of 4-bit instructions are provided for ICSP mode, so the most common instructions used for ICSP can be fetched quickly, and reduce the amount of time required to program a device. The 4-bit opcode is shifted in while the previously fetched instruction executes. The 4-bit instruction contains the lower 4 bits of an instruction opcode. The upper 12 bits default to all 0’s. Instructions with all 0’s in the upper byte of the instruction word are by default, considered special instructions. The serial instructions are decoded as shown in Table 2-1.

<table>
<thead>
<tr>
<th>Mnemonic, Operands</th>
<th>Description</th>
<th>Cycles</th>
<th>4-bit Opcode</th>
<th>Status Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>No Operation (shift in 16-bit instruction)</td>
<td>1</td>
<td>0000</td>
<td>None</td>
</tr>
<tr>
<td>TBLRD *</td>
<td>Table Read (no change to TBLPTR)</td>
<td>2</td>
<td>1000</td>
<td>None</td>
</tr>
<tr>
<td>TBLRD *+</td>
<td>Table Read (post-increment TBLPTR)</td>
<td>2</td>
<td>1001</td>
<td>None</td>
</tr>
<tr>
<td>TBLRD *-</td>
<td>Table Read (post-decrement TBLPTR)</td>
<td>2</td>
<td>1010</td>
<td>None</td>
</tr>
<tr>
<td>TBLRD +*</td>
<td>Table Read (pre-increment TBLPTR)</td>
<td>2</td>
<td>1011</td>
<td>None</td>
</tr>
<tr>
<td>TBLWT *</td>
<td>Table Write (no change to TBLPTR)</td>
<td>2</td>
<td>1100</td>
<td>None</td>
</tr>
<tr>
<td>TBLWT +*</td>
<td>Table Write (post-increment TBLPTR)</td>
<td>2</td>
<td>1101</td>
<td>None</td>
</tr>
<tr>
<td>TBLWT *-</td>
<td>Table Write (post-decrement TBLPTR)</td>
<td>2</td>
<td>1110</td>
<td>None</td>
</tr>
<tr>
<td>TBLWT +**</td>
<td>Table Write (pre-increment TBLPTR)</td>
<td>2</td>
<td>1111</td>
<td>None</td>
</tr>
</tbody>
</table>

Legend: Refer to the PIC18CXXX Data Sheet (DS39026 or DS30475) for opcode field descriptions.

Note: All special instructions not included in this table are decoded as NOPs.
2.2.2 INITIAL SERIAL INSTRUCTION OPERATION

Upon ICSP mode entry, the CPU is idle. The execution of the CPU is governed by a state machine. While the first instruction is being clocked in, a forced \texttt{NOP} (\texttt{FNOP}) is executed.

Following the \texttt{FNOP} instruction execution and shifting in of the next instruction, the serial state machine will do one of three things, depending upon the 4-bit instruction fetched:

1. If the instruction fetched was a \texttt{NOP}, the state machine will suspend the CPU, awaiting a 16-bit wide instruction to be shifted in.
2. If the instruction is a \texttt{TBLWT} as shown in Figure 2-1, the state machine suspends the CPU from execution, while sixteen bits of data are shifted in as data for the \texttt{TBLWT} instruction.
3. If the instruction is a \texttt{TBLRD}, then execution of the \texttt{TBLRD} instruction begins immediately for eight clock cycles, followed by eight clock cycles where the contents of the TABLAT register is shifted out onto RB7.

Once sixteen clock cycles have elapsed, the next 4-bit instruction is fetched, while the current instruction is executed. Each instruction type is described in later sections.

FIGURE 2-1: SERIAL INSTRUCTION TIMING AFTER RESET

![Serial Instruction Timing Diagram](image-url)
2.2.3 NOP SERIAL INSTRUCTION EXECUTION

The NOP serial instruction is used to allow execution of all other instructions not included in Table 2-1. When the NOP instruction is fetched, the serial execution state machine suspends the CPU for 16 clock cycles. During these 16 clock cycles, all 16 bits of an instruction are fed into the CPU and the NOP instruction is discarded. Once all 16 bits have been shifted in, the state machine will allow the instruction to be executed for the next four clock cycles.

Note: 16-bit TBLWT and TBLRD instructions are not permitted. They will cause timing problems with the serial state machine. If the user wishes to perform a TBLWT or TBLRD instruction, it must be performed as a 4-bit instruction.

2.2.4 ONE-CYCLE 16-BIT INSTRUCTIONS

If the instruction fetched is a one-cycle instruction, then the instruction operation will be completed in the four clock cycles following the instruction fetched. During instruction execution, the next 4-bit serial instruction is fetched (see Figure 2-2).

FIGURE 2-2: SERIAL INSTRUCTION TIMING FOR 1-CYCLE, 16-BIT INSTRUCTIONS

![Serial Instruction Timing Diagram](https://example.com/image.png)
FIGURE 2-3: 16-BIT, 1-CYCLE SERIAL INSTRUCTION FLOW AFTER RESET

Start

MCLR = Vss, RB6, RB7 = 0

VPP = VihH

Enable CPU, execute FNOP, and shift in 1st 4-bit instruction, Num_Clk = 1, Qstate = Q<Num_Clk>

Clock transition RB6?

Yes

Shift(R) RB7 into ROMLAT<3>, Num_Clk = Num_Clk + 1

Qstate = Q<4>?

No

Yes

Hold CPU in Q4, 4-bit instruction = NOP, shift in 16-bit instruction, Num_Clk = 1

Clock transition RB6?

No

Yes

Shift(R) RB7 into ROMLAT<3>, Num_Clk = Num_Clk + 1

Qstate = Q<4>?

No

Yes

Enable CPU, execute 16-bit instruction, and shift in next 4-bit instruction, Num_Clk = 1, Qstate = Q<Num_Clk>

Clock transition RB6?

Yes

Shift(R) RB7 into ROMLAT<3>, Num_Clk = Num_Clk + 1

Qstate = Q<4>?

No

Yes

End

Num_Clk = 16?
FIGURE 2-4: 16-BIT, 1-CYCLE SERIAL INSTRUCTION FLOW

Start

1. Execute (PC - 2), and shift in next 4-bit instruction, Num_Clk = 1

2. Clock transition RB6?
   No
   4-bit instruction = NOP, shift in 16-bit instruction, Num_Clk = 1
   Yes
   Shift(R) RB7
   Num_Clk = Num_Clk + 1

3. Clock transition RB6?
   No
   4-bit instruction = NOP, shift in 16-bit instruction, Num_Clk = 1
   Yes
   Shift(R) RB7
   Num_Clk = Num_Clk + 1

4. Num_Clk = 167
   No
   Num_Clk = Num_Clk + 1
   Yes

5. Shift(R) RB7
   Num_Clk = Num_Clk + 1

6. Execute 16-bit instruction, and shift in next 4-bit instruction, Num_Clk = 1
2.3 Serial Instruction Execution For Two-Cycle, One-Word Instructions

When a \texttt{NOP} instruction is fetched, the serial execution state machine suspends the CPU for 16 clock cycles. During these 16 clock cycles, all 16 bits of an instruction are fed in and the \texttt{NOP} instruction is discarded.

If the instruction fetched is a two-cycle, one-word instruction, the instruction operation will require a second "dummy fetch" to be performed before the instruction execution can be completed. The first cycle of the instruction will be executed in the four clock cycles following the instruction fetched. During the first cycle of instruction execution, the next 4-bit serial instruction is fetched. To perform the second half of the two cycle instruction, this 4-bit instruction must be a \texttt{NOP}, so the state machine will remain idle for the second half of the instruction. Following the fetch of the second \texttt{NOP}, the state machine will shift 16 bits of data that will be discarded. After the 16 bits of data are shifted in, the state machine will release the CPU, and allow it to perform the second half of the two-cycle instruction. During the second half of the two-cycle instruction execution, the next 4-bit instruction is loaded (see Figure 2-5).

\begin{figure}
\centering
\includegraphics[width=\textwidth]{pic18cxxx.pdf}
\caption{16-BIT, 2-CYCLE, 1-WORD INSTRUCTION SEQUENCE}
\end{figure}
2.4 Serial Instruction Execution For Two-Word, Two-Cycle Instructions

After a NOP instruction is fetched, the serial execution state machine suspends the CPU in the Q4 state for 16 clock cycles. During these 16 clock cycles, all 16 bits of an instruction are fed in and the NOP instruction is discarded.

If the 16-bit instruction fetched is a two-cycle, two-word instruction, the instruction operation will require a second operand fetch to be performed before the instruction execution can be completed. The first cycle of the instruction will be executed in the four clock cycles following the 16-bit instruction fetch. During the first cycle of instruction execution, the next 4-bit serial instruction is fetched. To perform the second half of the two-cycle instruction, this 4-bit instruction must also be a NOP, so the state machine will remain idle for the second half of the instruction. Following the fetch of the second NOP, the state machine will shift 16 bits of data that will be used as an operand for the two-cycle instruction. After the 16 bits of data are shifted in, the state machine will release the CPU, and allow it to execute the second half of the two-cycle instruction. During the second half of the two-cycle instruction execution, the next 4-bit instruction is loaded (see Figure 2-6).

![FIGURE 2-6: 16-BIT, 2-CYCLE, 2-WORD INSTRUCTION SEQUENCE](image-url)

<table>
<thead>
<tr>
<th>Q Cycles</th>
<th>'Q1' 'Q2' 'Q3' 'Q4'</th>
<th>'Q1' 'Q2' 'Q3' 'Q4'</th>
<th>'Q1' 'Q2' 'Q3' 'Q4'</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCLR/VPP = VIH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RB6 (Clock)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RB7 (Data)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RB7 = Input

ICSP Mode
FIGURE 2-7: 16-BIT, 2-CYCLE, 2-WORD SERIAL INSTRUCTION FLOW AFTER RESET

Start

MCLR = Vss, RB6, RB7 = 0

VPP = VinH

Execute FNOP and shift in 1st 4-bit instruction, Num_Clk = 1

Clock transition RB6?

Yes

Shift(R) RB7, Num_Clk = Num_Clk + 1

4-bit instruction = NOP, shift in 16-bit instruction, Num_Clk = 1

Clock transition RB6?

No

Yes

Shift(R) RB7, Num_Clk = Num_Clk + 1

Num_Clk = 16?

Yes

No

Enable CPU, execute 1st cycle of 16-bit instruction, and shift in next 4-bit instruction, Num_Clk = 1

End

Clock transition RB6?

Yes

Shift(R) RB7, Num_Clk = Num_Clk + 1

4-bit instruction = NOP, shift in 2nd 16-bit operand, Num_Clk = 1

Clock transition RB6?

No

Yes

Shift(R) RB7, Num_Clk = Num_Clk + 1

Num_Clk = 16?

Yes

No

Execute 2nd cycle of 16-bit instruction, and shift in next 4-bit instruction Num_Clk = 1
FIGURE 2-8: 16-BIT, 2-CYCLE, 2-WORD SERIAL INSTRUCTION FLOW

Start

Execute (PC-2) and shift in 4-bit instruction, Num_Clk = 1

Clock transition RB6?

Yes

Shift(R) RB7, Num_Clk = Num_Clk + 1

No

4-bit instruction = NOP, shift in 16-bit instruction, Num_Clk = 1

Clock transition RB6?

Yes

Shift(R) RB7, Num_Clk = Num_Clk + 1

No

Execute 1st cycle of 16-bit instruction, and shift in next 4-bit instruction, Num_Clk = 1

Num_Clk = 16?

Yes

Execute 2nd cycle of 16-bit instruction, and shift in next 4-bit instruction, Num_Clk = 1

No

Shift(R) RB7, Num_Clk = Num_Clk + 1

4-bit instruction = NOP, shift in 2nd 16-bit operand, Num_Clk = 1

Clock transition RB6?

Yes

Shift(R) RB7, Num_Clk = Num_Clk + 1

No

Num_Clk = 16?

Yes

End

No
2.5 **TBLWT Instruction**

The **TBLWT** instruction is a special two-cycle instruction. All forms of **TBLWT** instructions (post/pre-increment, post-decrement, etc.) are encoded as 4-bit special instructions. This is useful as **TBLWT** instructions are used repeatedly in ICSP mode. A 4-bit instruction will minimize the total number of clock cycles required to perform programming algorithms.

The **TBLWT** instruction sequence operates as follows:

1. The 4-bit **TBLWT** instruction is read in by the state machine on RB7 during the four clock cycle execution of the instruction fetched previous to the **TBLWT** (which is a **FNOP** if the **TBLWT** is executed following a **RESET**).

2. Once the state machine recognizes that the instruction fetched is a **TBLWT**, the state machine proceeds to fetch in the 16 bits of data that will be written into the program memory location pointed to by the **TBLPTR**.

3. The state machine releases the CPU to execute the first cycle of the **TBLWT** instruction, while the first four bits of the 16-bit data word are shifted in. After the first cycle of **TBLWT** instruction has completed, the state machine shifts in the remaining 12 of the 16 bits of data. The data word will not be used until the second cycle of the instruction.

4. After all 16 bits of data are shifted in and the first cycle of the **TBLWT** is performed, the CPU will execute the second cycle of the **TBLWT** operation, programming the current memory location with the 16-bit value. The next instruction following the **TBLWT** instruction, **NOP**, is shifted in during the execution of the second cycle (see Figure 2-9).

The **TBLWT** instruction is used in ICSP mode to program the EPROM array. When writing a 16-bit value to the EPROM, ID locations, or configuration locations, the device, RB6 must be held high for the appropriate programming time during the **TBLWT** instruction, as specified by parameter P9.

When RB6 is asserted low, the device will cease programming the specified location.

After RB6 is asserted low, RB6 is held low for the time specified by parameter P10, to allow high voltage discharge of the program memory array.
FIGURE 2-10: TBLWT SERIAL INSTRUCTION FLOW AFTER RESET

Start

- MCLR = Vss, RB6, RB7 = 0
  - Vpp = Vih

  Execute FNOP, and shift in 4-bit TBLWT instruction, Num_Clk = 1

  Clock transition RB6?
    - Yes
      - Shift(R) RB7, Num_Clk = Num_Clk + 1
      - 4-bit instruction = TBLWT, execute 1st cycle of TBLWT, begin shifting in TBLWT data, Num_Clk = 1
    - No

  Clock transition RB6?
    - Yes
      - Shift(R) RB7, Num_Clk = Num_Clk + 1
      - Num_Clk = 4?
        - No
          - Shift in last 12 bits of TBLWT data, Num_Clk = 1
        - Yes
          - End
    - No

- Clock transition RB6?
  - Yes
    - Shift(R) RB7, Num_Clk = Num_Clk + 1
    - Num_Clk = 12?
      - No
        - End
      - Yes
        - Execute 2nd cycle of TBLWT instruction and shift in next 4-bit instruction, Num_Clk = 1
        - Clock transition RB6?
          - No
            - Shift(R) RB7, Num_Clk = Num_Clk + 1
          - Yes
            - End
FIGURE 2-11: TBLWT SERIAL INSTRUCTION FLOW

Start

Execute (PC-2), and shift in 4-bit TBLWT instruction, Num_Clk = 1

Clock transition RB6?

Yes

Shift(R) RB7
Num_Clk = Num_Clk + 1

4-bit instruction = TBLWT, execute 1st cycle of TBLWT, begin shifting in TBLWT data, Num_Clk = 1

Clock transition RB6?

Yes

Shift(R) RB7
Num_Clk = Num_Clk + 1

No

Num_Clk = 4?

Yes

Shift in last 12 bits of TBLWT data, Num_Clk = 1

No

Execute (PC-2), and shift in 4-bit TBLWT instruction, Num_Clk = 1

Clock transition RB6?

Yes

Shift(R) RB7
Num_Clk = Num_Clk + 1

No

Num_Clk = 12?

Yes

Execute 2nd cycle of TBLWT instruction and shift in next 4-bit instruction, Num_Clk = 1

No

Clock transition RB6?

Yes

Shift(R) RB7
Num_Clk = Num_Clk + 1

No

End
2.6 **TBLRD Instruction**

The TBLRD instruction is another special two-cycle instruction. All forms of TBLRD instructions (post/pre-increment, post-decrement, etc.) are encoded as 4-bit special instructions. This is useful as TBLRD instructions are used repeatedly in ICSP mode. A 4-bit instruction will minimize the total number of clock cycles required to perform programming algorithms.

The TBLRD instruction sequence operates as follows:

1. The 4-bit TBLRD instruction is read in by the state machine on RB7 during the four clock cycle execution of the instruction fetched previous to the TBLRD (which is an FNOP if the TBLRD is executed following a RESET).

2. Once the state machine recognizes that the instruction fetched is a TBLRD, the state machine releases the CPU and allows execution of the first and second cycles of the TBLRD instruction for eight clock cycles. When the TBLRD is performed, the contents of the program memory byte pointed to by the TBLPTR is loaded into the TABLAT register.

3. After eight clock cycles have transitioned on RB6, and the TBLRD instruction has completed, the state machine will suspend the CPU for eight clock cycles. During these eight clock cycles, the state machine configures RB7 as an output, and will shift out the contents of the TABLAT register onto RB7, LSb first.

4. When the state machine has shifted out all eight bits of data, the state machine suspends the CPU to allow an instruction pre-fetch. Four clock cycles are required on RB6 to shift in the next 4-bit instruction.

---

**FIGURE 2-12: TBLRD INSTRUCTION SEQUENCE**

| Q Cycles | Q1 Q2 Q3 Q4
| MCLR/VPP = VIH | Q1 Q2 Q3 Q4
| RB6 (Clock) | 1 2 3 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 1 2 3 4
| RB7 (Data) | 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1

Execute PC-2
Fetch TBLRD

Execute Cycle 1
TBLRD

Execute Cycle 2
TBLRD

Shift data out from TABLAT

No execution takes place, fetch next 4-bit instruction

RB7 = Input

RB7 = Output

RB7 = Input

ICSP Mode

© 2003 Microchip Technology Inc.
FIGURE 2-13: TBLRD SERIAL INSTRUCTION FLOW AFTER RESET

- **Start**
  - MCLR = Vss, RB6, RB7 = 0
  - VPP = VIH

- **Execute FNOP**, and shift in 4-bit TBLRD instruction, Num_Clk = 1

- **Clock transition RB6?**
  - No
  - Shift(R) RB7, Num_Clk = Num_Clk + 1
  - Enable CPU, execute 1st and 2nd cycle TBLRD instruction
  - **Clock transition RB6?**
    - No
    - TBLRD instruction execution takes place here, Num_Clk = Num_Clk + 1
    - **Num_Clk = 8?**
      - No
      - Shift out 8 bits of data to RB7
      - End
    - Yes
      - Shift(R) TABLAT<0> out onto RB7, Num_Clk = Num_Clk + 1
      - **Num_Clk = 8?**
        - No
        - Shift in next 4-bit instruction
        - **Clock transition RB6?**
          - No
          - **Clock transition RB6?**
            - Yes
            - Shift(R) RB7, Num_Clk = Num_Clk + 1
            - **Num_Clk = 4?**
              - No
              - End
              - Yes
                - Shift(R) TABLAT<0> out onto RB7, Num_Clk = Num_Clk + 1
                - **Num_Clk = 8?**
                  - No
                  - Shift out 8 bits of data to RB7
                  - End
                  - Yes
                    - Shift(R) TABLAT<0> out onto RB7, Num_Clk = Num_Clk + 1
                    - **Num_Clk = 8?**
                      - No
                      - Shift out 8 bits of data to RB7
                      - End
                      - Yes
FIGURE 2-14: TBLRD SERIAL INSTRUCTION FLOW

Start

Execute (PC-2), and shift in 4-bit TBLRD instruction, Num_Clk = 1

Clock transition RB6?

Yes

No

Shift(R) RB7
Num_Clk = Num_Clk + 1

Execute 1st and 2nd cycle TBLRD instruction

Clock transition RB6?

Yes

No

TBLRD instruction execution takes place here
Num_Clk = Num_Clk + 1

Num_Clk = 8?

Yes

No

Shift out 8 bits of data to RB7

Clock transition RB6?

Yes

No

Shift(R) TABLAT<0> out onto RB7
Num_Clk = Num_Clk + 1

Num_Clk = 8?

Yes

No

Shift in next 4-bit instruction

Clock transition RB6?

Yes

No

Shift(R) RB7
Num_Clk = Num_Clk + 1

Num_Clk = 4?

Yes

No

End
2.6.1 SOFTWARE COMMANDS
ICSP commands of the PICmicro® MCU are supported in the PIC18CXXX family by simply combining CPU instructions. Once in the ICSP mode, instructions are loaded into a shift register, and the device waits for a command to be received. The ICSP commands for the PIC18CXXX family are now “pseudo-commands” and are shown in Table 2-2. The following sections describe how to implement the pseudo-commands using CPU instructions.

<table>
<thead>
<tr>
<th>TABLE 2-2: ICSP PSEUDO COMMAND MAPPING</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICSP™ Command</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>Load Configuration</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Load Data</td>
</tr>
<tr>
<td>Read Data</td>
</tr>
<tr>
<td>Increment Address</td>
</tr>
<tr>
<td>Load Address</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>RESET Address</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Begin Programming</td>
</tr>
<tr>
<td>End Programming</td>
</tr>
</tbody>
</table>

2.6.2 RESET ADDRESS
A reset of the program memory pointer is a write to the upper, high, and low bytes of the TBLPTR. To reset the program memory pointer, the following instruction sequence is used.

NOP ;(4-BIT INSTRUCTION)
MOVLIW 00h
NOP ;(4-BIT INSTRUCTION)
MOVWF TBLPTRL
NOP ;(4-BIT INSTRUCTION)
MOVWF TBLPTRH
NOP ;(4-BIT INSTRUCTION)
MOVWF TBLPTRU
FIGURE 2-15:  RESET ADDRESS SERIAL INSTRUCTION SEQUENCE

Start

Execute (PC - 2), shift in next 4-bit instruction, Num_Clk = 1

On rising edge RB6, Shift(R) RB7 into Shift Reg<3>, Num_Clk = Num_Clk + 1

Num_Clk = 4?
No

4-bit instruction = NOP, Shift in 16-bit MOVWF instruction, Num_Clk = 1

MOVLW 00h

On rising edge RB6, Shift(R) RB7 into Shift Reg<3>, Num_Clk = Num_Clk + 1

Num_Clk = 16?
No

Execute MOVLW instruction, shift in 4-bit NOP instruction, Num_Clk = 1

On rising edge RB6, Shift(R) RB7 into Shift Reg<3>, Num_Clk = Num_Clk + 1

Num_Clk = 4?
No

4-bit instruction = NOP, shift in 4-bit NOP instruction, Num_Clk = 1

On rising edge RB6, Shift(R) RB7 into Shift Reg<15>, Num_Clk = Num_Clk + 1

Num_Clk = 16?
No

Execute MOVWF instruction, shift in 4-bit NOP instruction, Num_Clk = 1

On rising edge RB6, Shift(R) RB7 into Shift Reg<15>, Num_Clk = Num_Clk + 1

Num_Clk = 4?
No

4-bit instruction = NOP, shift in 16-bit MOVWF instruction, Num_Clk = 1

On rising edge RB6, Shift(R) RB7 into Shift Reg<15>, Num_Clk = Num_Clk + 1

MOVLW TBLPTRH

Num_Clk = 16?
No

Execute MOVWF instruction, shift in next 4-bit instruction, Num_Clk = 1
2.6.3 LOAD ADDRESS

This is used to load the address pointer to the Program Memory with a specific 22-bit value, and is useful when a specific range of locations are to be accessed. To load the address into the table pointer, the following commands must be used:

```assembly
NOP ; 4-bit instruction
MOVLW Low_Address
NOP ; 4-bit instruction
MOVWF TBLPTRL
NOP ; 4-bit instruction
MOVLW High_Address
NOP ; 4-bit instruction
MOVWF TBLPTRH
NOP ; 4-bit instruction
MOVLW Upper_Address
NOP ; 4-bit instruction
MOVWF TBLPTRU
```
FIGURE 2-17: LOAD ADDRESS SERIAL INSTRUCTION SEQUENCE

1. **Start**
   - Execute (PC - 2), shift in next 4-bit instruction, Num_Clk = 1

2. **On rising edge RB6, Shift(R) RB7 into Shift Reg<3>, Num_Clk = Num_Clk + 1**
   - Num_Clk = 47? No
   - Num_Clk = 16? No
   - Execute MOVWF instruction, shift in 4-bit NOP instruction, Num_Clk = 1

3. **On rising edge RB6, Shift(R) RB7 into Shift Reg<15>, Num_Clk = Num_Clk + 1**
   - Num_Clk = 47? No
   - Num_Clk = 16? No
   - Execute MOVWF instruction, shift in 4-bit NOP instruction, Num_Clk = 1

4. **4-bit instruction = NOP, shift in 16-bit MOVWF instruction, Num_Clk = 1**
   - MOVLW Low_Address
   - MOVLW High_Address

5. **On rising edge RB6, Shift(R) RB7 into Shift Reg<3>, Num_Clk = Num_Clk + 1**
   - Num_Clk = 47? No
   - Num_Clk = 16? No
   - Execute MOVWF instruction, shift in next 4-bit instruction, Num_Clk = 1
FIGURE 2-18: LOAD ADDRESS SERIAL INSTRUCTION SEQUENCE (CONTINUED)

On rising edge RB6,
Shift(R) RB7
into Shift Reg<3>,
Num_Clk = Num_Clk + 1
Yes

4-bit instruction = NOP,
Shift in 16-bit MOVWF instruction,
Num_Clk = 1

On rising edge RB6,
Shift(R) RB7
into Shift Reg<15>,
Num_Clk = Num_Clk + 1
Yes

Execute MOVWF instruction,
shift in next 4-bit instruction,
Num_Clk = 1
End

A

Num_Clk = 4?
No

Num_Clk = 16?
No

MOVLM
Upper_Address

MOVWM
Upper_Address
2.6.4 ICSP BEGIN PROGRAMMING

Programming is performed by executing a TBLWT instruction. In ICSP mode, the TBLWT instruction sequence will include 16 bits of data shifted into a data buffer, and then written to the word location addressed by the TBLPTR. Although the TBLPTR addresses the program memory on a byte wide boundary, all 16 bits of data shifted in during the TBLWT sequence are written at once. The 16 bits are shifted into the TABLAT and buffer registers. The TBLPTR points to the word that will be programmed; it can point to either the high or the low byte (see Figure 2-19).

The sequence for programming a location could occur as follows:

1. Set up the TLPBTR with the first address to be programmed (even or odd byte).
2. Shift in a 4-bit TBLWT instruction.
3. 16 bits of data are shifted in for programming both high and low byte of the first programmed location.
4. Execute TBLWT instruction to program location.
5. Verify high byte (odd address) by executing TBLRD* (post-decrement). (TBLPTR points at odd address.)
6. Verify low byte (even address) by executing TBLRD* (post-increment). (TBLPTR points at odd address again.)
7. If location doesn’t verify, go back to step 4.
8. If location does verify, begin 3x over-programming (see Section 2.6.7).

The TBLWT instruction offers flexibility with multiple addressing modes: pre-increment, post-increment, post-decrement, and no change of the TBLPTR. These modes eliminate the need for the increment address command sequence.

FIGURE 2-19: DATA BUFFERING SCHEME FOR ICSP
2.6.5 PROGRAMMING INSTRUCTION SEQUENCE

The instructions needed to execute a programming sequence are shown in the following example. Many of the instruction sequences are also shown in previous sections.

```
NOP              ; 4-bit instruction
                ; Set up low byte
                ; of program address
MOVLW Low_Byte_Address ; = 00
NOP              ; 4-bit instruction
MOVWF TBLPTRL
NOP              ; 4-bit instruction
                ; Set up high byte
                ; of program
                ; address
MOVLW High_Byte_Address ; = 00
NOP              ; 4-bit instruction
MOVWF TBLPTRH
NOP              ; 4-bit instruction
                ; Set up upper byte
                ; of program
                ; address
MOVLW Upper_Byte_Address ; = 00
NOP              ; 4-bit instruction
MOVWF TBLPTRU
TBLWT+*          ; TBLPTR = 000000h
```

A write of a program memory location with an odd or an even address causes a long write cycle in ICSP mode. The 16-bit data is encoded in the TBLWT sequence and is loaded into the temporary buffer register for word wide writes.

2.6.6 VERIFY SEQUENCE

The table pointer = 000001h in the last example. A TBLRD will then read the odd address byte of the current program word address location first. The verify sequence will be as follows:

```
TBLRD*            ; Read/verify high byte first
                ; TBLPTR = 0000 post-dec
TBLRD*            ; Read/verify low byte
```

The first TBLRD decrements the table pointer to point to the even address byte of the current program word. After the first and second cycle of the TBLRD are performed, all eight bits of data are shifted out on RB7. The fetch of the second TBLRD occurs on the next four clock cycles. The second TBLRD does not modify the table pointer address. This allows another programming cycle (TBLWT+) to take place if the verify doesn't match the program data, without having to update the table pointer.

If the contents of the verify do not match the intended program data word, then the TBLWT instruction must be repeated with the correct contents of the current program word. Therefore, only one instruction needs to be performed to repeat the programming cycle:

```
TBLWT+*
```

2.6.7 3X OVER-PROGRAMMING

Once a location has been both programmed and verified over the range of voltages, 3x over-programming should be applied. In other words, apply three times the number of programming pulses required to program a location in memory to ensure solid programming margin.

This means that every location will be programmed a minimum of four times (1 + 3x over-programming).
FIGURE 2-20: DETAILED PROGRAMMING FLOW CHART – PROGRAM MEMORY

Start

- \[ V_{PP} = V_{IH}, \text{ RB6, RB7 = 0} \]

- \[ N = 1 \]

- Execute `FNOP` for four clock cycles, shift in 4-bit `NOP`

- 4-bit instruction = `NOP`, shift in 16-bit `MOVLW Low_Addr` instruction for 16 clock cycles

- Execute `MOVLW` for four clock cycles and shift in 4-bit `NOP`

- 4-bit instruction = `NOP`, shift in 16-bit `MOVLW TBLPTRL` instruction for 16 clock cycles

- Execute `MOVLWF` for four clock cycles and shift in 4-bit `NOP`

- 4-bit instruction = `NOP`, shift in 16-bit `MOVLW Upper_Addr` instruction for 16 clock cycles

- Execute `MOVLW` for four clock cycles and shift in 4-bit `NOP`

- 4-bit instruction = `NOP`, shift in 16-bit `MOVLW TBLPTRU` instruction for 16 clock cycles

- Execute current instruction for four clock cycles, and shift in 4-bit `TBLWT+*`

- Execute 1st cycle `TBLWT+*`, and shift in first four bits of data for four clock cycles

- Shift in last 12 bits of data for 12 clock cycles

- Execute 2nd cycle `TBLWT+*` for four clock cycles and shift in `TBLRD*` for four clock cycles

- Hold RB6 clock high (P9)

- Clock low for discharge (P10)

- Execute 1st and 2nd cycle `TBLRD*` for eight clock cycles

- Shift data out for eight clock cycles

- Hold CPU, shift in `TBLRD*` for four clock cycles

- Execute 1st and 2nd cycle `TBLRD*` for eight clock cycles

- Shift data out for eight clock cycles

- Verify?

  - Yes

  - No

  - \[ N = N + 1 \]

  - \[ N > 25? \]

  - Yes

  - No

  - Report Programming Failure
FIGURE 2-21: DETAILED PROGRAMMING FLOW CHART – PROGRAM MEMORY (CONTINUED)

A

N = 3 * N

Execute current instruction, shift in TBLWT++ for four clock cycles

Execute 1st cycle TBLWT++ or TBLWT+, and shift in first four bits of data for four clock cycles

Shift in last 12 bits of data for 12 clock cycles

N = 1?

Yes

Execute 2nd cycle TBLWT* for four clock cycles and shift in TBLWT* for four clock cycles

Hold RB6 high (P9)

Shift in last 12 bits of data for 12 clock cycles

Execute current instruction for four clock cycles, and shift in 4-bit TBLWT+*

Hold RB6 high (P9)

Clock low for discharge (P10)

N = N - 1

No

Execute 2nd cycle TBLWT* for four clock cycles and shift in TBLWT* for four clock cycles

Report all locations @ VDDMIN

Data correct?

Yes

Verify all locations @ VDDMAX

No

Data correct?

Yes

Verify all locations @ VDDMAX

No

End

All locations programmed?

No

B

Report Verify Error @ VDDMIN

Yes

Report Verify Error @ VDDMAX
2.6.8 LOAD CONFIGURATION

The Configuration registers are located in test memory, and are only addressable when the high address bit of the TBLPTR (bit 21) is set. Test program memory contains test memory, configuration registers, calibration registers, and ID locations. The desired address must be loaded into all three bytes of the table pointer to program specific ID locations, or the configuration bits. To program the configuration registers, the following sequence must be followed:

NOP ; 4-bit instruction
; shift in 16-bit
; MOVWL instruction
MOVLW 03h
NOP ; 4-bit instruction
; shift in 16-bit
; MOVWF instruction
; Enable Test memory
MOVWF TBLPTRU
NOP ; 4-bit instruction
; shift in 16-bit
; MOVWL instruction
MOVWL Low_Config_Address
NOP ; 4-bit instruction
; shift in 16-bit
; MOVWF instruction
MOVWF TBLPTRL
NOP ; 4-bit instruction
; shift in 16-bit
; MOVWL instruction
MOVWL High_Config_Address
NOP ; 4-bit instruction
; shift in 16-bit
; MOVWF instruction
MOVWF TBLPTRH
NOP ; 4-bit instruction
; shift in 16-bit
; MOVWL instruction

TBLWT*+ ; 16-bits of data are
; shifted in for write
; of config1L and
; config1H TBLWT is a
; 4-bit special
; instruction.
; Wait P9 for
; programming

2.6.9 END PROGRAMMING

When programming occurs, 16 bits of data are programmed into memory. The 16 bits of data are shifted in during the TBLWT sequence. After the programming command (TBLWT) has been executed, the user must wait P9 until programming is complete, before another command can be executed by the CPU. There is no command to end programming.

RB6 must remain high for as long as programming is desired. When RB6 is lowered, programming will cease.

After the falling edge occurs on RB6, RB6 must be held low for a period of time (Parameter 10), so a high voltage discharge can be performed. This ensures the program array isn’t stressed at high voltage during execution of the next instruction. The high voltage discharge will occur while RB6 is low, following the programming time.
FIGURE 2-22: DETAILED PROGRAMMING FLOW CHART – CONFIG WORD

**START**

1. MCLR = Vss
   4.75 V < Vdd < 5.25 V
2. VPP = Vihh
3. Execute FNOP for four clock cycles, shift in 4-bit NOP
4. 4-bit instruction = NOR, shift in 16-bit MOVLW 30
   instruction for 16 clock cycles
5. Execute MOVLW for four clock cycles
   and shift in 4-bit NOP
6. 4-bit instruction = NOR, shift in 16-bit MOVLW 00
   instruction for 16 clock cycles
7. Execute MOVLW for four clock cycles
   and shift in 4-bit NOP
8. 4-bit instruction = NOR, shift in 16-bit MOVLW 00
   instruction for 16 clock cycles
9. Execute MOVLW for four clock cycles
   and shift in 4-bit NOP
10. 4-bit instruction = NOR, shift in 16-bit MOVLW 00
    instruction for 16 clock cycles

**Flowchart**

- **B**
  - Execute MOVLW for four clock cycles
    and shift in 4-bit NOP
  - 4-bit instruction = NOR, shift in 16-bit MOVLW 30
    instruction for 16 clock cycles
  - Execute MOVLW for four clock cycles
    and shift in 4-bit NOP
  - 4-bit instruction = NOR, shift in 16-bit MOVLW 00
    instruction for 16 clock cycles
  - Execute MOVLW for four clock cycles
    and shift in 4-bit NOP
  - 4-bit instruction = NOR, shift in 16-bit MOVLW 00
    instruction for 16 clock cycles

- **A**
  - Execute last fetched instruction for four clock cycles
    and shift in 4-bit TBLWT*-
  - Execute 1st cycle TBLWT, and shift in first four bits
    of configuration registers for four clock cycles
  - Shift in last 12 bits of data for 12 clock cycles
  - N = 1?
    - Yes
    - Execute 2nd cycle TBLWT for four clock cycles
      and shift in TBLWT* for four clock cycles
    - RB6 high (P9)
    - Clock low for discharge (P10)
    - N = N - 1
    - Execute 2nd cycle TBLWT* for four clock cycles
      and shift in TBLWT*- for four clock cycles
    - Wait P9 + P10 to ensure programming
  - No
  - Wait P9 + P10 to ensure programming

- **Avoid**

- **Start**
  - Execute last fetched instruction for four clock cycles
    and shift in 4-bit TBLWT*-
  - Execute 1st cycle TBLWT, and shift in first four bits
    of configuration registers for four clock cycles
  - Shift in last 12 bits of data for 12 clock cycles
  - N = 1?
    - Yes
    - Execute 2nd cycle TBLWT for four clock cycles
      and shift in TBLWT* for four clock cycles
    - RB6 high (P9)
    - Clock low for discharge (P10)
    - N = N - 1
    - Execute 2nd cycle TBLWT* for four clock cycles
      and shift in TBLWT*- for four clock cycles
    - Wait P9 + P10 to ensure programming
  - No
  - Wait P9 + P10 to ensure programming
FIGURE 2-23: DETAILED PROGRAMMING FLOW CHART – CONFIG WORD

A

Execute 1st cycle TBLWT*-, and shift in first four bits of configuration registers for four clock cycles

Shift in last 12 bits of data for 12 clock cycles

Execute 2nd cycle TBLWT* for four clock cycles and shift in TBLRD*+ for four clock cycles

Execute 1st and 2nd cycle TBLRD*+ for eight clock cycles

Shift data out for eight clock cycles

Shift in TBLRD*+ for four clock cycles

Execute 1st and 2nd cycle TBLRD*+ for eight clock cycles

Shift data out for eight clock cycles

Verify?

Yes

Report Verify Error

No

All locations programmed?

Yes

Verify all ID Locations @ VDDMIN

Data correct?

Yes

Report Verify @ VDDMIN

No

Verify all locations @ VDDMAX

Data correct?

Yes

Report Verify Error @ VDDMAX

No

DONE

Yes

Verify?

No

Report Verify Error

Yes
FIGURE 2-24: DETAILED PROGRAMMING FLOW CHART – ID LOCATION

Start

VPP = Vih, RB6, RB7 = 0

N = 1

Execute FNOP for four clock cycles, shift in 4-bit NOP

4-bit instruction = NOR, shift in 16-bit MOVLW Low_Addr instruction for 16 clock cycles

Execute MOVLW for four clock cycles and shift in 4-bit NOP

4-bit instruction = NOR, shift in 16-bit MOVLW TBLPTRL instruction for 16 clock cycles

Execute MOVF for four clock cycles and shift in 4-bit NOP

4-bit instruction = NOR, shift in 16-bit MOVLW High_Addr instruction for 16 clock cycles

Execute MOVLW for four clock cycles and shift in 4-bit NOP

4-bit instruction = NOR, shift in 16-bit MOVF TBLPTRH instruction for 16 clock cycles

Execute current instruction for four clock cycles, and shift in 4-bit TBLWT+*

B

Execute 1st cycle TBLWT+*, and shift in first four bits of data for four clock cycles

Shift in last 12 bits of data for 12 clock cycles

Execute 2nd cycle TBLWT+* for four clock cycles and shift in TBLRD* for four clock cycles

Execute 1st and 2nd cycle TBLRD* for eight clock cycles

Shift data out for eight clock cycles

Execute 1st and 2nd cycle TBLRD* for eight clock cycles

Shift data out for eight clock cycles

A

Verify?

Yes

N = N + 1

No

N > 25?

Yes

Report Programming Failure

No
FIGURE 2-25: DETAILED PROGRAMMING FLOW CHART – ID LOCATION (CONTINUED)

A

N = 3 * N

Execute current instruction, shift in TBLWT*+ for four clock cycles

Execute 1st cycle TBLWT** or TBLWT*, and shift in first four bits of data for four clock cycles

Shift in last 12 bits of data for 12 clock cycles

N = 1?

Yes

Execute 2nd cycle TBLWT* for four clock cycles and shift in TBLWT* for four clock cycles

Hold RB6 high (P9)

Clock low for discharge (P10)

N = N - 1

No

Execute 2nd cycle TBLWT* for four clock cycles, and shift in 4-bit TBLWT**

Hold RB6 high (P9)

Clock low for discharge (P10)

All locations programmed?

Yes

B

Report Verify Error @ VDDMIN

Data correct?

Yes

Verify all locations @ VDDMAX

Data correct?

No

End

No

Verify all locations @ VDDMIN

Clock low for discharge (P10)
3.0 CONFIGURATION WORD

The configuration bits can be programmed (read as ‘0’),
or left unprogrammed (read as ‘1’), to select various
device configurations. These bits are mapped starting
at program memory location 300000h.

The user will note that address 300000h is beyond the
user program memory space. In fact, it belongs to the
configuration memory space (300000h – 3FFFFFh).

3.1 ID Locations

A user may store identification information (ID) in eight
ID locations mapped in [0x200000:0x200007]. It is
recommended that the user use only the four Least
Significant bits of each ID location.

The ID locations do not read out in a scrambled fashion
after code protection is enabled. For all devices, it is
recommended to write ID locations as ‘1111 bbbb’
where ‘bbbb’ is the ID information.

Note: The PIC18C601/801 devices do not have
user ID locations.

### TABLE 3-1: 18CXX2 CONFIGURATION BITS AND DEVICE IDS

<table>
<thead>
<tr>
<th>File Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Default/ Unprogrammed Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>300000h</td>
<td>CONFIG1L</td>
<td>CP</td>
<td>CP</td>
<td>CP</td>
<td>CP</td>
<td>CP</td>
<td>CP</td>
<td>CP</td>
<td>1111 1111</td>
</tr>
<tr>
<td>300001h</td>
<td>CONFIG1H</td>
<td>r</td>
<td>r</td>
<td>OSCSEN</td>
<td>—</td>
<td>—</td>
<td>FOSC2</td>
<td>FOSC1</td>
<td>FOSC0</td>
</tr>
<tr>
<td>300002h</td>
<td>CONFIG2L</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>BORV1</td>
<td>BORV0</td>
<td>BOREN</td>
</tr>
<tr>
<td>300003h</td>
<td>CONFIG2H</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>WDTPS2</td>
<td>WDTPS1</td>
<td>WDTPS0</td>
</tr>
<tr>
<td>300005h</td>
<td>CONFIG3H</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>C2MX</td>
</tr>
<tr>
<td>300006h</td>
<td>CONFIG4L</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>r STVREN</td>
</tr>
<tr>
<td>3FFFFeh</td>
<td>DEVID1</td>
<td>DEV2</td>
<td>DEV1</td>
<td>DEV0</td>
<td>REV4</td>
<td>REV3</td>
<td>REV2</td>
<td>REV1</td>
<td>REV0 0000 0000</td>
</tr>
<tr>
<td>3FFFFf</td>
<td>DEVID2</td>
<td>DEV10</td>
<td>DEV9</td>
<td>DEV8</td>
<td>DEV7</td>
<td>DEV6</td>
<td>DEV5</td>
<td>DEV4</td>
<td>DEV3 0000 0010</td>
</tr>
</tbody>
</table>

Legend: x = unknown, u = unchanged, - = unimplemented, q = value depends on condition, r = reserved.
Grayed cells are unimplemented, read as 0.

### TABLE 3-2: 18CXX8 CONFIGURATION BITS AND DEVICE IDS

<table>
<thead>
<tr>
<th>File Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Default/ Unprogrammed Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>300000h</td>
<td>CONFIG1L</td>
<td>CP</td>
<td>CP</td>
<td>CP</td>
<td>CP</td>
<td>CP</td>
<td>CP</td>
<td>CP</td>
<td>1111 1111</td>
</tr>
<tr>
<td>300001h</td>
<td>CONFIG1H</td>
<td>r</td>
<td>r</td>
<td>OSCSEN</td>
<td>—</td>
<td>—</td>
<td>FOSC2</td>
<td>FOSC1</td>
<td>FOSC0</td>
</tr>
<tr>
<td>300002h</td>
<td>CONFIG2L</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>BORV1</td>
<td>BORV0</td>
<td>BOREN</td>
</tr>
<tr>
<td>300003h</td>
<td>CONFIG2H</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>WDTPS2</td>
<td>WDTPS1</td>
<td>WDTPS0</td>
</tr>
<tr>
<td>300006h</td>
<td>CONFIG4L</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>r STVREN</td>
</tr>
<tr>
<td>3FFFFeh</td>
<td>DEVID1</td>
<td>DEV2</td>
<td>DEV1</td>
<td>DEV0</td>
<td>REV4</td>
<td>REV3</td>
<td>REV2</td>
<td>REV1</td>
<td>REV0 0000 0000</td>
</tr>
<tr>
<td>3FFFFf</td>
<td>DEVID2</td>
<td>DEV10</td>
<td>DEV9</td>
<td>DEV8</td>
<td>DEV7</td>
<td>DEV6</td>
<td>DEV5</td>
<td>DEV4</td>
<td>DEV3 0000 0010</td>
</tr>
</tbody>
</table>

Legend: x = unknown, u = unchanged, - = unimplemented, q = value depends on condition, r = reserved.
Grayed cells are unimplemented, read as 0.

### TABLE 3-3: 18C601/801 CONFIGURATION BITS AND DEVICE IDS

<table>
<thead>
<tr>
<th>Filename</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Default/ Unprogrammed Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>300001h</td>
<td>CONFIG1H</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>FOSC1</td>
<td>FOSC0</td>
<td>----- 10</td>
</tr>
<tr>
<td>300002h</td>
<td>CONFIG2L</td>
<td>—</td>
<td>BW</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>PWRTEN</td>
</tr>
<tr>
<td>300003h</td>
<td>CONFIG2H</td>
<td>—</td>
<td>—</td>
<td>WDTPS2</td>
<td>WDTPS1</td>
<td>WDTPS0</td>
<td>WDTEN</td>
<td>----- 1111</td>
<td></td>
</tr>
<tr>
<td>300006h</td>
<td>CONFIG4L</td>
<td>r</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>STVREN</td>
</tr>
<tr>
<td>3FFFFeh</td>
<td>DEVID1</td>
<td>DEV2</td>
<td>DEV1</td>
<td>DEV0</td>
<td>REV4</td>
<td>REV3</td>
<td>REV2</td>
<td>REV1</td>
<td>REV0 0000 0000</td>
</tr>
<tr>
<td>3FFFFf</td>
<td>DEVID2</td>
<td>DEV10</td>
<td>DEV9</td>
<td>DEV8</td>
<td>DEV7</td>
<td>DEV6</td>
<td>DEV5</td>
<td>DEV4</td>
<td>DEV3 0000 0010</td>
</tr>
</tbody>
</table>

Legend: x = unknown, u = unchanged, - = unimplemented, q = value depends on condition, r = reserved.
Shaded cells are unimplemented, read as ‘0’.
<table>
<thead>
<tr>
<th>Bit Name</th>
<th>Bit Type</th>
<th>File Name/Devices</th>
<th>Description</th>
</tr>
</thead>
</table>
| CP         | R/P – 1  | CONFIG1L/18CXX2 and 18CXX8 | Code Protection bits  
1 = Program memory code protection off  
0 = All of program memory code protected |
| OSCSEN     | R/P – 1  | CONFIG1H/18CXX2 and 18CXX8 | Oscillator System Clock Switch Enable bit  
1 = Oscillator system clock switch option is disabled (main oscillator is source)  
0 = Oscillator system clock switch option is enabled (oscillator switching is enabled) |
| FOSC2: FOSC0 | R/P – 1 | CONFIG1H/18CXXX | Oscillator Selection bits  
111 = RC oscillator w/OSC2 configured as RA6 (reserved on PIC18C601/801)  
110 = HS oscillator with PLL enabled/Clock frequency = (4 X Fosc) (reserved on PIC18C601/801)  
101 = EC oscillator w/OSC2 configured as RA6 (reserved on PIC18C601/801)  
100 = EC oscillator w/OSC2 configured as divide by 4 clock output (reserved on PIC18C601/801)  
011 = RC oscillator  
010 = HS oscillator  
001 = XT oscillator  
000 = LP oscillator |
| BORV1: BORV0 | R/P – 1 | CONFIG2L/18CXX2 and 18CXX8 | Brown-out Reset Voltage bits  
11 = Vbor set to 2.5V  
10 = Vbor set to 2.7V  
01 = Vbor set to 4.2V  
00 = Vbor set to 4.5V |
| BOREN      | R/P – 1  | CONFIG2L/18CXX2 and 18CXX8 | Brown-out Reset Enable bit  
1 = Brown-out Reset enabled  
0 = Brown-out Reset disabled |
| PWRTEN     | R/P – 1  | CONFIG2L/18CXXX | Power-up Timer Enable bit  
1 = PWRT disabled  
0 = PWRT enabled  
Enabling Brown-out Reset automatically enables the Power-up Timer (PWRT), regardless of the value of bit PWRTEN. Ensure Power-up Timer is enabled when Brown-out Reset is enabled. |
| WDTPS2: WDTPS0 | R/P – 1 | CONFIG2H/18CXXX | Watchdog Timer Postscale Select bits  
111 = 1:128  
110 = 1:64  
101 = 1:32  
100 = 1:16  
011 = 1:8  
010 = 1:4  
001 = 1:2  
000 = 1:1 |

Legend: R = readable, P = programmable, U = unimplemented, read as '0',  
- n = value when device is unprogrammed, u = unchanged.
<table>
<thead>
<tr>
<th>Bit Name</th>
<th>Bit Type</th>
<th>File Name/Devices</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDTEN</td>
<td>R/P – 1</td>
<td>CONFIG2H/18CXXX</td>
<td>Watchdog Timer Enable bit&lt;br&gt;1 = WDT enabled&lt;br&gt;0 = WDT disabled (control is placed on SWDTEN bit)</td>
</tr>
<tr>
<td>CCP2MX</td>
<td>R/P – 1</td>
<td>CONFIG3H/18CXX2</td>
<td>CCP2 Mux bit&lt;br&gt;1 = CCP2 input/output is multiplexed with RC1&lt;br&gt;0 = CCP2 input/output is multiplexed with RB3</td>
</tr>
<tr>
<td>STVREN</td>
<td>R/P – 1</td>
<td>CONFIG4L/18CXXX</td>
<td>Stack Overflow/Underflow Reset Enable bit&lt;br&gt;1 = Stack Overflow/Underflow will cause RESET&lt;br&gt;0 = Stack Overflow/Underflow will not cause RESET</td>
</tr>
<tr>
<td>BW</td>
<td>R/P – 1</td>
<td>CONFIG2L/18C601/801</td>
<td>External Bus Data Width bit&lt;br&gt;1 = 16-bit External Bus mode&lt;br&gt;0 = 8-bit External Bus mode</td>
</tr>
<tr>
<td>DEV10:DEV3</td>
<td>R</td>
<td>DEVID2/18CXXX</td>
<td>Device ID bits&lt;br&gt;These bits are used with the DEV2:DEV0 bits in the DEVID1 register to identify part number.</td>
</tr>
<tr>
<td>DEV2:DEV0</td>
<td>R</td>
<td>DEVID1/18CXXX</td>
<td>Device ID bits&lt;br&gt;These bits are used with the DEV10:DEV3 bits in the DEVID2 register to identify part number.</td>
</tr>
<tr>
<td>REV4:REV0</td>
<td>R</td>
<td>DEVID1/18CXXX</td>
<td>These bits are used to indicate the revision of the device.</td>
</tr>
</tbody>
</table>

Legend: R = readable, P = programmable, U = unimplemented, read as '0',<br>- n = value when device is unprogrammed, u = unchanged.
3.2 Embedding Configuration Word Information in the HEX File

To allow portability of code, a PIC18CXXX programmer is required to read the configuration word locations from the HEX file when loading the HEX file. If configuration word information was not present in the HEX file, then a simple warning message may be issued. Similarly, while saving a HEX file, all configuration word information must be included. An option to not include the configuration word information may be provided. When embedding configuration word information in the HEX file, it should be to address FE00h.

Microchip Technology Inc. feels strongly that this feature is important for the benefit of the end customer.

3.3 Checksum Computation

The checksum is calculated by summing the following:

- The contents of all program memory locations
- The configuration word, appropriately masked
- Masked ID locations (when applicable)

The Least Significant 16 bits of this sum are the checksum.

Table 3-5 describes how to calculate the checksum for each device. Note that the checksum calculation differs depending on the code protect setting. Since the program memory locations read out differently, depending on the code protect setting, the table describes how to manipulate the actual program memory values to simulate the values that would be read from a protected device. When calculating a checksum by reading a device, the entire program memory can simply be read and summed. The configuration word and ID locations can always be read.

Note that some older devices have an additional value added in the checksum. This is to maintain compatibility with older device programmer checksums.

Note: The checksum computations are shown only for devices with on-chip EPROM (i.e., PIC18CXX2 and PIC18CXX8 devices). Because PIC18C601/801 devices do not have on-chip EPROM, no formulas are shown for them. The decision to implement a checksum for these devices, as well as the details of the checksum scheme, are left to the discretion of the user.
# TABLE 3-5: CHECKSUM COMPUTATION

<table>
<thead>
<tr>
<th>Device</th>
<th>Code Protect</th>
<th>Checksum</th>
<th>Blank Value</th>
<th>0xAA at 0 and Max Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC18C242</td>
<td>Disabled</td>
<td>SUM[0x0000:0x3FFF] + CONFIG1L &amp; 0xFF + CONFIG1H &amp; 0x27 + CONFIG2L &amp; 0x0F + CONFIG2H &amp; 0x0F + CONFIG3H &amp; 0x01 + CONFIG4L &amp; 0x01</td>
<td>0xC146</td>
<td>0xC09C</td>
</tr>
<tr>
<td></td>
<td>Enabled</td>
<td>CONFIG1L &amp; 0xFF + CONFIG1H &amp; 0x27 + CONFIG2L &amp; 0x0F + CONFIG2H &amp; 0x0F + CONFIG3H &amp; 0x01 + CONFIG4L &amp; 0x01 + SUM_ID</td>
<td>0x005E</td>
<td>0x0068</td>
</tr>
<tr>
<td>PIC18C252</td>
<td>Disabled</td>
<td>SUM[0x0000:0x7FFF] + CONFIG1L &amp; 0xFF + CONFIG1H &amp; 0x27 + CONFIG2L &amp; 0x0F + CONFIG2H &amp; 0x0F + CONFIG3H &amp; 0x01 + CONFIG4L &amp; 0x01</td>
<td>0x8146</td>
<td>0x809C</td>
</tr>
<tr>
<td></td>
<td>Enabled</td>
<td>CONFIG1L &amp; 0xFF + CONFIG1H &amp; 0x27 + CONFIG2L &amp; 0x0F + CONFIG2H &amp; 0x0F + CONFIG3H &amp; 0x01 + CONFIG4L &amp; 0x01 + SUM_ID</td>
<td>0x005A</td>
<td>0x0064</td>
</tr>
<tr>
<td>PIC18C442</td>
<td>Disabled</td>
<td>SUM[0x0000:0x3FFF] + CONFIG1L &amp; 0xFF + CONFIG1H &amp; 0x27 + CONFIG2L &amp; 0x0F + CONFIG2H &amp; 0x0F + CONFIG3H &amp; 0x01 + CONFIG4L &amp; 0x01</td>
<td>0xC146</td>
<td>0xC09C</td>
</tr>
<tr>
<td></td>
<td>Enabled</td>
<td>CONFIG1L &amp; 0xFF + CONFIG1H &amp; 0x27 + CONFIG2L &amp; 0x0F + CONFIG2H &amp; 0x0F + CONFIG3H &amp; 0x01 + CONFIG4L &amp; 0x01 + SUM_ID</td>
<td>0x005E</td>
<td>0x0068</td>
</tr>
<tr>
<td>PIC18C452</td>
<td>Disabled</td>
<td>SUM[0x0000:0x7FFF] + CONFIG1L &amp; 0xFF + CONFIG1H &amp; 0x27 + CONFIG2L &amp; 0x0F + CONFIG2H &amp; 0x0F + CONFIG3H &amp; 0x01 + CONFIG4L &amp; 0x01</td>
<td>0x8146</td>
<td>0x809C</td>
</tr>
<tr>
<td></td>
<td>Enabled</td>
<td>CONFIG1L &amp; 0xFF + CONFIG1H &amp; 0x27 + CONFIG2L &amp; 0x0F + CONFIG2H &amp; 0x0F + CONFIG3H &amp; 0x01 + CONFIG4L &amp; 0x01 + SUM_ID</td>
<td>0x005A</td>
<td>0x0064</td>
</tr>
<tr>
<td>PIC18C558</td>
<td>Disabled</td>
<td>SUM[0x0000:0x7FFF] + CONFIG1L &amp; 0xFF + CONFIG1H &amp; 0x27 + CONFIG2L &amp; 0x0F + CONFIG2H &amp; 0x0F + CONFIG4L &amp; 0x01</td>
<td>0x8145</td>
<td>0x809B</td>
</tr>
<tr>
<td></td>
<td>Enabled</td>
<td>CONFIG1L &amp; 0xFF + CONFIG1H &amp; 0x27 + CONFIG2L &amp; 0x0F + CONFIG2H &amp; 0x0F + CONFIG4L &amp; 0x01</td>
<td>0x0058</td>
<td>0x0062</td>
</tr>
<tr>
<td>PIC18C858</td>
<td>Disabled</td>
<td>SUM[0x0000:0x7FFF] + CONFIG1L &amp; 0xFF + CONFIG1H &amp; 0x27 + CONFIG2L &amp; 0x0F + CONFIG2H &amp; 0x0F + CONFIG4L &amp; 0x01</td>
<td>0x8145</td>
<td>0x809B</td>
</tr>
<tr>
<td></td>
<td>Enabled</td>
<td>CONFIG1L &amp; 0xFF + CONFIG1H &amp; 0x27 + CONFIG2L &amp; 0x0F + CONFIG2H &amp; 0x0F + CONFIG4L &amp; 0x01</td>
<td>0x0058</td>
<td>0x0062</td>
</tr>
</tbody>
</table>

Legend: **Item** | **Description**
--- | ---
CFGW = Configuration Word
SUM[a:b] = Sum of locations a to b inclusive
SUM_ID = Byte-wise sum of lower four bits of all customer ID locations
+ = Addition
& = Bitwise AND
# 4.0 AC/DC CHARACTERISTICS

## TIMING REQUIREMENTS FOR PROGRAM/VERIFY TEST MODE

<table>
<thead>
<tr>
<th>Param No.</th>
<th>Sym</th>
<th>Characteristic</th>
<th>Min</th>
<th>Typ†</th>
<th>Max</th>
<th>Units</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIHH</td>
<td>Programming Voltage on VPP/MCLR pin</td>
<td>12.75</td>
<td>—</td>
<td>13.25</td>
<td>V</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>IPP</td>
<td>Programming current on MCLR pin</td>
<td>18CXX2/XX8</td>
<td>—</td>
<td>25</td>
<td>50</td>
<td>mA</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>18C601/801</td>
<td>—</td>
<td>.5</td>
<td>1</td>
<td>mA</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>TSER</td>
<td>Serial setup time</td>
<td>20</td>
<td>—</td>
<td>—</td>
<td>ns</td>
<td>—</td>
</tr>
<tr>
<td>P2</td>
<td>TSCLK</td>
<td>Serial clock period</td>
<td>100</td>
<td>—</td>
<td>—</td>
<td>ns</td>
<td>—</td>
</tr>
<tr>
<td>P3</td>
<td>TSET1</td>
<td>Input Data Setup Time to serial clock ↓</td>
<td>15</td>
<td>—</td>
<td>—</td>
<td>ns</td>
<td>—</td>
</tr>
<tr>
<td>P4</td>
<td>THLD1</td>
<td>Input Data Hold Time from serial clock ↓</td>
<td>15</td>
<td>—</td>
<td>—</td>
<td>ns</td>
<td>—</td>
</tr>
<tr>
<td>P5</td>
<td>TDLY1</td>
<td>Delay between last clock ↓ to first clock ↑ of next command</td>
<td>20</td>
<td>—</td>
<td>—</td>
<td>ns</td>
<td>—</td>
</tr>
<tr>
<td>P6</td>
<td>TDLY2</td>
<td>Delay between last clock ↓ of command byte to first clock ↑ of read of data word</td>
<td>20</td>
<td>—</td>
<td>—</td>
<td>ns</td>
<td>—</td>
</tr>
<tr>
<td>P8</td>
<td>TDLY4</td>
<td>Data input not driven to next clock input</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>ns</td>
<td>—</td>
</tr>
<tr>
<td>P9</td>
<td>TDLY5</td>
<td>RB6 high time (minimum programming time)</td>
<td>18CXX2/XX8</td>
<td>100</td>
<td>—</td>
<td>—</td>
<td>μs</td>
</tr>
<tr>
<td></td>
<td>18C601/801</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>ms</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>P10</td>
<td>TDLY6</td>
<td>RB6 low time after programming (high voltage discharge time)</td>
<td>18CXX2/XX8</td>
<td>100</td>
<td>—</td>
<td>—</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td>18C601/801</td>
<td>5</td>
<td>—</td>
<td>—</td>
<td>μs</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>P14</td>
<td>TVALID</td>
<td>Data out valid from SCLK ↑</td>
<td>10</td>
<td>—</td>
<td>—</td>
<td>ns</td>
<td>—</td>
</tr>
</tbody>
</table>

† Data in “Typ” column is at 5V, 25°C, unless otherwise stated.
1.0 PROGRAMMING THE PIC16F8X

The PIC16F8X devices are programmed using a serial method. The Serial mode will allow these devices to be programmed while in the user's system. This allows for increased design flexibility. This programming specification applies to only the above devices in all packages.

1.1 Hardware Requirements

The PIC16F8X devices require one programmable power supply for VDD (4.5V to 5.5V) and a VPP of 12V to 14V. Both supplies should have a minimum resolution of 0.25V.

1.2 Programming Mode

The Programming mode for the PIC16F8X devices allows programming of user program memory, data memory, special locations used for ID, and the configuration word. On PIC16CR8X devices, only data EEPROM and CDP can be programmed.

### Pin Diagram

![Pic16f8x Pin Diagram](image)

### TABLE 1-1: PIN DESCRIPTIONS (DURING PROGRAMMING): PIC16F8X

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>During Programming</th>
<th>Pin Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RB6</td>
<td>CLOCK</td>
<td>Clock Input</td>
</tr>
<tr>
<td>RB7</td>
<td>DATA</td>
<td>Data Input/Output</td>
</tr>
<tr>
<td>MCLR</td>
<td>VTEST MODE</td>
<td>Program Mode Select</td>
</tr>
<tr>
<td>VDD</td>
<td></td>
<td>Power Supply</td>
</tr>
<tr>
<td>VSS</td>
<td></td>
<td>Ground</td>
</tr>
</tbody>
</table>

Legend: I = Input, O = Output, P = Power

Note 1: In the PIC16F8X, the programming high voltage is internally generated. To activate the Programming mode, high voltage needs to be applied to MCLR input. Since the MCLR is used for a level source, this means that MCLR does not draw any significant current.
2.0 PROGRAM MODE ENTRY

2.1 User Program Memory Map

The user memory space extends from 0000h to 1FFFh (8 Kbytes), of which 1 Kbyte (0000h - 03FFh) is physically implemented. In actual implementation, the on-chip user program memory is accessed by the lower 10 bits of the PC, with the upper 3 bits of the PC ignored. Therefore, if the PC is greater than 03FFh, it will wrap around and address a location within the physically implemented memory (see Figure 2-1).

In Programming mode, the program memory space extends from 0000h to 3FFFh, with the first half (0000h-1FFFh) being user program memory and the second half (2000h-3FFFh) being configuration memory. The PC will increment from 0000h to 1FFFh and wrap to 0000h, or 2000h to 3FFFh and wrap around to 2000h (not to 0000h). Once in configuration memory, the highest bit of the PC stays a ‘1’, thus always pointing to the configuration memory. The only way to point to user program memory is to reset the part and re-enter Program/Verify mode, as described in Section 2.3.

In the configuration memory space, 2000h-200Fh are physically implemented. However, only locations 2000h through 2007h are available. Other locations are reserved. Locations beyond 2000Fh will physically access user memory (see Figure 2-1).

FIGURE 2-1: PROGRAM MEMORY MAPPING
2.2 ID Locations

A user may store identification information (ID) in four ID locations, mapped in addresses 2000h through 2003h. It is recommended that the user use only the four Least Significant bits of each ID location. The ID locations read out in an unscrambled fashion after code protection is enabled. It is recommended that ID location is written as "11 1111 1000 bbbb", where "bbbb" is ID information.

2.3 Program/Verify Mode

The Program/Verify mode is entered by holding pins RB6 and RB7 low, while raising MCLR pin from VIH to VIHH (high voltage). Once in this mode, the user program memory and the configuration memory can be accessed and programmed in serial fashion. RB6 and RB7 are Schmitt Trigger inputs in this mode.

Note: Do not allow excess time when transitioning MCLR between VIH and VIHH; this can cause spurious program executions to occur. The maximum transition time is

\[ T_{CY} + T_{PWRT} \ (\text{if enabled}) + 1024 \ T_{OSC} \ (\text{for LP, HS and XT modes only}) \]

where \( T_{CY} \) is the Instruction Cycle Time, \( T_{PWRT} \) is the Power-up Timer Period, and \( T_{OSC} \) is the Oscillator Period (all values in \( \mu \)s or ns).

For specific values, refer to the Electrical Characteristics section of the Device Data Sheet for the particular device.

The sequence that enters the device into the Programming/Verify mode places all other logic into the RESET state (the MCLR pin was initially at VIH). This means that all I/O are in the RESET state (high impedance inputs).

The normal sequence for programming is to use the load data command to set a value to be written at the selected address. Issue the "begin programming command" followed by "read data command" to verify and then, increment the address.

2.3.1 SERIAL PROGRAM/VERIFY OPERATION

The RB6 pin is used as a clock input pin, and the RB7 pin is used for entering command bits and data input/output during serial operation. To input a command, the clock pin (RB6) is cycled six times. Each command bit is latched on the falling edge of the clock with the Least Significant bit (LSb) of the command being input first. The data on pin RB7 is required to have a minimum setup and hold time (see AC/DC specifications in Table 5-1), with respect to the falling edge of the clock. Commands that have data associated with them (read and load) are specified to have a minimum delay of 1 \( \mu \)s between the command and the data. After this delay, the clock pin is cycled 16 times with the first cycle being a START bit and the last cycle being a STOP bit. Data is also input and output LSb first.

Therefore, during a read operation, the LSb will be transmitted onto pin RB7 on the rising edge of the second cycle, and during a load operation, the LSb will be latched on the falling edge of the second cycle. A minimum 1 \( \mu \)s delay is also specified between consecutive commands.

All commands are transmitted LSb first. Data words are also transmitted LSb first. The data is transmitted on the rising edge and latched on the falling edge of the clock. To allow for decoding of commands and reversal of data pin configuration, a time separation of at least 1 \( \mu \)s is required between a command and a data word (or another command).

The available commands (Load Configuration and Load Data for Program Memory) are discussed in the following sections.
2.3.1.1 Load Configuration
After receiving this command, the program counter (PC) will be set to 2000h. By then applying 16 cycles to the clock pin, the chip will load 14-bits in a “data word,” as described above, to be programmed into the configuration memory. A description of the memory mapping schemes of the program memory for normal operation and Configuration mode operation is shown in Figure 2-1. After the configuration memory is entered, the only way to get back to the user program memory is to exit the Program/Verify Test mode by taking MCLR below VIL.

2.3.1.2 Load Data for Program Memory
After receiving this command, the chip will load in a 14-bit “data word” when 16 cycles are applied, as described previously. A timing diagram for the load data command is shown in Figure 5-1.

---

### TABLE 2-1: COMMAND MAPPING FOR PIC16F83/CR83/F84/CR84

<table>
<thead>
<tr>
<th>Command</th>
<th>Mapping (MSb ... LSb)</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Configuration</td>
<td>000000 0, 0, 0, 0, 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Load Data for Program Memory</td>
<td>000010 0, 0, 0, 1, 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Read Data from Program Memory</td>
<td>000100 0, 0, 1, 0, 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Increment Address</td>
<td>000110</td>
<td></td>
</tr>
<tr>
<td>Begin Erase Programming Cycle</td>
<td>001000</td>
<td></td>
</tr>
<tr>
<td>Load Data for Data Memory</td>
<td>000011 0, 0, 0, 1, 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Read Data from Data Memory</td>
<td>000101 0, 0, 0, 1, 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Bulk Erase Program Memory</td>
<td>000111 0, 0, 0, 1, 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Bulk Erase Data Memory</td>
<td>000010 0, 0, 1, 0, 0</td>
<td>0, data (14), 0</td>
</tr>
</tbody>
</table>

### TABLE 2-2: COMMAND MAPPING FOR PIC16F84A

<table>
<thead>
<tr>
<th>Command</th>
<th>Mapping (MSb ... LSb)</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Configuration</td>
<td>X X 0 0 0, 0, 0, 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Load Data for Program Memory</td>
<td>X X 0 0 1, 0, 0, 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Read Data from Program Memory</td>
<td>X X 0 1 0, 0, 0, 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Increment Address</td>
<td>X X 0 1 1, 0, 0, 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Begin Erase Programming Cycle</td>
<td>001000</td>
<td></td>
</tr>
<tr>
<td>Begin Programming Only Cycle</td>
<td>011000</td>
<td></td>
</tr>
<tr>
<td>Load Data for Data Memory</td>
<td>X X 0 0 1, 1, 0, 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Read Data from Data Memory</td>
<td>X X 0 1 0, 1, 0, 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Bulk Erase Program Memory</td>
<td>X X 1 0 0, 0, 0, 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Bulk Erase Data Memory</td>
<td>X X 1 0 1, 0, 0, 0</td>
<td>0, data (14), 0</td>
</tr>
</tbody>
</table>
FIGURE 2-2: PROGRAM FLOW CHART - PIC16F8X PROGRAM MEMORY

Start

Set VDD = VDDP

Program Cycle

Read Data Command

Data Correct?

Result: Programming Failure

Report Verify Error @ VDDMIN

Data Correct?

Verify all Locations @ VDDMIN

Result: Programming Failure

Verify all Locations @ VDDMAX

Result: Programming Failure

Report Verify Error @ VDDMAX

All Locations Done?

No: Increment Address Command

Yes: Wait 8 ms - PIC16F84A

Wait 20 ms - All Others

Program Cycle

Load Data Command

Begin Programming Command

Wait 8 ms - PIC16F84A

Wait 20 ms - All Others

Done
FIGURE 2-3: PROGRAM FLOW CHART - PIC16F8X CONFIGURATION MEMORY

1. Start
2. Load Configuration Data
3. Program ID Location?
   - Yes: Program Cycle
   - No: Increment Address Command
4. Address = 0x2004?
   - Yes: Increment Address Command
   - No: Address = 0x2004?
5. Increment Address Command
6. Increment Address Command
7. Increment Address Command
8. Program Cycle (Config. Word)
9. Set VDD = VDDMAX
10. Report Program Configuration Word Error
    - No: Data Correct?
    - Yes: Set VDD = VDDMAX
11. Data Correct?
    - Yes: Read Data Command
    - No: Done
12. Data Correct?
    - Yes: Read Data Command
    - No: Done
2.3.1.3 Load Data for Data Memory
After receiving this command, the chip will load in a 14-bit “data word” when 16 cycles are applied. However, the data memory is only 8-bits wide, and thus, only the first 8-bits of data after the START bit will be programmed into the data memory. It is still necessary to cycle the clock the full 16 cycles, in order to allow the internal circuitry to reset properly. The data memory contains 64 words. Only the lower 8 bits of the PC are decoded by the data memory, and therefore, if the PC is greater than 0x3F, it will wrap around and address a location within the physically implemented memory.

2.3.1.4 Read Data from Program Memory
After receiving this command, the chip will transmit data bits out of the program memory (user or configuration) currently accessed, starting with the second rising edge of the clock input. The RB7 pin will go into Output mode on the second rising clock edge, and it will revert back to Input mode (hi-impedance) after the 16th rising edge. A timing diagram of this command is shown in Figure 5-2.

2.3.1.5 Read Data from Data Memory
After receiving this command, the chip will transmit data bits out of the data memory starting with the second rising edge of the clock input. The RB7 pin will go into Output mode on the second rising edge, and it will revert back to Input mode (hi-impedance) after the 16th rising edge. As previously stated, the data memory is 8-bits wide, and therefore, only the first 8 bits that are output are actual data.

2.3.1.6 Increment Address
The PC is incremented when this command is received. A timing diagram of this command is shown in Figure 5-3.

2.3.1.7 Begin Erase/Program Cycle
A load command must be given before every begin programming command. Programming of the appropriate memory (configuration memory, user program memory or data memory) will begin after this command is received and decoded. An internal timing mechanism executes an erase before write. The user must allow for both erase and programming cycle times for programming to complete. No “end programming” command is required.

2.3.1.8 Begin Programming
This command is available only on the PIC16F84A. A load command must be given before every begin programming command. Programming of the appropriate memory (configuration memory, user program memory or data memory) will begin after this command is received and decoded. An internal timing mechanism executes a write. The user must allow for program cycle time for programming to complete. No “end programming” command is required.

This command is similar to the ERASE/PROGRAM CYCLE command, except that a word erase is not done. It is recommended that a bulk erase be performed before starting a series of programming only cycles.
2.3.1.9 Bulk Erase Program Memory

After this command is performed, the next program command will erase the entire program memory.

To perform a bulk erase of the program memory, the following sequence must be performed.

For PIC16F84A, perform the following commands:
1. Do a “Load Data All ‘1’s” command
2. Do a “Bulk Erase User Memory” command
3. Do a “Begin Programming” command
4. Wait 10 ms to complete bulk erase

If the address is pointing to the configuration memory (2000h - 200Fh), then both the user memory and the configuration memory will be erased. The configuration word will not be erased, even if the address is pointing to location 2007h.

For PIC16CR83/CR84 and PIC16F84, perform the following commands:
1. Issue Command 2 (write program memory)
2. Send out 3FFF data
3. Issue Command 1 (toggle select even rows)
4. Issue Command 7 (toggle select even rows)
5. Issue Command 8 (begin programming)
6. Delay 10 ms
7. Issue Command 1 (toggle select even rows)
8. Issue Command 7 (toggle select even rows)

Note: If the device is code protected (PIC16F84A), the BULK ERASE command will not work.

2.3.1.10 Bulk Erase Data Memory

To perform a bulk erase of the data memory, the following sequence must be performed.

For PIC16F84A, perform the following commands:
1. Do a “Load Data All ‘1’s” command
2. Do a “Bulk Erase Data Memory” command
3. Do a “Begin Programming” command
4. Wait 10 ms to complete bulk erase

For PIC16CR83/CR84 and PIC16F84, perform the following commands:
5. Send out 3FFFH data
6. Issue Command 1 (toggle select even rows)
7. Issue Command 7 (toggle select even rows)
8. Issue Command 8 (begin data)
9. Delay 10 ms
10. Issue Command 1 (toggle select even rows)
11. Issue Command 7 (toggle select even rows)

Note: All BULK ERASE operations must take place at 4.5 to 5.5 VDD range.

2.4 Programming Algorithm Requires Variable VDD

The PIC16F8X devices use an intelligent algorithm. The algorithm calls for program verification at VDDMIN, as well as VDDMAX. Verification at VDDMIN ensures good “erase margin”. Verification at VDDMAX ensures good “program margin”.

The actual programming must be done with VDD in the VDDP range (see Table 5-1):

\[ V\text{DDP} = V\text{CC} \text{ range required during programming} \]
\[ V\text{DDMIN} = \text{minimum operating VDD spec for the part} \]
\[ V\text{DDMAX} = \text{maximum operating VDD spec for the part} \]

Programmers must verify the PIC16F8X devices at their specified VDDMAX and VDDMIN levels. Since Microchip may introduce future versions of the PIC16F8X devices with a broader VDD range, it is best that these levels are user selectable (defaults are acceptable).

Note: Any programmer not meeting these requirements may only be classified as “prototype” or “development” programmer, but not a “production” quality programmer.
## 3.0 CONFIGURATION WORD

Most of the PIC16F8X devices have five configuration bits. These bits can be set (reads '0'), or left unchanged (reads '1') to select various device configurations. Their usage in the Device Configuration Word is shown in Register 3-1.

### 3.1 Device ID Word

The device ID word for the PIC16F84A device is located at 2006h. Older devices do not have device ID.

#### TABLE 3-1: DEVICE ID WORD

<table>
<thead>
<tr>
<th>Device</th>
<th>Device ID Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16F84A</td>
<td>00 0101 011 X XXXX</td>
</tr>
</tbody>
</table>

For PIC16F83/84/84A:

<table>
<thead>
<tr>
<th>CP</th>
<th>CP</th>
<th>CP</th>
<th>CP</th>
<th>CP</th>
<th>CP</th>
<th>CP</th>
<th>CP</th>
<th>CP</th>
<th>CP</th>
<th>PWTREN</th>
<th>WDTEN</th>
<th>FOSC1</th>
<th>FOSC0</th>
</tr>
</thead>
</table>

**bit 13-8:**

**CP:** Code Protection bits

1 = Code protection off

0 = Code protection on

**bit 6-4**

For PIC16F83/84/84A:

**CP:** Code Protection bits

1 = Code protection off

0 = Code protection on

For PIC16CR83/84:

**DP:** Data Memory Code Protection bit

1 = Code protection off

0 = Data memory is code protected

**bit 3**

**PWTREN:** Power-up Timer Enable bit

1 = PWRT disabled

0 = PWRT enabled

**bit 2**

**WDTEN:** Watchdog Timer Enable bit

1 = WDT enabled

0 = WDT disabled

**bit 1-0**

**FOSC1:FOSC0:** Oscillator Selection bits

11 = RC oscillator

10 = HS oscillator

01 = XT oscillator

00 = LP oscillator

---

**Note 1:** All of the CP bits have to be given the same value to enable the code protection scheme listed.
4.0 CODE PROTECTION

For PIC16F8X devices, once code protection is enabled, all program data memory locations read all ‘0’s. The ID locations and the configuration word read out in an unscrambled fashion. Further programming is disabled for the entire program memory as well as data memory. It is possible to program the ID locations and the configuration word.

For PIC16CR8X devices, once code protection is enabled, all program memory locations read all ‘0’s; data memory locations read all ‘1’s.

A description of the code protection schemes for the various PIC16F8X devices is provided on page 157. For each device, the bit configuration for the device configuration word to enable code protection is provided. This is followed with a comparison of read and write operations for selected memory spaces in both protected and unprotected modes.

4.1 Disabling Code Protection

It is recommended that the following procedure be performed before any other programming is attempted. It is also possible to turn code protection off (code protect bit = ‘1’) using this procedure; however, all data within the program memory and the data memory will be erased when this procedure is executed, and thus, the security of the data or code is not compromised.

Procedure to disable code protect:

1. Execute load configuration (with a ‘1’ in bits 4-13, code protect)
2. Increment to configuration word location (2007h)
3. Execute command (000001)
4. Execute command (000111)
5. Execute ‘Begin Programming’ (001000)
6. Wait 10 ms
7. Execute command (000001)
8. Execute command (000111)

4.2 Embedding Configuration Word and ID Information in the HEX File

Note: To allow portability of code, the programmer is required to read the configuration word and ID locations from the HEX file when loading the HEX file. If configuration word information was not present in the HEX file, then a simple warning message may be issued. Similarly, while saving a HEX file, configuration word and ID information must be included. An option to not include this information may be provided. Specifically for the PIC16F8X, the EEPROM data memory should also be embedded in the HEX file (see Section 5.1). Microchip Technology Inc. feels strongly that this feature is important for the benefit of the end customer.
### Device: PIC16F83

**To code protect:** 0000000000XXXX

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (2007h)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>All memory</td>
<td>Read All '0's, Write Disabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>ID Locations [2000h : 2003h]</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
</tbody>
</table>

### Device: PIC16CR83

**To code protect:** 0000000000XXXX

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (2007h)</td>
<td>Read Unscrambled</td>
<td>Read Unscrambled</td>
</tr>
<tr>
<td>All memory</td>
<td>Read All '0's for Program Memory, Read All '1's for Data Memory - Write Disabled</td>
<td>Read Unscrambled, Data Memory - Write Enabled</td>
</tr>
<tr>
<td>ID Locations [2000h : 2003h]</td>
<td>Read Unscrambled</td>
<td>Read Unscrambled</td>
</tr>
</tbody>
</table>

### Device: PIC16CR84

**To code protect:** 0000000000XXXX

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (2007h)</td>
<td>Read Unscrambled</td>
<td>Read Unscrambled</td>
</tr>
<tr>
<td>All memory</td>
<td>Read All '0's for Program Memory, Read All '1's for Data Memory - Write Disabled</td>
<td>Read Unscrambled, Data Memory - Write Enabled</td>
</tr>
<tr>
<td>ID Locations [2000h : 2003h]</td>
<td>Read Unscrambled</td>
<td>Read Unscrambled</td>
</tr>
</tbody>
</table>

### Device: PIC16F84

**To code protect:** 0000000000XXXX

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (2007h)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>All memory</td>
<td>Read All '0's, Write Disabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>ID Locations [2000h : 2003h]</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
</tbody>
</table>

### Device: PIC16F84A

**To code protect:** 0000000000XXXX

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (2007h)</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>All memory</td>
<td>Read All '0's, Write Disabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>ID Locations [2000h : 2003h]</td>
<td>Read Unscrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
</tbody>
</table>

Legend: \(x = \) Don’t care
4.3 Checksum Computation

4.3.1 CHECKSUM

Checksum is calculated by reading the contents of the PIC16F8X memory locations and adding up the opcodes, up to the maximum user addressable location, e.g., 1FFh for the PIC16F83. Any carry bits exceeding 16-bits are neglected. Finally, the configuration word (appropriately masked) is added to the checksum. Checksum computation for each member of the PIC16F8X devices is shown in Table 4-1.

The checksum is calculated by summing the following:
- The contents of all program memory locations
- The configuration word, appropriately masked
- Masked ID locations (when applicable)

The Least Significant 16 bits of this sum are the checksum.

The following table describes how to calculate the checksum for each device. Note that the checksum calculation differs depending on the code protect setting. Since the program memory locations read out differently depending on the code protect setting, the table describes how to manipulate the actual program memory values to simulate the values that would be read from a protected device. When calculating a checksum by reading a device, the entire program memory can simply be read and summed. The configuration word and ID locations can always be read.

Note that some older devices have an additional value added in the checksum. This is to maintain compatibility with older device programmer checksums.

### TABLE 4-1: CHECKSUM COMPUTATION

<table>
<thead>
<tr>
<th>Device</th>
<th>Code Protect</th>
<th>Checksum*</th>
<th>Blank Value</th>
<th>25E6h at 0 and Max Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16F83</td>
<td>OFF</td>
<td>SUM[000h:1FFh] + CFGW &amp; 3FFFh</td>
<td>3DFFh</td>
<td>09CDh</td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>CFGW &amp; 3FFFh + SUM_ID</td>
<td>3E0Eh</td>
<td>09DCh</td>
</tr>
<tr>
<td>PIC16CR83</td>
<td>OFF</td>
<td>SUM[000h:1FFh] + CFGW &amp; 3FFFh</td>
<td>3DFFh</td>
<td>09CDh</td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>CFGW &amp; 3FFFh + SUM_ID</td>
<td>3E0Eh</td>
<td>09DCh</td>
</tr>
<tr>
<td>PIC16F84</td>
<td>OFF</td>
<td>SUM[000h:3FFh] + CFGW &amp; 3FFFh</td>
<td>3BFFh</td>
<td>07CDh</td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>CFGW &amp; 3FFFh + SUM_ID</td>
<td>3C0Eh</td>
<td>07DCh</td>
</tr>
<tr>
<td>PIC16CR84</td>
<td>OFF</td>
<td>SUM[000h:3FFh] + CFGW &amp; 3FFFh</td>
<td>3BFFh</td>
<td>07CDh</td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>CFGW &amp; 3FFFh + SUM_ID</td>
<td>3C0Eh</td>
<td>07DCh</td>
</tr>
<tr>
<td>PIC16F84A</td>
<td>OFF</td>
<td>SUM[000h:3FFh] + CFGW &amp; 3FFFh</td>
<td>3BFFh</td>
<td>07CDh</td>
</tr>
<tr>
<td></td>
<td>ON</td>
<td>CFGW &amp; 3FFFh + SUM_ID</td>
<td>3C0Eh</td>
<td>07DCh</td>
</tr>
</tbody>
</table>

Legend: CFGW = Configuration Word
SUM[a:b] = [Sum of locations a to b inclusive]
SUM_ID = ID locations masked by 0xF then made into a 16-bit value with ID0 as the most significant nibble.
For example, ID0 = 01h, ID1 = 02h, ID3 = 03h, ID4 = 04h, then SUM_ID = 1234h.

*Checksum = [Sum of all the individual expressions] MODULO [FFFFh]
+ = Addition
& = Bitwise AND
5.0 PROGRAM/VERIFY MODE ELECTRICAL CHARACTERISTICS

5.1 Embedding Data EEPROM Contents in HEX File

The programmer should be able to read data EEPROM information from a HEX file and conversely (as an option), write data EEPROM contents to a HEX file, along with program memory information and fuse information.

The 64 data memory locations are logically mapped, starting at address 2100h. The format for data memory storage is one data byte per address location, LSB aligned.

### TABLE 5-1: AC/DC CHARACTERISTICS

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VDDP</td>
<td>Supply voltage during programming</td>
<td>4.5</td>
<td>5.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VDDV</td>
<td>Supply voltage during verify</td>
<td>VDDMIN</td>
<td>VDDMAX</td>
<td>V</td>
<td>(Note 1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VIH</td>
<td>High voltage on MCLR for Test mode entry</td>
<td>12</td>
<td>14.0</td>
<td>V</td>
<td>(Note 2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IDD</td>
<td>Supply current (from VDD) during program/verify</td>
<td>50</td>
<td>V</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IHH</td>
<td>Supply current from VIH (on MCLR)</td>
<td>200</td>
<td>µA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Vih1</td>
<td>(RB6, RB7) input high level</td>
<td>0.8 VDD</td>
<td>V</td>
<td></td>
<td>Schmitt Trigger input</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Vil1</td>
<td>(RB6, RB7) input low level MCLR (Test mode selection)</td>
<td>0.2 VDD</td>
<td>V</td>
<td></td>
<td>Schmitt Trigger input</td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>TVHHR</td>
<td>MCLR rise time (VIH to VIH) for Test mode entry</td>
<td>8.0</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>Tset0</td>
<td>RB6, RB7 setup time (before pattern setup time)</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>Tset1</td>
<td>Data in setup time before clock ↓</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>Thld1</td>
<td>Data in hold time after clock ↓</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>Tdly1</td>
<td>Data input not driven to next clock input (delay required between command/data or command/command)</td>
<td>1.0</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>Tdly2</td>
<td>Delay between clock ↓ to clock ↑ of next command or data</td>
<td>1.0</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td>Tdly3</td>
<td>Clock to data out valid (during read data)</td>
<td>80</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td>Thld0</td>
<td>RB&lt;7:6&gt; hold time after MCLR ↑</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>—</td>
<td>Erase cycle time</td>
<td>—</td>
<td>—</td>
<td>4</td>
<td>ms</td>
<td>PIC16F84A only</td>
</tr>
<tr>
<td></td>
<td>—</td>
<td>Program cycle time</td>
<td>—</td>
<td>—</td>
<td>4</td>
<td>ms</td>
<td>PIC16F84A only</td>
</tr>
<tr>
<td></td>
<td>—</td>
<td>Erase and program time</td>
<td>—</td>
<td>—</td>
<td>8</td>
<td>ms</td>
<td>PIC16F84A only</td>
</tr>
</tbody>
</table>

**Note 1:** Program must be verified at the minimum and maximum VDD limits for the part.

**Note 2:** VihH must be greater than VDD + 4.5V to stay in Programming/Verify mode.
This document includes the programming specifications for the following devices:

- PIC16F627
- PIC16F628
- PIC16LF627
- PIC16LF628

**1.0 PROGRAMMING THE PIC16F62X**

The PIC16F62X is programmed using a serial method. The Serial mode will allow the PIC16F62X to be programmed while in the users system. This allows for increased design flexibility. This programming specification applies to PIC16F62X devices in all packages.

**1.1 Hardware Requirements**

The PIC16F62X requires one programmable power supply for V\text{DD} (4.5V to 5.5V) and a V\text{PP} of 12V to 14V, or V\text{PP} of 4.5V to 5.5V, when using low voltage. Both supplies should have a minimum resolution of 0.25V.

**1.2 Programming Algorithm Requires Variable \text{VDD}**

The PIC16F62X uses an intelligent algorithm. The algorithm calls for program verification at \text{VDDMIN} as well as \text{VDDMAX}. Verification at \text{VDDMIN} ensures good “erase margin”. Verification at \text{VDDMAX} ensures good “program margin”.

The actual programming must be done with \text{VDD} in the \text{VDDP} range.

\text{VDDP} = \text{VCC} range required during programming.

\text{VDDMIN} = minimum operating \text{VDD} spec for the part.

\text{VDDMAX} = maximum operating \text{VDD} spec for the part.

Programmers must verify the PIC16F62X is at its specified \text{VDDMAX} and \text{VDDMIN} levels. Since Microchip may introduce future versions of the PIC16F62X with a broader \text{VDD} range, it is best that these levels are user selectable (defaults are ok).

**1.3 Programming Mode**

The Programming mode for the PIC16F62X allows programming of user program memory, data memory, special locations used for ID, and the configuration word.

---

**Note:** All references to PIC16F62X also apply to PIC16LF62X.

**Note:** Any programmer not meeting these requirements may only be classified as a “prototype” or “development” programmer, not a “production” quality programmer.
### Pin Diagram

#### PDIP, SOIC

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA2/AN2/VREF</td>
<td>1</td>
</tr>
<tr>
<td>RA3/AN3/CMP1</td>
<td>2</td>
</tr>
<tr>
<td>RA4/T0CKI/CMP2</td>
<td>3</td>
</tr>
<tr>
<td>RA5/MCLR/THV</td>
<td>4</td>
</tr>
<tr>
<td>VSS</td>
<td>5</td>
</tr>
<tr>
<td>RB0/INT</td>
<td>6</td>
</tr>
<tr>
<td>RB1/RX/DT</td>
<td>7</td>
</tr>
<tr>
<td>RB2/TX/CLK</td>
<td>8</td>
</tr>
<tr>
<td>RB3/CCP1</td>
<td>9</td>
</tr>
<tr>
<td>RA1/AN1</td>
<td>10</td>
</tr>
<tr>
<td>RA0/AN0</td>
<td>11</td>
</tr>
<tr>
<td>RA7/OSC1/CLKIN</td>
<td>12</td>
</tr>
<tr>
<td>RA6/OSC2/CLKOUT</td>
<td>13</td>
</tr>
<tr>
<td>VDD</td>
<td>14</td>
</tr>
<tr>
<td>RB7/DATA/T1OSI</td>
<td>15</td>
</tr>
<tr>
<td>RB6/CLOCK/T1OSO/T1CKI</td>
<td>16</td>
</tr>
<tr>
<td>RB5</td>
<td>17</td>
</tr>
<tr>
<td>RB4/PGM</td>
<td>18</td>
</tr>
</tbody>
</table>

#### SSOP

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA2/AN2/VREF</td>
<td>1</td>
</tr>
<tr>
<td>RA3/AN3/CMP1</td>
<td>2</td>
</tr>
<tr>
<td>RA4/T0CKI/CMP2</td>
<td>3</td>
</tr>
<tr>
<td>RA5/MCLR/THV</td>
<td>4</td>
</tr>
<tr>
<td>VSS</td>
<td>5</td>
</tr>
<tr>
<td>VSS</td>
<td>6</td>
</tr>
<tr>
<td>RB0/INT</td>
<td>7</td>
</tr>
<tr>
<td>RB1/RX/DT</td>
<td>8</td>
</tr>
<tr>
<td>RB2/TX/CLK</td>
<td>9</td>
</tr>
<tr>
<td>RB3/CCP1</td>
<td>10</td>
</tr>
<tr>
<td>RA1/AN1</td>
<td>11</td>
</tr>
<tr>
<td>RA0/AN0</td>
<td>12</td>
</tr>
<tr>
<td>RA7/OSC1/CLKIN</td>
<td>13</td>
</tr>
<tr>
<td>RA6/OSC2/CLKOUT</td>
<td>14</td>
</tr>
<tr>
<td>VDD</td>
<td>15</td>
</tr>
<tr>
<td>VDD</td>
<td>16</td>
</tr>
<tr>
<td>RB7/DATA/T1OSI</td>
<td>17</td>
</tr>
<tr>
<td>RB6/CLOCK/T1OSO/T1CKI</td>
<td>18</td>
</tr>
<tr>
<td>RB5</td>
<td>19</td>
</tr>
<tr>
<td>RB4/PGM</td>
<td>20</td>
</tr>
</tbody>
</table>

### TABLE 1-1: PIN DESCRIPTIONS (DURING PROGRAMMING): PIC16F62X

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Function</th>
<th>Pin Type</th>
<th>During Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>RB4</td>
<td>PGM</td>
<td>I</td>
<td>Low Voltage Programming input if configuration bit equals 1</td>
</tr>
<tr>
<td>RB6</td>
<td>CLOCK</td>
<td>I</td>
<td>Clock input</td>
</tr>
<tr>
<td>RB7</td>
<td>DATA</td>
<td>I/O</td>
<td>Data input/output</td>
</tr>
<tr>
<td>MCLR</td>
<td>Programming Mode</td>
<td>P*</td>
<td>Program Mode Select</td>
</tr>
<tr>
<td>VDD</td>
<td>VDD</td>
<td>P</td>
<td>Power Supply</td>
</tr>
<tr>
<td>VSS</td>
<td>VSS</td>
<td>P</td>
<td>Ground</td>
</tr>
</tbody>
</table>

Legend: I = Input, O = Output, P = Power

* In the PIC16F62X, the programming high voltage is internally generated. To activate the Programming mode, high voltage needs to be applied to MCLR input. Since the MCLR is used for a level source, this means that MCLR does not draw any significant current.
2.0  PROGRAM DETAILS

2.1  User Program Memory Map

The user memory space extends from 0x0000 to 0x1FFF. In Programming mode, the program memory space extends from 0x0000 to 0x3FFF, with the first half (0x0000-0x1FFF) being user program memory and the second half (0x2000-0x3FFF) being configuration memory. The PC will increment from 0x0000 to 0x1FFF and wrap to 0x0000, 0x2000 to 0x3FFF and wrap around to 0x2000 (not to 0x0000). Once in configuration memory, the highest bit of the PC stays a '1', thus always pointing to the configuration memory. The only way to point to user program memory is to reset the part and re-enter Program/Verify mode as described in Section 2.3.

In the configuration memory space, 0x2000-0x200F are physically implemented. However, only locations 0x2000 through 0x2007 are available. Other locations are reserved. Locations beyond 0x200F will physically access user memory (See Figure 2-1).

2.2  User ID Locations

A User may store identification information (ID) in four User ID locations. The User ID locations are mapped in [0x2000 : 0x2003]. These locations read out normally, even after the code protection is enabled.

Note 1: All other locations in PICmicro® MCU configuration memory are reserved and should not be programmed.

2: Only the low order 4 bits of the User ID locations may be included in the device checksum. See Section 3.1 for checksum calculation details.

FIGURE 2-1:  PROGRAM MEMORY MAPPING
2.3 Program/Verify Mode

The programming module operates on simple command sequences entered in serial fashion with the data being latched on the falling edge of the clock pulse. The sequences are entered serially, via the clock and data lines, which are Schmitt Trigger in this mode. The general form for all command sequences consists of a 6-bit command and conditionally a 16-bit data word. Both command and data word are clocked LSb first.

The signal on the data pin is required to have a minimum setup and hold time (see AC/DC specifications), with respect to the falling edge of the clock. Commands that have data associated with them (read and load), require a minimum delay of Tdly1 between the command and the data.

The 6-bit command sequences are shown in Table 2-1.

<table>
<thead>
<tr>
<th>Command</th>
<th>Mapping (MSb ... LSb)</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Configuration</td>
<td>X X 0 0 0 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Load Data for Program Memory</td>
<td>X X 0 0 1 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Load Data for Data Memory</td>
<td>X X 0 1 1 0</td>
<td>0, data (8), zero (6), 0</td>
</tr>
<tr>
<td>Increment Address</td>
<td>X X 0 1 1 0</td>
<td></td>
</tr>
<tr>
<td>Read Data from Program Memory</td>
<td>X X 0 1 0 0</td>
<td>0, data (14), 0</td>
</tr>
<tr>
<td>Read Data from Data Memory</td>
<td>X X 0 1 0 1</td>
<td>0, data (8), zero (6), 0</td>
</tr>
<tr>
<td>Begin Erase/Programming Cycle</td>
<td>0 0 1 0 0 0</td>
<td></td>
</tr>
<tr>
<td>Begin Programming Only Cycle</td>
<td>0 1 1 0 0 0</td>
<td></td>
</tr>
<tr>
<td>Bulk Erase Program Memory</td>
<td>X X 1 0 0 1</td>
<td></td>
</tr>
<tr>
<td>Bulk Erase Data Memory</td>
<td>X X 1 0 1 1</td>
<td></td>
</tr>
<tr>
<td>Bulk Erase Setup 1</td>
<td>0 0 0 0 1 1</td>
<td></td>
</tr>
<tr>
<td>Bulk Erase Setup 2</td>
<td>0 0 0 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>
The optional 16-bit data word will either be an input to, or an output from the PICmicro® MCU, depending on the command. Load Data commands will be input, and Read Data commands will be output. The 16-bit data word only contains 14 bits of data to conform to the 14-bit program memory word. The 14 bits are centered within the 16-bit word, padded with a leading and trailing zero.

Program/Verify mode may be entered via one of two methods. High voltage Program/Verify is entered by holding clock and data pins low while raising VPP first, then VDD, as shown in Figure 2-2. Low voltage Program/Verify mode is entered by raising VDD, then MCLR and PGM, as shown in Figure 2-3. The PC will be set to '0' upon entering into Program/Verify mode. The PC can be changed by the execution of either an increment PC command, or a Load Configuration command, which sets the PC to 0x2000.

All other logic is held in the RESET state while in Program/Verify mode. This means that all I/O are in the RESET state (high impedance inputs).

**Note:** PGM should be held low to prevent inadvertent entry into LVP mode.

**Figure 2-2:** ENTERING HIGH VOLTAGE PROGRAM/VERIFY MODE

**Figure 2-3:** ENTERING LOW VOLTAGE PROGRAM/VERIFY MODE

**Note:** If the device is in LVP mode, raising VPP to Vih does not override LVP mode.
2.3.1 LOAD DATA FOR PROGRAM MEMORY

Load data for program memory receives a 14-bit word, and readies it to be programmed at the PC location. See Figure 2-4 for timing details.

FIGURE 2-4: LOAD DATA COMMAND FOR PROGRAM MEMORY

2.3.2 LOAD DATA FOR DATA MEMORY

Load data for data memory receives an 8-bit byte, and readies it to be programmed into data memory at location specified by the lower 7 bits of the PC. Though the data byte is only 8-bits wide, all 16 clock cycles are required to allow the programming module to reset properly.

FIGURE 2-5: LOAD DATA COMMAND FOR DATA MEMORY
2.3.3 LOAD DATA FOR CONFIGURATION MEMORY

The load configuration command advances the PC to the start of configuration memory (0x2000-0x200F). Once it is set to the configuration region, only exiting and re-entering Program/Verify mode will reset PC to the user memory space (see Figure 2-6).

**FIGURE 2-6: LOAD CONFIGURATION**

2.3.4 BEGIN PROGRAMMING ONLY CYCLE

Begin Programming Only Cycle programs the previously loaded word into the appropriate memory (User Program, Data or Configuration memory). A Load command must be given before every Programming command. Programming begins after this command is received and decoded. An internal timing mechanism executes the write. The user must allow for program cycle time before issuing the next command. No “End Programming” command is required.

This command is similar to the Erase/Program command, except that a word erase is not done. It is recommended that a bulk erase be performed before starting a series of programming only cycles.

**FIGURE 2-7: BEGIN PROGRAMMING ONLY CYCLE**
2.3.5 BEGIN ERASE/PROGRAMMING CYCLE

Begin Erase/Programming Cycle erases the word address specified by the PC, and programs the previously loaded word into the appropriate memory (User Program, Data or Configuration memory). A Load command must be given before every Programming command. Erasing and programming begins after this command is received and decoded. An internal timing mechanism executes an erase before the write. The user must allow for both erase and program cycle time before issuing the next command. No “End Programming” command is required.

FIGURE 2-8: BEGIN ERASE/PROGRAMMING CYCLE

2.3.6 INCREMENT ADDRESS

The PC is incremented when this command is received. See Figure 2-9.

FIGURE 2-9: INCREMENT ADDRESS COMMAND (PROGRAM/VERIFY)
2.3.7 READ DATA FROM PROGRAM MEMORY

Read data from program memory reads the word addressed by the PC and transmits it on the data pin during the data phase of the command. This command will report words from either user or configuration memory, depending on the PC setting. The data pin will go into Output mode on the second rising clock edge and revert back to input moved (hi-impedance) after the 16th rising edge.

FIGURE 2-10: READ DATA FROM PROGRAM MEMORY

2.3.8 READ DATA FROM DATA MEMORY

Read data from data memory reads the byte in data memory addressed by the low order 7 bits of PC and transmits it on the data pin during the data phase of the command. The data pin will go into Output mode on the second rising clock edge, and revert back to input moved (hi-impedance) after the 16th rising edge. As only 8 bits are transmitted, the last 8 bits are zero padded.

FIGURE 2-11: READ DATA FROM DATA MEMORY
2.3.9 BULK ERASE SETUP 1 AND BULK ERASE SETUP 2

These commands are used in conjunction to reset the configuration word. See Section 3.3 for details on how to reset the configuration word.

**FIGURE 2-12: BULK ERASE SETUP 1**

**FIGURE 2-13: BULK ERASE SETUP 2**
3.0 COMMON PROGRAMMING TASKS

These programming commands may be combined in several ways, in order to accomplish different programming goals.

3.1 Bulk Erase Program Memory

If the device is not code protected, the program memory can be erased with the Bulk Erase Program Memory command. See Section 3.4 for removing code protection if it is set.

**Note:** All bulk erase operations must take place with VDD between 4.5-5.5V.

To perform a bulk erase of the program memory, the following sequence must be performed:

1. Execute a Load Data for Program Memory with the data word set to all '1's (0x3FFF).
2. Execute a Bulk Erase Program Memory command.
3. Execute a Begin Programming command.
4. Wait Tera for the erase cycle to complete.

If the address is pointing to the ID/configuration word memory (0x2000-0x200F), then both ID locations and program memory will be erased. However, the configuration word will not be cleared by this method.

**Note:** If the device is code protected, the Bulk Erase command will not work.
3.2 Bulk Erase Data Memory

If the device is not data protected, the program memory can be erased with the Bulk Erase Data Memory command. See Section 3.3 for removing code protection, if it is not set.

To perform a bulk erase of the data memory, the following sequence must be performed:

1. Execute a Load Data for Data Memory with the data word set to all '1's (0x3FFF).
2. Execute a Bulk Erase Data Memory command.
3. Execute a Begin Programming command.
4. Wait Tera for the erase cycle to complete.

![Figure 3-2: Bulk Erase Data Memory Command](image)

Note: All bulk erase operations must take place with VDD between 4.5-5.5V

Note: If the device is code protected, the Bulk Erase command will not work.

3.3 Disabling Code Protection

Once the device has been code protected, the code protected regions of program memory read out as zeros and the device may no longer be written until the following process has been completed. The Bulk Erase commands will not erase the device. Instead, the following procedure, to reset the code protection bits, must be used. Resetting the Code Protection bits will also erase Program, Data and Configuration memory, thus maintaining security of the code and data.

1. Execute a Load Configuration command (data word 0x0000) to set PC to 0x2000.
2. Execute Increment Address command 7 times to advance PC to 0x2007.
3. Execute Bulk Erase Setup 1 command.
4. Execute Bulk Erase Setup 2 command.
5. Execute Begin Erase Programming command.
7. Execute Bulk Erase Setup 1 command.
8. Execute Bulk Erase Setup 2 command.
3.4 Programming Program Memory

FIGURE 3-3: PROGRAM FLOW CHART - PIC16F62X PROGRAM MEMORY

Start High Voltage Programming
  - Set RB4 = VSS
  - Set MCLR = Vih
  - Set VDD = VDD

Start Low Voltage Programming
  - Set VDD = VDD
  - Set MCLR = VDD
  - Set RB4 = VDD

Program Cycle
  - Read Data from Program Memory
  - Data Correct?
    - Yes: Report
    - No: Program Failure

Increment Address Command

All Locations Done?
  - Yes: End
  - No: Verify all Locations @ VDDMIN

Report Verify Error @ VDDMIN
  - Yes: Data Correct?
    - Yes: End
    - No: Report Verify Error @ VDDMAX
  - No: Verify all Locations @ VDDMAX

Wait tprog

Begin Programming Command

Load Data Command

PROGRAM CYCLE
FIGURE 3-4: PROGRAM FLOW CHART - PIC16F62X CONFIGURATION MEMORY

Start

Load Configuration Data

Program ID Location?

Yes

Program Cycle

No

Report Programming Failure

Increment Address Command

No

Address = 0x2004?

Yes

Increment Address Command

Increment Address Command

Increment Address Command

Increment Address Command

Program Cycle (Config. Word)

Set VDD = VDDMIN

Yes

Read Data Command

No

Data Correct?

Yes

Report Program Configuration Word Error

Data Correct?

Yes

Set VDD = VDDMAX

No

Read Data Command

Data Correct?

Read Data Command

Done

Yes
3.5 Program Data Memory

FIGURE 3-5: PROGRAM FLOW CHART - PIC16F62X DATA MEMORY
3.6  Programming Range of Program Memory

FIGURE 3-6: PROGRAM FLOW CHART - PIC16F62X PROGRAM MEMORY

Start High Voltage Programming
Set RB4 = VSS
Set MCLR = V\text{inh}
Set V\text{DD} = V\text{DD}

Start Low Voltage Programming
Set V\text{DD} = V\text{DD}
Set MCLR = V\text{DD}
Set RB4 = V\text{DD}

Increment Address Command

Address = Start Address?

Program Cycle

Read Data from Program Memory

Data Correct?

Report Programming Failure

Increment Address Command

All Locations Done?

Verify all Locations @ V\text{DDMIN} & V\text{DDMAX}

Done

PROGRAM CYCLE

Load Data Command

Begin Programming Command

Wait T_{prog}
3.7 Configuration Word

The PIC16F62X has several configuration bits. These bits can be set (reads ‘0’), or left unchanged (reads ‘1’), to select various device configurations.

3.8 Device ID Word

The device ID word for the PIC16F62X is hard coded at 2006h.

<table>
<thead>
<tr>
<th>Device</th>
<th>Device ID Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16F627</td>
<td>00 0111 101</td>
</tr>
<tr>
<td>PIC16F628</td>
<td>00 0111 110</td>
</tr>
</tbody>
</table>

TABLE 3-1: DEVICE ID VALUES
**REGISTER 3-1:  CONFIGURATION WORD FOR PIC16F627/628 (ADDRESS: 2007h)**

<table>
<thead>
<tr>
<th>CP1</th>
<th>CP0</th>
<th>CP1</th>
<th>CP0</th>
<th>—</th>
<th>CPD</th>
<th>LVP</th>
<th>BOREN</th>
<th>MCLRE</th>
<th>FOSC2</th>
<th>PWRTEN</th>
<th>WDTEN</th>
<th>F0SC1</th>
<th>F0SC0</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 13-10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**bit 13-10**  
**CP1:**CP0: Code Protection bits  
11 = Program memory code protection off  
10 = 0400h-07FFh code protected  
01 = 0200h-07FFh code protected  
00 = 0000h-07FFh code protected  

**Code protection for 2K program memory**  
11 = Program memory code protection off  
10 = 0400h-07FFh code protected  
01 = 0200h-03FFh code protected  
00 = 0000h-03FFh code protected  

**Code protection for 1K program memory**  
11 = Program memory code protection off  
10 = Program memory code protection off  
01 = 0200h-03FFh code protected  
00 = 0000h-03FFh code protected  

**bit 9**  
Unimplemented: Read as ‘1’  

**bit 8**  
**CPD:** Data Code Protection bit  
1 = Data memory code protection off  
0 = Data memory code protected  

**bit 7**  
**LVP:** Low Voltage Programming Enable bit  
1 = RB4/PGM pin has PGM function, Low Voltage Programming enabled  
0 = RB4/PGM is digital input, HV on MCLR must be used for programming  

**bit 6**  
**BODEN:** Brown-out Detect Reset Enable bit  
1 = BOD Reset enabled  
0 = BOD Reset disabled  

**bit 5**  
**MCLRE:** RA5/MCLR Pin Function Select bit  
1 = RA5/MCLR pin function is MCLR  
0 = RA5/MCLR pin function is digital input, MCLR internally tied to VDD  

**bit 3**  
**PWRTEN:** Power-up Timer Enable bit  
1 = PWRT disabled  
0 = PWRT enabled  

**bit 2**  
**WDTEN:** Watchdog Timer Enable bit  
1 = WDT enabled  
0 = WDT disabled  

**bit 4, 1-0**  
**FOSC2:**FOSC0: Oscillator Selection bits  
111 = ER oscillator: CLKOUT function on RA6/OSC2/CLKOUT pin, Resistor on RA7/OSC1/CLKIN  
110 = ER oscillator: I/O function on RA6/OSC2/CLKOUT pin, Resistor on RA7/OSC1/CLKIN  
101 = INTRC oscillator: CLKOUT function on RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN  
100 = INTRC oscillator: I/O function on RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN  
011 = ExtClk: I/O function on RA6/OSC2/CLKOUT pin, CLKin on RA7/OSC1/CLKIN  
010 = HS oscillator: High speed crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN  
001 = XT oscillator: Crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN  
000 = LP oscillator: Low power crystal on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN  

**Note 1:**  
Enabling Brown-out Detect Reset automatically enables Power-up Timer (PWRT) regardless of the value of bit PWRTEN. Ensure the Power-up Timer is enabled any time Brown-out Reset is enabled.  

**Note 2:**  
All of the CP1:CP0 pairs have to be given the same value to enable the code protection scheme listed. The entire program EEPROM will be erased if the code protection is reset.  

**Note 3:**  
The entire data EEPROM will be erased when the code protection is turned off. The calibration memory is not erased.  

**Note 4:**  
When MCLR is asserted in INTRC or ER mode, the internal clock oscillator is disabled.  

---  

**Legend:**  
R = Readable bit  
W = Writable bit  
U = Unimplemented bit, read as ‘0’  
-n = Value at POR  
'1' = Bit is set  
'0' = Bit is cleared  
x = Bit is unknown  

---  

DS30034D-page 3-178  
Preliminary  
© 2003 Microchip Technology Inc.
3.9 Embedding Configuration Word and ID Information in the HEX File

To allow portability of code, the programmer is required to read the configuration word and ID locations from the HEX file when loading the HEX file. If configuration word information was not present in the HEX file, then a simple warning message may be issued. Similarly, while saving a HEX file, configuration word and ID information must be included. An option to not include this information may be provided.

Specifically for the PIC16F62X, the EEPROM data memory should also be embedded in the HEX file (see Section 4.1).

Microchip Technology Inc. feels strongly that this feature is important for the benefit of the end customer.

3.10 Checksum Computation

3.10.1 CHECKSUM

Checksum is calculated by reading the contents of the PIC16F62X memory locations and adding up the opcodes up to the maximum user addressable location (e.g., 0x7FF for the PIC16F628). Any carry bits, exceeding 16 bits, are neglected. Finally, the configuration word (appropriately masked) is added to the checksum. Checksum computation for each member of the PIC16F62X devices is shown in Table 3-2.

The checksum is calculated by summing the following:

- The contents of all program memory locations
- The configuration word, appropriately masked
- Masked ID locations (when applicable)

The Least Significant 16 bits of this sum is the checksum.

The following table describes how to calculate the checksum for each device. Note that the checksum calculation differs depending on the code protect setting. Since the program memory locations read out differently depending on the code protect setting, the table describes how to manipulate the actual program memory values to simulate the values that would be read from a protected device. When calculating a checksum, by reading a device, the entire program memory can simply be read and summed. The configuration word and ID locations can always be read.

Note: Some older devices have an additional value added in the checksum. This is to maintain compatibility with older device programmer checksums.

### TABLE 3-2: CHECKSUM COMPUTATION

<table>
<thead>
<tr>
<th>Device</th>
<th>Code Protect</th>
<th>Checksum*</th>
<th>Blank Value</th>
<th>0x25E6 at 0 and Max Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16F627</td>
<td>OFF</td>
<td>SUM[0x0000:0x3FFF] + CFGW &amp; 0x3DFF</td>
<td>0x39FF</td>
<td>0x05CD</td>
</tr>
<tr>
<td></td>
<td>0x200 : 0x3FF</td>
<td>SUM[0x0000:0x01FF] + CFGW &amp; 0x3DFF + SUM_ID</td>
<td>0x4DFE</td>
<td>0xFFB3</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3DFF + SUM_ID</td>
<td>0x3BFE</td>
<td>0x07CC</td>
</tr>
<tr>
<td>PIC16F627</td>
<td>OFF</td>
<td>SUM[0x0000:0x7FFF] + CFGW &amp; 0x3DFF</td>
<td>0x35FF</td>
<td>0x01CD</td>
</tr>
<tr>
<td></td>
<td>0x400 : 0x7FF</td>
<td>SUM[0x0000:0x03FF] + CFGW &amp; 0x3DFF + SUM_ID</td>
<td>0x5BFE</td>
<td>0x0DB3</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>SUM[0x0000:0x01FF] + CFGW &amp; 0x3DFF + SUM_ID</td>
<td>0x49FE</td>
<td>0xFBB3</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3DFF + SUM_ID</td>
<td>0x37FE</td>
<td>0x03CC</td>
</tr>
</tbody>
</table>

Legend: CFGW = Configuration Word
SUM[a:b] = [Sum of locations a to b inclusive]
SUM_ID = ID locations masked by 0xF then made into a 16-bit value with ID0 as the most significant nibble.

For example, ID0 = 0x1, ID1 = 0x2, ID3 = 0x3, ID4 = 0x4, then SUM_ID = 0x1234

*Checksum = [Sum of all the individual expressions] MODULO [0xFFFF]

+ = Addition
& = Bitwise AND
4.0 PROGRAM/VERIFY MODE ELECTRICAL CHARACTERISTICS

4.1 Embedding Data EEPROM Contents in HEX File

The programmer should be able to read data EEPROM information from a HEX file, and conversely (as an option) write data EEPROM contents to a HEX file, along with program memory information and fuse information.

The 128 data memory locations are logically mapped starting at address 0x2100. The format for data memory storage is one data byte per address location, LSB aligned.

**TABLE 4-1: AC/DC CHARACTERISTICS TIMING REQUIREMENTS FOR PROGRAM/VERIFY MODE**

<table>
<thead>
<tr>
<th>AC/DC Characteristics</th>
<th>Standard Operating Conditions (unless otherwise stated)</th>
<th>VDD level for word operations, program memory</th>
<th>VDD</th>
<th>2.0</th>
<th>5.5</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>VDD level for word operations, data memory</td>
<td>VDD</td>
<td>2.0</td>
<td>5.5</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VDD level for bulk erase/write operations, program and data memory</td>
<td>VDD</td>
<td>4.5</td>
<td>5.5</td>
<td>V</td>
</tr>
<tr>
<td>High voltage on MCLR and RA4/T0CKI for Programming mode entry</td>
<td>VDD + 3.5</td>
<td>13.5</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Sym</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
<th>Conditions/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD level for word operations, program memory</td>
<td>VDD</td>
<td>2.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD level for word operations, data memory</td>
<td>VDD</td>
<td>2.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD level for bulk erase/write operations, program and data memory</td>
<td>VDD</td>
<td>4.5</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High voltage on MCLR and RA4/T0CKI for Programming mode entry</td>
<td>VDD + 3.5</td>
<td>13.5</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLR rise time (VSS to VIHH) for Programming mode entry</td>
<td>TVHHR</td>
<td>1.0</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hold time after VPP↑</td>
<td>Tppdp</td>
<td>5</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CLOCK, DATA) input high level</td>
<td>Vih1</td>
<td>0.8 VDD</td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CLOCK, DATA) input low level</td>
<td>Vil1</td>
<td>0.2 VDD</td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOCK, DATA setup time before MCLR↑</td>
<td>Tset0</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOCK, DATA hold time after MCLR↑</td>
<td>Thld0</td>
<td>5</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Serial Program/Verify**

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Sym</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
<th>Conditions/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD level for word operations, program memory</td>
<td>VDD</td>
<td>2.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD level for word operations, data memory</td>
<td>VDD</td>
<td>2.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD level for bulk erase/write operations, program and data memory</td>
<td>VDD</td>
<td>4.5</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High voltage on MCLR and RA4/T0CKI for Programming mode entry</td>
<td>VDD + 3.5</td>
<td>13.5</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLR rise time (VSS to VIHH) for Programming mode entry</td>
<td>TVHHR</td>
<td>1.0</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hold time after VPP↑</td>
<td>Tppdp</td>
<td>5</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CLOCK, DATA) input high level</td>
<td>Vih1</td>
<td>0.8 VDD</td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CLOCK, DATA) input low level</td>
<td>Vil1</td>
<td>0.2 VDD</td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOCK, DATA setup time before MCLR↑</td>
<td>Tset0</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOCK, DATA hold time after MCLR↑</td>
<td>Thld0</td>
<td>5</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Sym</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
<th>Conditions/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD level for word operations, program memory</td>
<td>VDD</td>
<td>2.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD level for word operations, data memory</td>
<td>VDD</td>
<td>2.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD level for bulk erase/write operations, program and data memory</td>
<td>VDD</td>
<td>4.5</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High voltage on MCLR and RA4/T0CKI for Programming mode entry</td>
<td>VDD + 3.5</td>
<td>13.5</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLR rise time (VSS to VIHH) for Programming mode entry</td>
<td>TVHHR</td>
<td>1.0</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hold time after VPP↑</td>
<td>Tppdp</td>
<td>5</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CLOCK, DATA) input high level</td>
<td>Vih1</td>
<td>0.8 VDD</td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CLOCK, DATA) input low level</td>
<td>Vil1</td>
<td>0.2 VDD</td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOCK, DATA setup time before MCLR↑</td>
<td>Tset0</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOCK, DATA hold time after MCLR↑</td>
<td>Thld0</td>
<td>5</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 4-1: AC/DC CHARACTERISTICS TIMING REQUIREMENTS FOR PROGRAM/VERIFY MODE**

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Sym</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
<th>Conditions/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD level for word operations, program memory</td>
<td>VDD</td>
<td>2.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD level for word operations, data memory</td>
<td>VDD</td>
<td>2.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD level for bulk erase/write operations, program and data memory</td>
<td>VDD</td>
<td>4.5</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High voltage on MCLR and RA4/T0CKI for Programming mode entry</td>
<td>VDD + 3.5</td>
<td>13.5</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLR rise time (VSS to VIHH) for Programming mode entry</td>
<td>TVHHR</td>
<td>1.0</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hold time after VPP↑</td>
<td>Tppdp</td>
<td>5</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CLOCK, DATA) input high level</td>
<td>Vih1</td>
<td>0.8 VDD</td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CLOCK, DATA) input low level</td>
<td>Vil1</td>
<td>0.2 VDD</td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOCK, DATA setup time before MCLR↑</td>
<td>Tset0</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOCK, DATA hold time after MCLR↑</td>
<td>Thld0</td>
<td>5</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Serial Program/Verify**

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Sym</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
<th>Conditions/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD level for word operations, program memory</td>
<td>VDD</td>
<td>2.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD level for word operations, data memory</td>
<td>VDD</td>
<td>2.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD level for bulk erase/write operations, program and data memory</td>
<td>VDD</td>
<td>4.5</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High voltage on MCLR and RA4/T0CKI for Programming mode entry</td>
<td>VDD + 3.5</td>
<td>13.5</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLR rise time (VSS to VIHH) for Programming mode entry</td>
<td>TVHHR</td>
<td>1.0</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hold time after VPP↑</td>
<td>Tppdp</td>
<td>5</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CLOCK, DATA) input high level</td>
<td>Vih1</td>
<td>0.8 VDD</td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CLOCK, DATA) input low level</td>
<td>Vil1</td>
<td>0.2 VDD</td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOCK, DATA setup time before MCLR↑</td>
<td>Tset0</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOCK, DATA hold time after MCLR↑</td>
<td>Thld0</td>
<td>5</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 4-1: AC/DC CHARACTERISTICS TIMING REQUIREMENTS FOR PROGRAM/VERIFY MODE**

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Sym</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
<th>Conditions/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD level for word operations, program memory</td>
<td>VDD</td>
<td>2.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD level for word operations, data memory</td>
<td>VDD</td>
<td>2.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD level for bulk erase/write operations, program and data memory</td>
<td>VDD</td>
<td>4.5</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High voltage on MCLR and RA4/T0CKI for Programming mode entry</td>
<td>VDD + 3.5</td>
<td>13.5</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLR rise time (VSS to VIHH) for Programming mode entry</td>
<td>TVHHR</td>
<td>1.0</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hold time after VPP↑</td>
<td>Tppdp</td>
<td>5</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CLOCK, DATA) input high level</td>
<td>Vih1</td>
<td>0.8 VDD</td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CLOCK, DATA) input low level</td>
<td>Vil1</td>
<td>0.2 VDD</td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOCK, DATA setup time before MCLR↑</td>
<td>Tset0</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOCK, DATA hold time after MCLR↑</td>
<td>Thld0</td>
<td>5</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Serial Program/Verify**

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Sym</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
<th>Conditions/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD level for word operations, program memory</td>
<td>VDD</td>
<td>2.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD level for word operations, data memory</td>
<td>VDD</td>
<td>2.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDD level for bulk erase/write operations, program and data memory</td>
<td>VDD</td>
<td>4.5</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High voltage on MCLR and RA4/T0CKI for Programming mode entry</td>
<td>VDD + 3.5</td>
<td>13.5</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLR rise time (VSS to VIHH) for Programming mode entry</td>
<td>TVHHR</td>
<td>1.0</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hold time after VPP↑</td>
<td>Tppdp</td>
<td>5</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CLOCK, DATA) input high level</td>
<td>Vih1</td>
<td>0.8 VDD</td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CLOCK, DATA) input low level</td>
<td>Vil1</td>
<td>0.2 VDD</td>
<td>V</td>
<td>Schmitt Trigger input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOCK, DATA setup time before MCLR↑</td>
<td>Tset0</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOCK, DATA hold time after MCLR↑</td>
<td>Thld0</td>
<td>5</td>
<td>µs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This document includes the programming specifications for the following devices:

- PIC16F870
- PIC16F871
- PIC16F872
- PIC16F873
- PIC16F874
- PIC16F876
- PIC16F877

1.0 PROGRAMMING THE PIC16F87X

The PIC16F87X is programmed using a serial method. The Serial mode will allow the PIC16F87X to be programmed while in the user’s system. This allows for increased design flexibility. This programming specification applies to PIC16F87X devices in all packages.

1.1 Programming Algorithm Requirements

The programming algorithm used depends on the operating voltage (VDD) of the PIC16F87X device. Algorithm 1 is designed for a VDD range of 2.2V ≤ VDD < 5.5V. Algorithm 2 is for a range of 4.5V ≤ VDD ≤ 5.5V. Either algorithm can be used with the two available programming entry methods. The first method follows the normal Microchip Programming mode entry of applying a VPP voltage of 13V ± .5V. The second method, called Low Voltage ICSP™ or LVP for short, applies VDD to MCLR and uses the I/O pin RB3 to enter Programming mode. When RB3 is driven to VDD from ground, the PIC16F87X device enters Programming mode.

1.2 Programming Mode

The Programming mode for the PIC16F87X allows programming of user program memory, data memory, special locations used for ID, and the configuration word.

---

**Pin Diagram**

PDIP, SOIC

MCLR/VPP  RA0/AN0  RA1/AN1  RA2/AN2/VREF  RA3/AN3/VREF  RA4/T0CKI  RA5/AN4/SS  RE0/T0AN5  RE1/T1AN6  RE2/CCS/AN7  VDD  VSS  OSC1/CLKIN  OSC2/CLKOUT  RC0/T1OSO/T1CKI  RC1/T1OSI/CCP2  RC2/CCP1  RC3/SCK/SDI  RC4/SDO  RC5/SDI/SDA

MCLR/VPP  RA0/AN0  RA1/AN1  RA2/AN2/VREF  RA3/AN3/VREF  RA4/T0CKI  RA5/AN4/SS  RE0/T0AN5  RE1/T1AN6  RE2/CCS/AN7  VDD  VSS  OSC1/CLKIN  OSC2/CLKOUT  RC0/T1OSO/T1CKI  RC1/T1OSI/CCP2  RC2/CCP1  RC3/SCK/SDI  RC4/SDO  RC5/SDI/SDA

1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40

2.2V ≤ VDD < 5.5V

4.5V ≤ VDD ≤ 5.5V
### PIN DESCRIPTIONS (DURING PROGRAMMING): PIC16F87X

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>During Programming</th>
<th>Pin Type</th>
<th>Pin Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RB3</td>
<td>PGM</td>
<td>I</td>
<td>Low voltage ICSP programming input if LVP configuration bit equals 1</td>
</tr>
<tr>
<td>RB6</td>
<td>CLOCK</td>
<td>I</td>
<td>Clock input</td>
</tr>
<tr>
<td>RB7</td>
<td>DATA</td>
<td>I/O</td>
<td>Data input/output</td>
</tr>
<tr>
<td>MCLR</td>
<td>VTEST MODE</td>
<td>P*</td>
<td>Program Mode Select</td>
</tr>
<tr>
<td>VDD</td>
<td>VDD</td>
<td>P</td>
<td>Power Supply</td>
</tr>
<tr>
<td>VSS</td>
<td>VSS</td>
<td>P</td>
<td>Ground</td>
</tr>
</tbody>
</table>

Legend: I = Input, O = Output, P = Power

* In the PIC16F87X, the programming high voltage is internally generated. To activate the Programming mode, high voltage needs to be applied to the MCLR input. Since the MCLR is used for a level source, this means that MCLR does not draw any significant current.
2.0 PROGRAM MODE ENTRY

2.1 User Program Memory Map

The user memory space extends from 0x0000 to 0x1FFF (8K). In Programming mode, the program memory space extends from 0x0000 to 0x3FFF, with the first half (0x0000-0x1FFF) being user program memory and the second half (0x2000-0x3FFF) being configuration memory. The PC will increment from 0x0000 to 0x1FFF and wrap to 0x0000, 0x2000 to 0x3FFF and wrap around to 0x2000 (not to 0x0000). Once in configuration memory, the highest bit of the PC stays a ‘1’, thus always pointing to the configuration memory. The only way to point to user program memory is to reset the part and re-enter Program/Verify mode, as described in Section 2.4.

In the configuration memory space, 0x2000-0x200F are physically implemented. However, only locations 0x2000 through 0x2007 are available. Other locations are reserved. Locations beyond 0x200F will physically access user memory (see Figure 2-1).

2.2 Data EEPROM Memory

The EEPROM data memory space is a separate block of high endurance memory that the user accesses using a special sequence of instructions. The amount of data EEPROM memory depends on the device and is shown below in number of bytes.

The contents of data EEPROM memory have the capability to be embedded into the HEX file.

The programmer should be able to read data EEPROM information from a HEX file and conversely (as an option), write data EEPROM contents to a HEX file, along with program memory information and configuration bit information.

The 256 data memory locations are logically mapped starting at address 0x2100. The format for data memory storage is one data byte per address location, LSB aligned.

2.3 ID Locations

A user may store identification information (ID) in four ID locations. The ID locations are mapped in [0x2000 : 0x2003]. It is recommended that the user use only the four Least Significant bits of each ID location. In some devices, the ID locations read out in an unscrambled fashion after code protection is enabled. For these devices, it is recommended that ID location is written as “11 1111 1000 bbbb” where ‘bbbb’ is ID information.

In other devices, the ID locations read out normally, even after code protection. To understand how the devices behave, refer to Table 5-1.

To understand the scrambling mechanism after code protection, refer to Section 4.0.

<table>
<thead>
<tr>
<th>Device</th>
<th># of Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16F870</td>
<td>64</td>
</tr>
<tr>
<td>PIC16F871</td>
<td>64</td>
</tr>
<tr>
<td>PIC16F872</td>
<td>64</td>
</tr>
<tr>
<td>PIC16F873</td>
<td>128</td>
</tr>
<tr>
<td>PIC16F874</td>
<td>128</td>
</tr>
<tr>
<td>PIC16F876</td>
<td>256</td>
</tr>
<tr>
<td>PIC16F877</td>
<td>256</td>
</tr>
</tbody>
</table>

The programmer should be able to read data EEPROM information from a HEX file and conversely (as an option), write data EEPROM contents to a HEX file, along with program memory information and configuration bit information.
### TABLE 2-1: PROGRAM MEMORY MAPPING

<table>
<thead>
<tr>
<th>Location</th>
<th>2K words</th>
<th>4K words</th>
<th>8K words</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>2001h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>2002h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>2003h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>2004h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>2005h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>2006h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>2007h</td>
<td>Implemented</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td>2008h</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>2100h</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>3FFFh</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

#### Diagram:

- ID Location
- Configuration Word
- Device ID
- Reserved
- Device ID
2.4 Program/Verify Mode

The Program/Verify mode is entered by holding pins RB6 and RB7 low, while raising MCLR pin from VIL to VIHH (high voltage). In this mode, the state of the RB3 pin does not affect programming. Low voltage ICSP Programming mode is entered by raising RB3 from VIL to VDD and then applying VDD to MCLR. Once in this mode, the user program memory and the configuration memory can be accessed and programmed in serial fashion. The mode of operation is serial, and the memory that is accessed is the user program memory. RB6 and RB7 are Schmitt Trigger Inputs in this mode.

The sequence that enters the device into the Programming/Verify mode places all other logic into the RESET state (the MCLR pin was initially at VIL). This means that all I/O are in the RESET state (high impedance inputs).

The normal sequence for programming is to use the load data command to set a value to be written at the selected address. Issue the begin programming command followed by read data command to verify, and then increment the address.

A device RESET will clear the PC and set the address to 0. The “increment address” command will increment the PC. The “load configuration” command will set the PC to 0x2000. The available commands are shown in Table 2-2.

2.4.1 LOW VOLTAGE ICSP PROGRAMMING MODE

Low voltage ICSP Programming mode allows a PIC16F87X device to be programmed using VDD only. However, when this mode is enabled by a configuration bit (LVP), the PIC16F87X device dedicates RB3 to control entry/exit into Programming mode.

When LVP bit is set to ‘1’, the low voltage ICSP programming entry is enabled. Since the LVP configuration bit allows low voltage ICSP programming entry in its erased state, an erased device will have the LVP bit enabled at the factory. While LVP is ‘1’, RB3 is dedicated to low voltage ICSP programming. Bring RB3 to VDD and then MCLR to VDD to enter programming mode. All other specifications for high voltage ICSP™ apply.

To disable low voltage ICSP mode, the LVP bit must be programmed to ‘0’. This must be done while entered with High Voltage Entry mode (LVP bit = 1). RB3 is now a general purpose I/O pin.

2.4.2 SERIAL PROGRAM/VERIFY OPERATION

The RB6 pin is used as a clock input pin, and the RB7 pin is used for entering command bits and data input/output during serial operation. To input a command, the clock pin (RB6) is cycled six times. Each command bit is latched on the falling edge of the clock, with the Least Significant bit (LSb) of the command being input first. The data on pin RB7 is required to have a minimum setup and hold time (see AC/DC specifications), with respect to the falling edge of the clock. Commands that have data associated with them (read and load) are specified to have a minimum delay of 1 \( \mu \)s between the command and the data. After this delay, the clock pin is cycled 16 times with the first cycle being a START bit and the last cycle being a STOP bit. Data is also input and output LSb first.

Therefore, during a read operation, the LSb will be transmitted onto pin RB7 on the rising edge of the second cycle, and during a load operation, the LSb will be latched on the falling edge of the second cycle. A minimum 1 \( \mu \)s delay is also specified between consecutive commands.

All commands are transmitted LSb first. Data words are also transmitted LSb first. The data is transmitted on the rising edge and latched on the falling edge of the clock. To allow for decoding of commands and reversal of data pin configuration, a time separation of at least 1 \( \mu \)s is required between a command and a data word (or another command).

The commands that are available are:

2.4.2.1 Load Configuration

After receiving this command, the program counter (PC) will be set to 0x2000. By then applying 16 cycles to the clock pin, the chip will load 14-bits in a “data word,” as described above, to be programmed into the configuration memory. A description of the memory mapping schemes of the program memory for normal operation and Configuration mode operation is shown in Figure 2-1. After the configuration memory is entered, the only way to get back to the user program memory is to exit the Program/Verify Test mode by taking MCLR low (VIL).

2.4.2.2 Load Data for Program Memory

After receiving this command, the chip will load in a 14-bit “data word” when 16 cycles are applied, as described previously. A timing diagram for the load data command is shown in Figure 6-1.

**Note:** The OSC must not have 72 osc clocks while the device MCLR is between VIL and VIHH.
2.4.2.3 Load Data for Data Memory
After receiving this command, the chip will load in a 14-bit “data word” when 16 cycles are applied. However, the data memory is only 8-bits wide, and thus, only the first 8-bits of data after the START bit will be programmed into the data memory. It is still necessary to cycle the clock the full 16 cycles in order to allow the internal circuitry to reset properly. The data memory contains up to 256 bytes. If the device is code protected, the data is read as all zeros.

2.4.2.4 Read Data from Program Memory
After receiving this command, the chip will transmit data bits out of the program memory (user or configuration) currently accessed, starting with the second rising edge of the clock input. The RB7 pin will go into Output mode on the second rising clock edge, and it will revert back to Input mode (hi-impedance) after the 16th rising edge. A timing diagram of this command is shown in Figure 6-2.

2.4.2.5 Read Data from Data Memory
After receiving this command, the chip will transmit data bits out of the data memory starting with the second rising edge of the clock input. The RB7 pin will go into Output mode on the second rising edge, and it will revert back to Input mode (hi-impedance) after the 16th rising edge. As previously stated, the data memory is 8-bits wide, and therefore, only the first 8-bits that are output are actual data.

2.4.2.6 Increment Address
The PC is incremented when this command is received. A timing diagram of this command is shown in Figure 6-3.

2.4.2.7 Begin Erase/Program Cycle
A load command must be given before every begin programming command. Programming of the appropriate memory (test program memory, user program memory or data memory) will begin after this command is received and decoded. An internal timing mechanism executes an erase before write. The user must allow for both erase and programming cycle times for programming to complete. No “end programming” command is required.

2.4.2.8 Begin Programming
This command is similar to the ERASE/PROGRAM CYCLE command, except that a word erase is not done. It is recommended that a bulk erase be performed before starting a series of programming only cycles.

### TABLE 2-2: COMMAND MAPPING FOR PIC16F87X

<table>
<thead>
<tr>
<th>Command</th>
<th>Mapping (MSB … LSB)</th>
<th>Data</th>
<th>Voltage Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Configuration</td>
<td>X X 0 0 0 0 0</td>
<td>0, data (14), 0</td>
<td>2.2V - 5.5V</td>
</tr>
<tr>
<td>Load Data for Program Memory</td>
<td>X X 0 0 1 0</td>
<td>0, data (14), 0</td>
<td>2.2V - 5.5V</td>
</tr>
<tr>
<td>Read Data from Program Memory</td>
<td>X X 0 1 0 0</td>
<td>0, data (14), 0</td>
<td>2.2V - 5.5V</td>
</tr>
<tr>
<td>Increment Address</td>
<td>X X 1 1 0 0</td>
<td></td>
<td>2.2V - 5.5V</td>
</tr>
<tr>
<td>Begin Erase Programming Cycle</td>
<td>0 0 1 0 0 0</td>
<td></td>
<td>2.2V - 5.5V</td>
</tr>
<tr>
<td>Begin Programming Only Cycle</td>
<td>0 1 1 0 0 0</td>
<td>0, data (14), 0</td>
<td>4.5V - 5.5V</td>
</tr>
<tr>
<td>Load Data for Data Memory</td>
<td>X X 0 0 1 1</td>
<td>0, data (14), 0</td>
<td>2.2V - 5.5V</td>
</tr>
<tr>
<td>Read Data from Data Memory</td>
<td>X X 0 1 0 1</td>
<td>0, data (14), 0</td>
<td>2.2V - 5.5V</td>
</tr>
<tr>
<td>Bulk Erase Setup1</td>
<td>0 0 0 0 0 1</td>
<td></td>
<td>4.5V - 5.5V</td>
</tr>
<tr>
<td>Bulk Erase Setup2</td>
<td>0 0 0 1 1 1</td>
<td></td>
<td>4.5V - 5.5V</td>
</tr>
</tbody>
</table>

Note: The Begin Program operation must take place at 4.5 to 5.5 VDD range.

A load command must be given before every begin programming command. Programming of the appropriate memory (test program memory, user program memory or data memory) will begin after this command is received and decoded. An internal timing mechanism executes a write. The user must allow for program cycle time for programming to complete. No “end programming” command is required.
2.5 Erasing Program and Data Memory

Depending on the state of the code protection bits, program and data memory will be erased using different procedures. The first set of procedures is used when both program and data memories are not code protected. The second set of procedures must be used when either memory is code protected. A device programmer should determine the state of the code protection bits and then apply the proper procedure to erase the desired memory.

2.5.1 ERASING NON-CODE PROTECTED PROGRAM AND DATA MEMORY

When both program and data memories are not code protected, they must be individually erased using the following procedures. The only way that both memories are erased using a single procedure is if code protection is enabled for one of the memories. These procedures do not erase the configuration word or ID locations.

Procedure to bulk erase program memory:

1. Execute a Load Data for Program Memory command (000010) with a '1' in all locations (0x3FFF)
2. Execute a Bulk Erase Setup1 command (000001)
3. Execute a Bulk Erase Setup2 command (000111)
4. Execute a Begin Erase/Programming command (001000)
5. Wait 8 ms
6. Execute a Bulk Erase Setup1 command (000001)
7. Execute a Bulk Erase Setup2 command (000111)

Procedure to bulk erase data memory:

1. Execute a Load Data for Data Memory command (000011) with a '1' in all locations (0x3FFF)
2. Execute a Bulk Erase Setup1 command (000001)
3. Execute a Bulk Erase Setup2 command (000111)
4. Execute a Begin Erase/Programming command (001000)
5. Wait 8 ms
6. Execute a Bulk Erase Setup1 command (000001)
7. Execute a Bulk Erase Setup2 command (000111)

2.5.2 ERASING CODE PROTECTED MEMORY

For the PIC16F87X devices, once code protection is enabled, all protected program and data memory locations read all '0's and further programming is disabled. The ID locations and configuration word read out unscrambled and can be reprogrammed normally. The only procedure to erase a PIC16F87X device that is code protected is shown in the following procedure. This method erases program memory, data memory, configuration bits and ID locations. Since all data within the program and data memory will be erased when this procedure is executed, the security of the data or code is not compromised.

1. Execute a Load Configuration command (000000) with a '1' in all locations (0x3FFF)
2. Execute Increment Address command (000110) to set address to configuration word location (0x2007)
3. Execute a Bulk Erase Setup1 command (000001)
4. Execute a Bulk Erase Setup2 command (000111)
5. Execute a Begin Erase/Programming command (001000)
6. Wait 8 ms
7. Execute a Bulk Erase Setup1 command (000001)
8. Execute a Bulk Erase Setup2 command (000111)
FIGURE 2-1: FLOW CHART - PIC16F87X PROGRAM MEMORY (2.2V ≤ VDD < 5.5V)

START

Set VDD = VDDP

Load Data Command

Begin Erase/Programming Command

Wait tEra + tprog

Increment Address Command

All Locations Done?

Verify all Locations

Report Verify Error

Data Correct?

DONE
FIGURE 2-2: FLOW CHART – PIC16F87X PROGRAM MEMORY (4.5V \leq V_{DD} \leq 5.5V)

START

Bulk Erase Sequence

Set V_{DD} = V_{DDP}

Load Data Command

Begin Programming Only Command

Wait t_{prog}

Increment Address Command

All Locations Done?

Verify all Locations

Report Verify Error

Data Correct?

DONE
FIGURE 2-3: FLOW CHART – PIC16F87X CONFIGURATION MEMORY (2.2V ≤ VDD < 5.5V)
FIGURE 2-4: FLOW CHART - PIC16F87X CONFIGURATION MEMORY

* Assumes that a bulk erase was issued before programming configuration word. If not, use the program flow from Figure 2-4.
3.0 CONFIGURATION WORD

The PIC16F87X has several configuration bits. These bits can be set (reads ‘0’), or left unchanged (reads ‘1’), to select various device configurations.

3.1 Device ID Word

The device ID word for the PIC16F87X is located at 2006h.

<table>
<thead>
<tr>
<th>Device</th>
<th>Device ID Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16F870</td>
<td>00 1101 000</td>
</tr>
<tr>
<td>PIC16F871</td>
<td>00 1101 001</td>
</tr>
<tr>
<td>PIC16F872</td>
<td>00 1000 111</td>
</tr>
<tr>
<td>PIC16F873</td>
<td>00 1001 011</td>
</tr>
<tr>
<td>PIC16F874</td>
<td>00 1001 001</td>
</tr>
<tr>
<td>PIC16F876</td>
<td>00 1001 111</td>
</tr>
<tr>
<td>PIC16F877</td>
<td>00 1001 101</td>
</tr>
</tbody>
</table>
**REGISTER 3-1: CONFIG: CONFIGURATION WORD FOR PIC16F873/874/876/877**

<table>
<thead>
<tr>
<th>Address 2007h</th>
</tr>
</thead>
<tbody>
<tr>
<td>U-0</td>
</tr>
<tr>
<td>CP1</td>
</tr>
</tbody>
</table>

**bit 13-12**
- **CP1:CP0**: FLASH Program Memory Code Protection bits \(^{(2)}\)
  - **4K Devices**:
    - 11 = Code protection off
    - 10 = 0F00h to 0FFFh code protected
    - 01 = 0800h to 0FFFh code protected
    - 00 = 0000h to 0FFFh code protected
  - **8K Devices**:
    - 11 = Code protection off
    - 10 = 1F00h to 1FFFh code protected
    - 01 = 1000h to 1FFFh code protected
    - 00 = 0000h to 1FFFh code protected

**bit 11**
- **Reserved**: Set to ‘1’ for normal operation

**bit 10**
- **Unimplemented**: Read as ‘1’

**bit 9**
- **WRT**: FLASH Program Memory Write Enable bit
  - 1 = Unprotected program memory may be written to by EECON control
  - 0 = Unprotected program memory may not be written to by EECON control

**bit 8**
- **CPD**: Data EE Memory Code Protection bit
  - 1 = Code protection off
  - 0 = Data EE memory code protected

**bit 7**
- **LVP**: Low Voltage ICSP Programming Enable bit
  - 1 = RB3/PGM pin has PGM function, low voltage programming enabled
  - 0 = RB3 is digital I/O, HV on MCLR must be used for programming

**bit 6**
- **BODEN**: Brown-out Reset Enable bit \(^{(2)}\)
  - 1 = BOR enabled
  - 0 = BOR disabled

**bit 3**
- **PWRTE**: Power-up Timer Enable bit
  - 1 = PWRT disabled
  - 0 = PWRT enabled

**bit 2**
- **WDTE**: Watchdog Timer Enable bit
  - 1 = WDT enabled
  - 0 = WDT disabled

**bit 1-0**
- **F0SC1:F0SC0**: Oscillator Selection bits
  - 11 = RC oscillator
  - 10 = HS oscillator
  - 01 = XT oscillator
  - 00 = LP oscillator

**Note 1**: Enabling Brown-out Reset automatically enables Power-up Timer (PWRT), regardless of the value of bit PWRTE. Ensure the Power-up Timer is enabled any time Brown-out Reset is enabled.

**Note 2**: All of the CP1:CP0 pairs have to be given the same value to enable the code protection scheme listed.

**Legend**:
- R = Readable bit
- P = Programmable bit
- U = Unimplemented bit, read as ‘0’
- -n = Value when device is unprogrammed
- u = Unchanged from programmed state
### REGISTER 3-2: CONFIG: CONFIGURATION WORD FOR PIC16F870/871/872 (ADDRESS 2007h)

<table>
<thead>
<tr>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5-4</th>
<th>Bit 3-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP1</td>
<td>CP0</td>
<td>RESV</td>
<td>WRT</td>
<td>CPD</td>
<td>LVP</td>
<td>BODEN</td>
<td>CP1</td>
<td>CP0</td>
<td>PWRTE</td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 13-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CP1:CP0:</td>
<td>FLASH Program Memory Code Protection bits&lt;sup&gt;(2)&lt;/sup&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 5-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CP1:CP0:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FLASH Program Memory Code Protection bits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11 = Code protection off</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 = Not supported</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>01 = Not supported</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>00 = 0000h to 07FFh code protected</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 11</td>
<td>Reserved:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set to ‘1’ for normal operation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 10</td>
<td>Unimplemented:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read as ‘1’</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 9</td>
<td>WRT:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FLASH Program Memory Write Enable bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Unprotected program memory may be written to by EECON control</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = Unprotected program memory may not be written to by EECON control</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 8</td>
<td>CPD:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data EE Memory Code Protection bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Code protection off</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = Data EE memory code protected</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 7</td>
<td>LVP:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Low Voltage ICSP Programming Enable bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = RB3/PGM pin has PGM function, low voltage programming enabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = RB3 is digital I/O, HV on MCLR must be used for programming</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 6</td>
<td>BODEN:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Brown-out Reset Enable bit&lt;sup&gt;(2)&lt;/sup&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = BOR enabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = BOR disabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 3</td>
<td>PWRTE:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Power-up Timer Enable bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = PWRT disabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = PWRT enabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 2</td>
<td>WDTE:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Watchdog Timer Enable bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = WDT enabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 = WDT disabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 1-0</td>
<td>FOSC1:FOSC0:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Oscillator Selection bits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11 = RC oscillator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 = HS oscillator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>01 = XT oscillator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>00 = LP oscillator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note 1:** Enabling Brown-out Reset automatically enables Power-up Timer (PWRT), regardless of the value of bit PWRTE. Ensure the Power-up Timer is enabled any time Brown-out Reset is enabled.

**Note 2:** All of the CP1:CP0 pairs have to be given the same value to enable the code protection scheme listed.

---

**Legend:**

- **R** = Readable bit
- **P** = Programmable bit
- **U** = Unimplemented bit, read as ‘0’
- **u** = Unchanged from programmed state

---
4.0 EMBEDDING THE CONFIGURATION WORD AND ID INFORMATION IN THE HEX FILE

To allow portability of code, the programmer is required to read the configuration word and ID locations from the HEX file when loading the HEX file. If configuration word information was not present in the HEX file, then a simple warning message may be issued. Similarly, while saving a HEX file, configuration word and ID information must be included. An option to not include this information may be provided.

Specifically for the PIC16F87X, the EEPROM data memory should also be embedded in the HEX file (see Section 2.2).

Microchip Technology Inc. feels strongly that this feature is important for the benefit of the end customer.
5.0 CHECKSUM COMPUTATION

Checksum is calculated by reading the contents of the
PIC16F87X memory locations and adding up the
opcodes, up to the maximum user addressable loca-
tion, e.g., 0x1FF for the PIC16F87X. Any carry bits
exceeding 16-bits are neglected. Finally, the configura-
tion word (appropriately masked) is added to the
checksum. Checksum computation for each member of
the PIC16F87X devices is shown in Table 5-1.

The checksum is calculated by summing the following:
- The contents of all program memory locations
- The configuration word, appropriately masked
- Masked ID locations (when applicable)

The Least Significant 16 bits of this sum are the
checksum.

The following table describes how to calculate the
checksum for each device. Note that the checksum cal-
culation differs depending on the code protect setting.
Since the program memory locations read out differ-
ently depending on the code protect setting, the table
describes how to manipulate the actual program mem-
ory values to simulate the values that would be read
from a protected device. When calculating a checksum
by reading a device, the entire program memory can
simply be read and summed. The configuration word
and ID locations can always be read.

Note that some older devices have an additional value
added in the checksum. This is to maintain compatibil-
ity with older device programmer checksums.
### Table 5-1: Checksum Computation

<table>
<thead>
<tr>
<th>Device</th>
<th>Code Protect</th>
<th>Checksum*</th>
<th>Blank Value</th>
<th>0x25E6 at 0 and max address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16F870</td>
<td>OFF</td>
<td>SUM[0x0000:0x07FF] + CFGW &amp; 0x3BFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3BFF + SUM_ID</td>
<td>0x33FF</td>
<td>0xFFCD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x3FCE</td>
<td>0x0B9C</td>
</tr>
<tr>
<td>PIC16F871</td>
<td>OFF</td>
<td>SUM[0x0000:0x07FF] + CFGW &amp; 0x3BFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3BFF + SUM_ID</td>
<td>0x33FF</td>
<td>0xFFCD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x3FCE</td>
<td>0x0B9C</td>
</tr>
<tr>
<td>PIC16F872</td>
<td>OFF</td>
<td>SUM[0x0000:0x07FF] + CFGW &amp; 0x3BFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3BFF + SUM_ID</td>
<td>0x33FF</td>
<td>0xFFCD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x3FCE</td>
<td>0x0B9C</td>
</tr>
<tr>
<td>PIC16F873</td>
<td>OFF</td>
<td>SUM[0x0000:0x07FF] + CFGW &amp; 0x3BFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0F00 : 0xFF</td>
<td>SUM[0x0000:0x0EFF] + SUM_ID</td>
<td>0x2BFF</td>
<td>0xF7CD</td>
</tr>
<tr>
<td></td>
<td>0x0800 : 0xFF</td>
<td>SUM[0x0000:0x07FF] + CFGW &amp; 0x3BFF + SUM_ID</td>
<td>0x48EE</td>
<td>0xFAA3</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3BFF + SUM_ID</td>
<td>0x3FDE</td>
<td>0xF193</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x37CE</td>
<td>0x039C</td>
</tr>
<tr>
<td>PIC16F874</td>
<td>OFF</td>
<td>SUM[0x0000:0x07FF] + CFGW &amp; 0x3BFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x0F00 : 0xFF</td>
<td>SUM[0x0000:0x0EFF] + SUM_ID</td>
<td>0x2BFF</td>
<td>0xF7CD</td>
</tr>
<tr>
<td></td>
<td>0x0800 : 0xFF</td>
<td>SUM[0x0000:0x07FF] + CFGW &amp; 0x3BFF + SUM_ID</td>
<td>0x48EE</td>
<td>0xFAA3</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3BFF + SUM_ID</td>
<td>0x3FDE</td>
<td>0xF193</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x37CE</td>
<td>0x039C</td>
</tr>
<tr>
<td>PIC16F876</td>
<td>OFF</td>
<td>SUM[0x0000:0x17FF] + CFGW &amp; 0x3BFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x1F00 : 0x1FF</td>
<td>SUM[0x0000:0x1EFF] + SUM_ID</td>
<td>0x1BFF</td>
<td>0xE7CD</td>
</tr>
<tr>
<td></td>
<td>0x1000 : 0x1FF</td>
<td>SUM[0x0000:0x0EFF] + SUM_ID</td>
<td>0x28EE</td>
<td>0xDAA3</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3BFF + SUM_ID</td>
<td>0x27DE</td>
<td>0xD993</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x27CE</td>
<td>0xF39C</td>
</tr>
<tr>
<td>PIC16F877</td>
<td>OFF</td>
<td>SUM[0x0000:0x17FF] + CFGW &amp; 0x3BFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x1F00 : 0x1FF</td>
<td>SUM[0x0000:0x1EFF] + SUM_ID</td>
<td>0x1BFF</td>
<td>0xE7CD</td>
</tr>
<tr>
<td></td>
<td>0x1000 : 0x1FF</td>
<td>SUM[0x0000:0x0EFF] + SUM_ID</td>
<td>0x28EE</td>
<td>0xDAA3</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>CFGW &amp; 0x3BFF + SUM_ID</td>
<td>0x27DE</td>
<td>0xD993</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x27CE</td>
<td>0xF39C</td>
</tr>
</tbody>
</table>

Legend:  
- CFGW = Configuration Word  
- SUM[a:b] = [Sum of locations a to b inclusive]  
- SUM_ID = ID locations masked by 0xF then made into a 16-bit value with ID0 as the most significant nibble.  
  
  For example, ID0 = 0x1, ID1 = 0x2, ID3 = 0x3, ID4 = 0x4, then SUM_ID = 0x1234  

*Checksum = [Sum of all the individual expressions] MODULO [0xFFFF]  
- + = Addition  
- & = Bitwise AND
### 6.0 PROGRAM/VERIFY MODE ELECTRICAL CHARACTERISTICS

#### TABLE 6-1: TIMING REQUIREMENTS FOR PROGRAM/VERIFY MODE

<table>
<thead>
<tr>
<th>AC/DC CHARACTERISTICS</th>
<th>Standard Operating Conditions (unless otherwise stated)</th>
<th>Characteristics</th>
<th>Sym</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
<th>Conditions/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Operating Temperature: (0°C \leq T_a &lt; +70°C)</td>
<td>General</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Operating Voltage: (2.2V \leq V_{DD} \leq 5.5V)</td>
<td>V_{DD} level for Algorithm 1</td>
<td>V_{DD}</td>
<td>2.2</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td>Limited command set (See Table 2-2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V_{DD} level for Algorithm 2</td>
<td>V_{DD}</td>
<td>4.5</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td>All commands available</td>
</tr>
<tr>
<td></td>
<td></td>
<td>High voltage on MCLR for high voltage programming entry</td>
<td>V_{IH}</td>
<td>V_{DD} + 3.5</td>
<td>13.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Voltage on MCLR for low voltage ICSP programming entry</td>
<td>V_{IH}</td>
<td>2.2</td>
<td>5.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MCLR rise time (V_{SS} to V_{IH}) for Test mode entry</td>
<td>t_{VHHR}</td>
<td>1.0</td>
<td></td>
<td>μs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(RB6, RB7) input high level</td>
<td>V_{IH1}</td>
<td>0.8 V_{DD}</td>
<td></td>
<td>V</td>
<td></td>
<td>Schmitt Trigger input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(RB6, RB7) input low level</td>
<td>V_{IL1}</td>
<td>0.2 V_{DD}</td>
<td></td>
<td>V</td>
<td></td>
<td>Schmitt Trigger input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RB&lt;7:6&gt; setup time before MCLR ↑</td>
<td>t_{set0}</td>
<td>100</td>
<td></td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RB&lt;7:6&gt; hold time after MCLR ↑</td>
<td>t_{thld0}</td>
<td>5</td>
<td></td>
<td>μs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RB3 setup time before MCLR ↑</td>
<td>t_{set2}</td>
<td>100</td>
<td></td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Serial Program/Verify</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data in setup time before clock ↓</td>
<td>t_{set1}</td>
<td>100</td>
<td></td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data in hold time after clock ↓</td>
<td>t_{thld1}</td>
<td>100</td>
<td></td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data input not driven to next clock input (delay required between command/data or command/command)</td>
<td>t_{dly1}</td>
<td>1.0</td>
<td></td>
<td>μs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delay between clock ↓ to clock ↑ of next command or data</td>
<td>t_{dly2}</td>
<td>1.0</td>
<td></td>
<td>μs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clock ↑ to data out valid (during read data)</td>
<td>t_{dly3}</td>
<td>80</td>
<td></td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Erase cycle time</td>
<td>t_{era}</td>
<td>2</td>
<td></td>
<td>4 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Programming cycle time</td>
<td>t_{prog}</td>
<td>2</td>
<td></td>
<td>4 ms</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FIGURE 6-1: LOAD DATA COMMAND MCLR = V\text{IH} (PROGRAM/VERIFY)

FIGURE 6-2: READ DATA COMMAND MCLR = V\text{IH} (PROGRAM/VERIFY)

FIGURE 6-3: INCREMENT ADDRESS COMMAND MCLR = V\text{IH} (PROGRAM/VERIFY)
FIGURE 6-4: LOAD DATA COMMAND MCLR = VDD (PROGRAM/VERIFY)

FIGURE 6-5: READ DATA COMMAND MCLR = VDD (PROGRAM/VERIFY)

FIGURE 6-6: INCREMENT ADDRESS COMMAND MCLR = VDD (PROGRAM/VERIFY)
Section 4 – Application Notes

IN-CIRCUIT SERIAL PROGRAMMING™ (ICSP™) OF CALIBRATION PARAMETERS
USING A PICmicro® MICROCONTROLLER ................................................................. 4-1
INTRODUCTION

Many embedded control applications, where sensor offsets, slopes and configuration information are measured and stored, require a calibration step. Traditionally, potentiometers or Serial EEPROM devices are used to set up and store this calibration information. This application note will show how to construct a programming jig that will receive calibration parameters from the application mid-range PICmicro® microcontrollers (MCU) and program this information into the application baseline PICmicro MCU using the In-Circuit Serial Programming (ICSP) protocol. This method uses the PIC16CXXX In-Circuit Serial Programming algorithm of the 14-bit core microcontrollers.

PROGRAMMING FIXTURE

A programming fixture is needed to assist with the self programming operation. This is typically a small reusable module that plugs into the application PCB being calibrated. Only five pin connections are needed and this programming fixture can draw its power from the application PCB to simplify the connections.

FIGURE 1:
**Electrical Interface**

There are a total of five electrical connections needed between the application PIC16CXXX microcontroller and the programming jig:

- **MCLR/VPP** - High voltage pin used to place application PIC16CXXX into programming mode
- **VDD** - +5 volt power supply connection to the application PIC16CXXX
- **VSS** - Ground power supply connection to the application PIC16CXXX
- **RB6** - PORTB, bit6 connection to application PIC16CXXX used to clock programming data
- **RB7** - PORTB, bit7 connection to application PIC16CXXX used to send programming data

This programming jig is intended to grab power from the application power supply through the VDD connection. The programming jig will require 100 mA of peak current during programming. The application will need to set RB6 and RB7 as inputs, which means external devices cannot drive these lines. The calibration data will be sent to the programming jig by the application PIC16CXXX through RB6 and RB7. The programming jig will later use these lines to clock the calibration data into the application PIC16CXXX.

**Programming Issues**

The PIC16CXXX programming specification suggests verification of program memory at both Maximum and Minimum VDD for each device. This is done to ensure proper programming margins and to detect (and reject) any improperly programmed devices. All production quality programmers vary VDD from VDDmin to VDDmax after programming and verify the device under each of these conditions.

Since both the application voltage and its tolerances are known, it is not necessary to verify the PIC16CXXX calibration parameters at the device VDDmax and VDDmin. It is only necessary to verify at the application power supply Max and Min voltages. This application note shows the nominal (+5V) verification routine and hardware. If the power supply is a regulated +5V, this is adequate and no additional hardware or software is needed. If the application power supply is not regulated (such as a battery powered or poorly regulated system) it is important to complete a VDDmin and VDDmax verification cycle following the +5V verification cycle. See programming specifications for more details on VDD verification procedures.

- PIC16C5X Programming Specifications - DS30190
- PIC16C55X Programming Specifications - DS30261
- PIC16C6X/7X/9XX Programming Specifications - DS30228
- PIC16C84 Programming Specifications - DS30189

**Note:** The designer must consider environmental conditions, voltage ranges, and aging issues when determining VDD min/max verification levels. Please refer to the programming specification for the application device.

The calibration programming and initial verification MUST occur at +5V. If the application is intended to run at lower (or higher voltages), a second verification pass must be added where those voltages are applied to VDD and the device is verified.
Communication Format (Application Microcontroller to Programming Jig)

Unused program memory, in the application PIC16CXXX, is left unprogrammed as all 1s; therefore the unprogrammed program memory for the calibration look-up table would contain 3FFF (hex). This is interpreted as an “ADDLW FF”. The application microcontroller simply needs one “RETLW FF” instruction at the end of the space allocated in program memory for the calibration parameter look-up table. When the application microcontroller is powered up, it will receive a “FFh” for each calibration parameter that is looked up; therefore, it can detect that it is uncalibrated and jump to the calibration code.

Once the calibration constants are calculated by the application PICmicro MCU, they need to be communicated to the (PIC16C58A based) programming jig. This is accomplished through the RB6 and RB7 lines. The format is a simple synchronous clock and data format as shown in Figure 1.

A pull-down on the clock line is used to hold it low. The application microcontroller needs to send the high and low bytes of the target start address of the calibration constants to the calibration jig. Next, the data bytes are sent followed by a checksum of the entire data transfer as shown in Figure 1.

Once the data transfer is complete, the checksum is verified by the programming jig and the data printed at 9600 baud, 8-bits, no parity, 1 stop bit through RB3. A connection to this pin is optional. Next the programming jig applies +13V, programs and verifies the application PIC16CXXX calibration parameters.

FIGURE 1:
LED Operation

When the programming jig is waiting for communication from the application PICmicro MCU, both LEDs are OFF. Once a valid data stream is received (with at least one calibration byte and a correct checksum) the WORK LED is lit while the calibration parameters are printed through the optional RB3 port. Next, the DONE LED is lit to indicate that these parameters are being programmed and verified by the programming jig. Once the programming is finished, the WORK LED is extinguished and the DONE LED remains lit. If any parameters fail programming, the DONE LED is extinguished; therefore both LEDs would remain off.

FIGURE 2: ISP CALIBRATION JIG PROGRAMMER SCHEMATIC
Code Protection

Selection of the code protection configuration bits on PIC16CXXX microcontrollers prevents further programming of the program memory array. This would prevent writing self calibration parameters if the device is code protected prior to calibration. There are two ways to address this issue:

1. Do not code protect the device when programming it with the programmer. Add additional code (See the PIC16C6X/7X programming Spec) to the ISPPRGM.ASM to program the code protection bit after complete verification of the calibration parameters
2. Only code protect 1/2 or 3/4 of the program memory with the programmer. Place the calibration constants into the unprotected part of program memory.

Software Routines

There are two source code files needed for this application note:

1. ISPTEST.ASM (Appendix A) Contains the source code for the application PIC16CXXX, sets up the calibration look-up table and implements the communication protocol to the programming jig.
2. ISPPRGM.ASM (Appendix B) Source code for a PIC16C58A to implement the programming jig. This waits for and receives the calibration parameters from the application PIC16CXXX, places it into programming mode and programs/verifies each calibration word.

CONCLUSION

Typically, calibration information about a system is stored in EEPROM. For calibration data that does not change over time, the In-circuit Serial Programming capability of the PIC16CXXX devices provide a simple, cost effective solution to an external EEPROM. This method not only decreases the cost of a design, but also reduces the complexity and possible failure points of the application.

TABLE 1: PARTS LIST FOR PIC16CXXX ISP CALIBRATION JIG

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Reference</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>C1,C2</td>
<td>15 pF</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>C3</td>
<td>620 pF</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>C4</td>
<td>0.1 mF</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>C5,C6</td>
<td>220 mF</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>D1,D2</td>
<td>LED</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>E1</td>
<td>PIC16C58</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>E2</td>
<td>LM78S40</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>J1</td>
<td>CON5</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>L1</td>
<td>270 mH</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>Q1,Q2</td>
<td>2N2222</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>Q3,Q4</td>
<td>2N2907</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>R1,R2,R3,R4,R15</td>
<td>1k</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>R5,R6,R12,R14</td>
<td>10k</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>R7,R8</td>
<td>270</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>R9</td>
<td>180</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>R10</td>
<td>23.7k</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>R11</td>
<td>2.49k</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>R13</td>
<td>2.2k</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>Y1</td>
<td>4.0 MHz</td>
</tr>
</tbody>
</table>
APPENDIX A:

MPASM 01.40.01 Intermediate ISPPRGM.ASM 3-31-1997 10:57:03 PAGE 1

LOC OBJECT CODE LINE SOURCE TEXT
VALUE

00001 ; Filename: ISPPRGM.ASM
00002 ; **********************************************
00003 ; * Author:   John Day                         *
00004 ; *           Sr. Field Applications Engineer  *
00005 ; *           Microchip Technology             *
00006 ; * Revision: 1.0                              *
00007 ; * Date      August 25, 1995                  *
00008 ; * Part:     PIC16C58                         *
00009 ; * Compiled using MPASM V1.40               *
00100 ; **********************************************
00101 ; * Include files:                             *
00102 ; *           P16CSX.ASM                       *
00103 ; **********************************************
00104 ; * Fuses:    OSC:  XT (4.0 Mhz xtal)          *
00105 ; *           WDT:  OFF                        *
00106 ; *            CP:  OFF                        *
00107 
00108 ; *********************************************************************************
00109 ; This program is intended to be used as a self programmer
00110 ; to store calibration constants into a lookup table
00111 ; within the main system processor. A 4 Mhz crystal
00112 ; is needed and an optional 9600 baud serial port will
00113 ; display the parameters to be programmed.
00114 ;
00115 ; *********************************************************************************
00116 ; * Program Memory:                                                        *
00117 ; *    Words - communication with test jig                                 *
00118 ; * 17 Words - calibration look-up table (16 bytes of data)              *
00119 ; * 13 Words - Test Code to generate Calibration Constants                 *
00120 ; * RAM memory:                                                            *
00121 ; * 64 Bytes - Store up to 64 bytes of calibration constant               *
00122 ; * 9 Bytes - Store 9 bytes of temp variables (reused)                    *
00123 ;
00124 ; *********************************************************************************
00125     list p=16C58A
00126     include <p16C5x.inc>
00127     __CONFIG _CP_OFF&_WDT_OFF&_XT_OSC

00128 ; ************************************
00129 ; * Port A (RA0-RA4) bit definitions *
00130 ; ************************************
00131     ISPCLOCK      EQU 6   ; Clock line for ISP and parameter comm
00132     ISPDATA      EQU 7   ; Data line for ISP and parameter comm
00133     VPPON       EQU 5   ; Apply +13V VPP voltage to MCLR (test mode)
00134     GNDON       EQU 4   ; Apply +0V (gnd) voltage to MCLR (reset)
00135     SEROUT      EQU 3   ; Optional RS-232 TX output (needs 12V driver)
00136     DONELED     EQU 2   ; Turns on LED when done sucessfully program
00137     WORKLED     EQU 1   ; On during programming, off when done
00138 
00139     DONELED     EQU 2   ; Turns on LED when done sucessfully program
00140 
00141 
00142 ; *********************************************************************************
00143 ; * Port B (RB0-RB7) bit definitions *
00144 ; *********************************************************************************
00145     ISPCLOCK      EQU 6   ; Clock line for ISP and parameter comm
00146     ISPDATA      EQU 7   ; Data line for ISP and parameter comm
00147     VPPON       EQU 5   ; Apply +13V VPP voltage to MCLR (test mode)
00148     GNDON       EQU 4   ; Apply +0V (gnd) voltage to MCLR (reset)
00149     SEROUT      EQU 3   ; Optional RS-232 TX output (needs 12V driver)
00150     DONELED     EQU 2   ; Turns on LED when done sucessfully program
00151     WORKLED     EQU 1   ; On during programming, off when done
00152 
00153
00054 ; **********************************************************************************
00055 ; * RAM register definition: *
00056 ; * 07h - 0Fh - used for internal counters, vars *
00057 ; * 10h - 7Fh - 64 bytes for cal param storage *
00058 ; **********************************************************************************
00059 ; ***
00060 ; *** The following VARS are used during ISP programming:
00061 ; ***
00000007 00062 HIADDR    EQU 07h ; High address of CAL params to be stored
00000008 00063 LOADDR    EQU 08h ; Low address of CAL params to be stored
00000007 00064 HIDATA    EQU 07h ; High byte of data to be sent via ISP
00000008 00065 LODATA    EQU 08h ; Low byte of data to be sent via ISP
00000009 00066 HIBYTE    EQU 09h ; High byte of data received via ISP
0000000A 00067 LOBYTE    EQU 0Ah ; Low byte of data received via ISP
0000000B 00068 PULSECNT EQU 0Bh ; Number of times PIC has been pulse programmed
0000000C 00069 TEMPCOUNT EQU 0Ch ; TEMP var used in counters
0000000D 00070 TEMP      EQU 0Dh ; TEMP var used throughout program
000071 ; ***
000072 ; *** The following VARS are used to receive and store CAL params:
000073 ; ***
00000007 00074 COUNT     EQU 07h ; Counter var used to receive cal params
00000008 00075 TEMPI     EQU 08h ; TEMP var used for RS-232 comm
00000009 00076 DATAREG   EQU 09h ; Data register used for RS-232 comm
0000000A 00077 CSUMTOTAL EQU 0Ah ; Running total of checksum (addr + data)
0000000B 00078 TIMEHIGH  EQU 0Bh ; Count how long CLOCK line is high
0000000C 00079 TIMELOW   EQU 0Ch ; Count how long CLOCK line is low
0000000E 00080 ADDRPTR   EQU 0Dh ; Pointer to next byte of CAL storage
0000000F 00081 BYTECOUNT EQU 0Eh ; Number of CAL bytes received
000082
000083 ; *************************************
000084 ; * Various constants used in program *
000085 ; *************************************
000086 00086 DATISPOUT EQU b'00000001' ; tris settings for ISP data out
000087 00087 DATISPIN  EQU b'10000001' ; tris settings for ISP data in
000088 00088 CMDISPCNT EQU b'00000110' ; Number of bits for ISP command
000089 00089 STARTCALBYTE EQU 10h ; Address in RAM where CAL byte data stored
000090 00090 VFYYES    EQU PA2 ; Flag bit enables verification (STATUS)
000091 00091 CMDISPINCRADDR EQU b'00000110' ; ISP Pattern to increment address
000092 00092 CMDISPSTART EQU b'00001000' ; ISP Pattern to start programming
000093 00093 CMDISPPGMEND EQU b'00001110' ; ISP Pattern to end programming
000094 00094 CMDISPLOAD EQU b'00000010' ; ISP Pattern to load data for program
000095 00095 CMDISPSREAD EQU b'00000100' ; ISP Pattern to read data for verify
000096 00096 UPPER6BITS EQU 034h ; Upper 6 bits for retlw instruction
000097
000098 ; *************************************
000099 ; * delaybit macro *
00100 ; * Delays for 104 uS (at 4 Mhz clock)*
00101 ; * for 9600 baud communications *
00102 ; * RAM used: COUNT *
00103 ; *************************************
00104 delaybit macro
00105 local dylables
00106 ; 9600 baud, 8 bit, no parity, 104 us per bit, 52 uS per half bit
00107 ; (8) shift/usage + (2) setup + (1) nop + (3 * 31) literal = (104) 4Mhz
00108 movlw .31 ; place 31 decimal literal into count
00109 movwf COUNT ; Initialize COUNT with loop count
00110 nop ; Add one cycle delay
00111 dylables
00112 decfsz COUNT,F ; Decrement count until done
00113 goto dylables ; Not done delaying - go back!
00114 ENDM ; Done with Macro
00115
00116 ; *************************************
00117 ; * addrtosr macro *
00118 ; * Converts logical, continuous address 10h-4Fh *
00119 ; * to FSR address as follows for access to (4) *
00120 ; * banks of file registers in PIC16C58: *
00121 ; * Logical Address FSR Value *
00122 ; * 10h-1Fh 10h-1Fh *
00123 ; * 20h-2Fh 30h-3Fh *
00124 ; * 30h-3Fh 50h-5Fh *
00125 ; * 40h-4Fh 70h-7Fh *
00126 ; * Variable Passed: Logical Address *
00127 ; * RAM used: FSR *
00128 ; * W *
00129 ; ************************************************
00130 addrtofsr macro TESTADDR
00131     movlw   STARTCALBYTE        ; Place base address into W
00132     subwf   TESTADDR,w          ; Offset by STARTCALBYTE
00133     movwf   FSR                 ; Place into FSR
00134     btfsc   FSR,5               ; Shift bits 4,5 to 5,6
00135     bsf     FSR,6               
00136     bcf     FSR,5               
00137     btfsc   FSR,4               
00138     bsf     FSR,5               
00139     bsf     FSR,4               
00140     endm                           
00141                           
00142                           
00143 ; ************************************************
00144 ; * The PC starts at the END of memory *
00145 ; ************************************************
00146  ORG 7FfH
00147 goto start
00148                           
00149 ; ************************************************
00150 ; * Start of CAL param read routine *
00151 ; ************************************************
00152  ORG 0h
00153 start                           
00154 movlw b'00001010' ; Serial OFF, LEDS OFF, VPP OFF
00155 movwf PORTB ; Place "0" into port b latch register
00156 movlw b'11000001' ; RB7:RB6, RB0 set to inputs
00157 tris PORTB ; Move to tris registers
00158 clrw ; Place 0 into W
00159 clrf PORTA ; Place all ZERO into latch
00160 tris PORTA ; Make all pins outputs to be safe..
00161 clrf PORTA, GNDON ; TEST ONLY-RESET PIC-NOT NEEDED IN REAL DESIGN!
00162 clearram                           
00163 movlw 010h ; Place start of buffer into W
00164 movwf COUNT ; Use count for RAM pointer
00165 loopclrram                           
00166 addrtofsr COUNT ; Set up FSR
00167 movlw STARTCALBYTE ; Place base address into W
00168 movwf COUNT,w ; Offset by STARTCALBYTE
00169 movwf FSR ; Place into FSR
00170 btfsc FSR,5 ; Shift bits 4,5 to 5,6
00171 bsf FSR,6               
00172 bcf FSR,5               
00173 btfsc FSR,4               
00174 bsf FSR,5               
00175 bsf FSR,4               
00176 clrf INDF ; Clear buffer value
00177 incf COUNT,F ; Move to next reg
00178 movlw 050h ; Move end of buffer addr to W
00179 subwf COUNT,W ; Check if at last MEM
00180 0743 btfss STATUS,2 ; Skip when at end of counter
00181 goto loopclrram ; go back to next location
00182 0486 bcf PORTA, GNDON ; TEST ONLY-LET IT GO-NOT NEEDED IN REAL DESIGN!
00183 calget                           
00184 006A clrf CSUMTOTAL ; Clear checksum total byte
001B 0069 00176  clrf  DATAREG ; Clear out data receive register
001C 0010 00177  movlw  STARTCALBYTE ; Place RAM start address of first cal byte
001D 0002E 00178  movwf  ADDRPTR ; Place this into ADDRPTR
001E 0018D 00179  waitclockpulse
001E 007CE 00180  btfs  PORTB,ISPCLOCK ; Wait for CLOCK high pulse - skip when high
001F 00A1E 00181  goto  waitclockpulse ; CLOCK is low - go back and wait!
0020 00182  looopcal
0020 006CB 00183  movlw  .8 ; Place 8 into W (8 bits/byte)
0021 0027 00184  movwf  COUNT ; set up counter register to count bits
0022 00185  looopsendcal
0022 006BC 00186  clrf  TIMEHIGH ; Clear timeout counter for high pulse
0023 006CC 00187  clrf  TIMELOW ; Clear timeout counter for low pulse
0024 00188  waitclkhi
0024 006CF 00189  btfs  PORTB,ISPCLOCK ; Wait for CLOCK high - skip if it is low
0025 00A29 00190  goto  waitclklo ; Jump to wait for CLOCK low state
0026 002EB 00191  decfsz  TIMEHIGH,F ; Decrement counter - skip if timeout
0027 00A2A 00192  goto  waitclkhi ; Jump back and wait for CLOCK high again
0028 00A47 00193  goto  timeout ; Timed out waiting for high - check data!
0029 00194  waitclklo
0029 006C6 00195  btfss  PORTB,ISPCLOCK ; Wait for CLOCK low - skip if it is high
002A 002EC 00196  goto  clockok ; Got a high to low pulse - jump to clockok
002B 0047 00197  goto  timeout ; Timed out waiting for low - check data!
002C 00198  clockok
002C 00C08 00199  movlw  .8 ; Place initial count value into W
002D 0008E 00201  subwf  COUNT,W ; Subtract from count, place into W
002E 00743 00202  btfss  STATUS,Z ; Skip if we are at count 8 (first value)
002F 003E6 00203  btfss  PORTB,ISPDATA ; Skip if the data bit was high
0030 00403 00204  bcf  STATUS,C ; Set data bit to low
0031 00369 00205  rlf  DATAREG,F ; Rotate next bit into DATAREG
0032 002E7 00206  decfsz  COUNT,F ; Skip after 8 bits
0033 00A22 00207  goto  loopsendcal ; Jump back and send next bit
0034 00213 00208  clrf  STATUS,C ; Assume data bit is high
0035 007B6 00209  btfs  PORTB,ISPCLOCK ; Skip if the data bit was high
0036 00210 0020A 00210  bcf  STATUS,C ; Set data bit to low
0037 00369 00211  rlf  DATAREG,F ; Rotate next bit into DATAREG
0038 002E7 00212  decfs  COUNT,F ; Skip after 8 bits
0039 00A22 00213  goto  loopcal ; Jump back and send next bit
003A 0016A 00214  addwf  CSUMTOTAL,F ; Add to checksum
003B 00503 00215  bsf  STATUS,C ; Assume data bit is high
003C 00576 00216  btfs  PORTB,ISRDATA ; Skip if the data bit was high
003D 00403 00217  bcf  STATUS,C ; Set data bit to low
003E 00369 00218  rlf  DATAREG,F ; Rotate next bit into DATAREG
003F 002E7 00219  decfs  COUNT,F ; Skip after 8 bits
0040 00426 00220  movwf  INDF ; Move recvd byte into CAL buffer location
0041 00426 00221  movwf  INDF ; Move recvd byte into CAL buffer location
0042 00426 00222  movwf  ADDRPTR,F ; Move to the next cal byte
0043 002A0 00223  goto  loopcal ; Go back for next byte
0044 002A0 00224  goto  timeout ; check if we received (4) params
0045 002A0 00225  movf  ADDRPTR,W ; Move current address pointer to W
0046 002A0 00226  movf  ADDRPTR,W ; Move current address pointer to W
0047 002A0 00227  movf  ADDRPTR,W ; Move current address pointer to W
0048 002A0 00228  movf  ADDRPTR,W ; Move current address pointer to W
0049 002A0 00229  movf  ADDRPTR,W ; Move current address pointer to W
004A 002A0 0022A  movf  ADDRPTR,W ; Move current address pointer to W
004B 002A0 0022B  movf  ADDRPTR,W ; Move current address pointer to W
004C 002A0 0022C  movf  ADDRPTR,W ; Move current address pointer to W
004D 002A0 0022D  movf  ADDRPTR,W ; Move current address pointer to W
004E 002A0 0022E  movf  ADDRPTR,W ; Move current address pointer to W
004F 002A0 0022F  movf  ADDRPTR,W ; Move current address pointer to W
0050 002A0 00230  movlw  STARTCALBYTE ; Place start pointer into W
0051 002A0 00231  movlw  ADDRPTR,W ; Place start pointer into W
0052 002A0 00232  movlf  BYTECOUNT ; Place into number of bytes into BYTECOUNT
0053 002B 00233 movwf TIMEHIGH ; TEMP store into timehigh reg
0054 0C10 00234 movlw STARTCALBYTE ; Place start address into W
0055 002E 00235 movwf ADDRPTR ; Set up address pointer
0056 00236 loopprintnums
00237 addrtosfr ADDRPTR ; Set up FSR
0056 0C10 M movlw STARTCALBYTE ; Place base address into W
0057 008E M subwf TEMP,F ; Offset by STARTCALBYTE
0058 0024 M movw FSR ; Place into FSR
0059 06A4 M btfsc FSR,5 ; Shift bits 4,5 to 5,6
005A 05C4 M bcf FSR,5
005B 04A4 M btfsc FSR,4
005C 05A4 M bcf FSR,5
005D 0584 M bcf FSR,4
005E 0584 M bcf FSR,4
005F 0380 00238 swapf INDF,W ; Place received char into W
0060 0B0F 00239 andlw 0Ph ; Strip off upper digits
0061 002D 00240 movwf TEMP ; Place into TEMP
0062 0C0A 00241 movlw .10 ; Place .10 into W
0063 00AD 00242 subwf TEMP,F ; Subtract 10 from TEMP
0064 0603 00243 btfsc STATUS,C ; Skip if TEMP is less than 9
0065 0A6D 00244 goto printhiletter ; Greater than 9 - print letter instead
0066 00245 printhinumber
0066 0380 00246 swapf INDF,W ; Place received char into W
0067 0B0F 00247 andlw 0Ph ; Strip off upper digits
0068 002D 00248 movwf TEMP ; Place into TEMP
0069 0C0A 00249 movlw '0' ; Place ASCII '0' into W
006A 09AE 00250 call putchar ; Send out char
006B 0A6D 00251 goto printlo ; Jump to print next char
006C 0A73 00252 goto printloletter
006D 0A73 00253 printhiletter
006D 0380 00254 swapf INDF,W ; Place received char into W
006E 0B0F 00255 andlw 0Ph ; Strip off upper digits
006F 002D 00256 movwf TEMP ; Place into TEMP
0070 0C37 00257 movlw 'A'-.10 ; Place ASCII 'A' into W
0071 01CD 00258 addwf TEMP,w ; Add to TEMP, place into W
0072 09AE 00259 call putchar ; Send out char
0073 00260 printlo
0073 0200 00261 movwf INDW ; Print char
0074 0B0F 00262 andlw 0Ph ; Strip off upper digits
0075 002D 00263 movwf TEMP ; Place into TEMP
0076 0C0A 00264 movlw .10 ; Place .10 into W
0077 00AD 00265 subwf TEMP,F ; Subtract 10 from TEMP
0078 0603 00266 btfsc STATUS,C ; Skip if TEMP is less than 9
0079 0A81 00267 goto printloletter ; Greater than 9 - print letter instead
007A 00268 printlnumber
007A 0200 00269 movf INDW ; Print char
007B 0B0F 00270 andlw 0Ph ; Strip off upper digits
007C 002D 00271 movwf TEMP ; Place into TEMP
007D 0C37 00272 movlw '0' ; Place ASCII '0' into W
007E 01CD 00273 addwf TEMP,w ; Add to TEMP, place into W
007F 09AE 00274 call putchar ; Send out char
0080 0A87 00275 goto printnext ; Jump to print next char
0081 00276 printloletter
0081 0200 00277 movf INDW ; Print char
0082 0B0F 00278 andlw 0Ph ; Strip off upper digits
0083 002D 00279 movwf TEMP ; Place into TEMP
0084 0C37 00280 movlw 'A'-.10 ; Place ASCII 'A' into W
0085 01CD 00281 addwf TEMP,w ; Add to TEMP, place into W
0086 09AE 00282 call putchar ; Send out char
0087 00283 printnext
0087 0C7C 00284 movlw '|' ; Place ASCII '|' into W
0088 09AE 00285 call putchar ; Send out character
0089 028E 00286 incf ADDRPRTR,W ; Go to next buffer value
008A 0B0F 00287 andlw 0Ph ; And with F
008B 0643 00288 btfsc STATUS,Z ; Skip if this is NOT multiple of 16
008C  09A9  00289  call  PRINTCRLF  ; Print CR and LF every 16 chars
008D  02AE  00290  incf  ADDRPRTR,F  ; go to next address
008E  02EF  00291  decfsz  BYTECOUNT,F  ; Skip after last byte
008F  0A56  00292  goto  loopprintnums  ; Go back and print next char
0090  09A9  00293  call  PRINTCRLF  ; Print CR and LF
0091  05A3  00294  bsf  STATUS,PA0  ; Set page bit to page 1
0092  0A6B  00295  goto  programpartisp  ; Go to program part through ISP
0093  0C4E  00296  movlw  'N'  ; Place 'N' into W
0094  09AE  00298  call  PUTCHAR  ; Send char in W to terminal
0095  0C4F  00299  movlw  'O'  ; Place 'O' into W
0096  09AE  00300  call  putchar  ; Send char in W to terminal
0097  0C49  00301  movlw  'I'  ; Place 'I' into W
0098  09AE  00302  call  putchar  ; Send char in W to terminal
0099  0C53  00303  movlw  'S'  ; Place 'S' into W
009A  09AE  00304  call  putchar  ; Send char in W to terminal
009B  0C45  00305  movlw  'E'  ; Place 'E' into W
009C  09AE  00306  call  putchar  ; Send char in W to terminal
009D  09A9  00307  call  PRINTCRLF  ; Print CR and LF
009E  0A1A  00308  goto  calget  ; RESET!
009F  0C43  00309  movlw  'C'  ; Place 'C' into W
00A0  09AE  00310  call  PUTCHAR  ; Send char in W to terminal
00A1  0C53  00311  movlw  'S'  ; Place 'S' into W
00A2  09AE  00312  call  putchar  ; Send char in W to terminal
00A3  0C55  00313  movlw  'U'  ; Place 'U' into W
00A4  09AE  00314  call  putchar  ; Send char in W to terminal
00A5  0C4D  00315  movlw  'M'  ; Place 'M' into W
00A6  09AE  00316  call  putchar  ; Send char in W to terminal
00A7  09A9  00317  call  PRINTCRLF  ; Print CR and LF
00A8  0A1A  00318  goto  calget  ; RESET!
00A9  0C0D  00319  movlw  .13  ; Value for CR placed into W
00AA  09AE  00320  call  putchar  ; Send char in W to terminal
00AB  0C0A  00321  movlw  .10  ; Value for LF placed into W
00AC  09AE  00322  call  putchar  ; Send char in W to terminal
00AD  0800  00323  retlw  0  ; Done - return!
00320
00321  ; ******************************************************
00322  ; * printcrlf                                          *
00323  ; * Sends char .13 (Carrage Return) and               *
00324  ; * char .10 (Line Feed) to RS-232 port               *
00325  ; * by calling putchar.                               *
00326  ; * RAM used:  W                                       *
00327  ; ******************************************************
00A9  0328  00328  PRINTCRLF
00A9  0C0D  00329  movlw  .13  ; Value for CR placed into W
00AA  09AE  00330  call  putchar  ; Send char in W to terminal
00AB  0C0A  00331  movlw  .10  ; Value for LF placed into W
00AC  09AE  00332  call  putchar  ; Send char in W to terminal
00AD  0800  00333  retlw  0  ; Done - return!
00334
00335  ; ******************************************************
00336  ; * putchar                                           *
00337  ; * Print out the character stored in W               *
00338  ; * by toggling the data to the RS-232              *
00339  ; * output pin in software.                           *
00340  ; * RAM used:  W,DATAREG,TEMP1                        *
00341  ; ******************************************************
00AE  0342  00342  putchar
00AE  0029  00343  movwf  DATAREG  ; Place character into DATAREG
00AF  0C09  00344  movlw  09h  ; Place total number of bits into W
00B0  0028  00345  movwf  TEMP1  ; Init TEMP1 for bit counter
00B1  0403  00346  bcf  STATUS,C  ; Set carry to send start bit
00B2  0AB4  00347  goto  putloop1  ; Send out start bit
00B3  0329  00348  rrf  DATAREG,F  ; Place next bit into carry
00B4  0350  00349  putloop1
00B4  0703  00350  btfss  STATUS,C  ; Skip if carry was set
00B5  0466  00351  bcf  PORTB,SEOUT  ; Clear RS-232 serial output bit
00B6  0603  00353  btfsc  STATUS,C  ; Skip if carry was clear

Message[306]: Crossing page boundary -- ensure page bits are set.
00B7 0566 00354  bsf PORTB, SEROUT ; Set RS-232 serial output bit
00355  delaybit ; Delay for one bit time
0000  M local dlylabels
M ; 9600 baud, 8 bit, no parity, 104 us per bit, 52 uS per half bit
M ; (8) shift/usage + (2) setup + (1) nop + (3 * 31) literal = (104) 4Mhz
00B8 0C1F  M movlw .31 ; place 31 decimal literal into count
00B9 0027  M movwf COUNT ; Initialize COUNT with loop count
00BA 0000  M nop ; Add one cycle delay
00BB  M dlylabels
00BB 02E7  M decfsz COUNT, F ; Decrement count until done
00BC 0ABB  M goto dlylabels ; Not done delaying - go back!
00BD 02E8  bsf PORTB, SEROUT ; Send out stop bit
00359  delaybit ; delay for stop bit
0000  M local dlylabels
M ; 9600 baud, 8 bit, no parity, 104 us per bit, 52 uS per half bit
M ; (8) shift/usage + (2) setup + (1) nop + (3 * 31) literal = (104) 4Mhz
00C0 0C1F  M movlw .31 ; place 31 decimal literal into count
00C1 0027  M movwf COUNT ; Initialize COUNT with loop count
00C2 0000  M nop ; Add one cycle delay
00C3  M dlylabels
00C3 02E7  M decfsz COUNT, F ; Decrement count until done
00C4 0AC3  M goto dlylabels ; Not done delaying - go back!
00C5 0800  retlw 0 ; Done - RETURN
00360 ; ******************************************
00361 ; *******************************************************************
00362 ; * ISP routines from PICSTART-16C *
00363 ; * Converted from PIC17C42 to PIC16CSX code by John Day *
00364 ; * Originally written by Jim Pepping *
00365 ; *******************************************************************
00366 ; ******************************************
00367  ORG 200 ; ISP routines stored on page 1
00368 0200 0C08  bsf PORTB, VPPON ; Turn off VPP just in case!
00369 ; ******************************************
00370 ; * poweroffisp *
00371 ; * Power off application PIC - turn off VPP and reset device after *
00372 ; * programming pass is complete *
00373 ; ******************************************
00374 04A6 00375  bcf PORTB, VPPON ; Turn off VPP 13 volts
00376 0586 00376  bsf PORTB, GNDON ; Apply 0 V to MCLR to reset PIC
00377 0026 00377  movwf PORTB ; Place "0" into port b latch register
00378 048D 00378  bcf TEMP, 4 ; Turn off MCLR voltage to place application PIC into test mode. *
00379 05AD 00380  bsf TEMP, W ; Turn off WORK LED
00381 0206 0800  retlw 0 ; Done so return!
00382 ; ******************************************
00383 ; ********* testmodeisp *
00384 ; * this enables ISP programming to proceed *
00385 ; * RAM used: TEMP *
00386 ; ******************************************
00387 0207 00389  testmodeisp
00388 ; ******************************************
00389 0C08 00390  movlw b'00001000'; Serial OFF, LEDS OFF, VPP OFF
00391 0026 00391  movwf PORTB ; Place "0" into port b latch register
00392 04A6 00392  bcf PORTB, VPPON ; Turn off VPP just in case!
00393 0586 00393  bsf PORTB, GNDON ; Apply 0 volts to MCLR
00394 0026 00394  movlw b'00000001'; RB6,7 set to inputs
00395 0006 00395  tris PORTB ; Move to tris registers
00396 048D 00396  bcf PORTB, GNDON ; Allow MCLR to go back to 5 volts, deassert reset
00397 0526 00397  bsf PORTB, WORKLED ; Turn off WORK LED
00398 0800 00398  retlw 0 ; Done so return!
00399 ; ******************************************
00354 AN656
0213  0546  00402  bsf  PORTB,DONELED  ; Turn ON GREEN LED
0214  0800  00403  retlw 0  ; Done so return!
00404
00405  ; **************************************************************************
00406  ; * p16cispout                                                      *
00407  ; * Send 14-bit data word to application PIC for writing this data  *
00408  ; * to it's program memory.  The data to be sent is stored in both *
00409  ; * HIBYTE (6 MSBs only) and LOBYTE.                              *
00410  ; * RAM used:            TEMP, W, HIBYTE (inputs), LOBYTE (inputs) *
00411  ; **************************************************************************
0215  00412  P16cispout
0215  0C0E  00413  movlw .14  ; Place 14 into W for bit counter
0216  002D  00414  movwf  TEMP  ; Use TEMP as bit counter
0217  04C6  00415  bcf  PORTB,ISPCLOCK  ; Clear CLOCK line
0218  04E6  00416  bcf  PORTB,ISPDATA  ; Clear DATA line
0219  0C01  00417  movlw DATISPOUT  ; Place tris value for data output
021A  0006  00418  tris  PORTB  ; Set tris latch as data output
021B  04E6  00419  bcf  PORTB,ISPDATA  ; Send a start bit (0)
021C  04C6  00420  bcf  PORTB,ISPCLOCK  ; Set CLOCK output
021D  04C6  00421  bcf  PORTB,ISPCLOCK  ; Clear CLOCK output (clock start bit)
021E  00422  P16cispoutloop
021E  0403  00423  bcf  STATUS,C  ; Clear carry bit to start clean
021F  04C6  00424  bcf  PORTB,ISPDATA  ; Clear DATA bit to start (assume 0)
0220  0329  00425  rcf  HIBYTE,F  ; Rotate HIBYTE output
0221  032A  00426  rcf  LOBYTE,F  ; Rotate LOBYTE output
0222  0603  00427  btfsc  STATUS,C  ; Skip if data bit is zero
0223  05C6  00428  bcf  PORTB,ISPDATA  ; Set DATA line to send a one
0224  05C6  00429  bcf  PORTB,ISPCLOCK  ; Set CLOCK output
0225  04C6  00430  bcf  PORTB,ISPCLOCK  ; Clear CLOCK output (clock bit)
0226  02ED  00431  decfsz  TEMP,F  ; Decrement bit counter, skip when done
0227  0A1E  00432  goto  P16cispoutloop  ; Jump back and send next bit
0228  04E6  00433  bcf  PORTB,ISPDATA  ; Send a stop bit (0)
0229  05C6  00434  bcf  PORTB,ISPCLOCK  ; Set CLOCK output
022A  04C6  00435  bcf  PORTB,ISPCLOCK  ; Clear CLOCK output (clock stop bit)
022B  0800  00436  retlw 0  ; Done so return!
00437
00438  ; **************************************************************************
00439  ; * p16cispin                                                       *
00440  ; * Receive 14-bit data word from application PIC for reading this  *
00441  ; * data from it's program memory.   The data received is stored in *
00442  ; * both HIBYTE (output), LOBYTE (output)                           *
00443  ; **************************************************************************
022C  00445  P16cispin
022C  0C0E  00446  movlw .14  ; Place 14 data bit count value into W
022D  002D  00447  movwf  TEMP  ; Init TEMP and use for bit counter
022E  0069  00448  clrf  HIBYTE  ; Clear recieved HIBYTE register
022F  006A  00449  clrf  LOBYTE  ; Clear recieved LOBYTE register
0230  0403  00450  bcf  STATUS,C  ; Clear carry bit to start clean
0231  04C6  00451  bcf  PORTB,ISPCLOCK  ; Clear CLOCK output
0232  04E6  00452  bcf  PORTB,ISPDATA  ; Clear DATA output
0233  0C81  00453  movlw DATISPIN  ; Place tris value for data input into W
0234  0006  00454  tris  PORTB  ; Set up tris latch for data input
0235  05C6  00455  bcf  PORTB,ISPCLOCK  ; Send a single clock to start things going
0236  04C6  00456  bcf  PORTB,ISPCLOCK  ; Clear CLOCK to start receive
0237  00457  P16cispinloop
0237  05C6  00458  bcf  PORTB,ISPCLOCK  ; Set CLOCK bit
0238  0800  00459  nop  ; Wait one cycle
0239  0403  00460  bcf  STATUS,C  ; Clear carry bit, assume 0 read
023A  06B6  00461  btfsc  PORTB,ISPDATA  ; Check the data, skip if it was zero
023B  0503  00462  bcf  STATUS,C  ; Set carry bit if data was one
023C  0329  00463  rcf  HIBYTE,F  ; Move received bit into HIBYTE
023D  032A  00464  rcf  LOBYTE,F  ; Update LOBYTE
023E  04C6  00465  bcf  PORTB,ISPCLOCK  ; Clear CLOCK line
023F  0800  00466  nop  ; Wait one cycle
0240  0800  00467  nop  ; Wait one cycle

© 2003 Microchip Technology Inc.
DS00656B-page 4-13
0241 02ED 00468  decfsz TEMP,F ; Decrement bit counter, skip when zero
0242 0A37 00469  goto Pl6cispinloop ; Jump back and receive next bit
0243 05C6 00470  bsf PORTB,ISPCLOCK ; Clock a stop bit (0)
0244 0000 00471  nop ; Wait one cycle
0245 04C6 00472  bcf PORTB,ISPCLOCK ; Clear CLOCK to send bit
0246 0000 00473  nop ; Wait one cycle
0247 0403 00474  bcf STATUS,C ; Clear carry bit
0248 0329 00475  rrf HIBYTE,F ; Update HIBYTE with the data
0249 032A 00476  rrf LOBYTE,F ; Update LOBYTE
024A 0403 00477  bcf STATUS,C ; Clear carry bit
024B 0329 00478  rrf HIBYTE,F ; Update HIBYTE with the data
024C 032A 00479  rrf LOBYTE,F ; Update LOBYTE with the data
024D 04C6 00480  bcf PORTB,ISPCLOCK ; Clear CLOCK line
024E 04E6 00481  bcf PORTB,ISPDATA ; Clear DATA line
024F 0C01 00482  movlw DATISPOUT ; Place tris value for data output into W
0250 0006 00483  tris PORTB ; Set tris to data output
0251 0800 00484  retlw 0 ; Done so RETURN!

00485
00486 ; *******************************************************
00487 ; * commandisp                                        *
00488 ; * Send 6-bit ISP command to application PIC. The command is sent *
00489 ; * in the W register and later stored in LOBYTE for shifting. *
00490 ; * RAM used: LOBYTE, W, TEMP                          *
00491 ; *******************************************************

0252 0092 commandisp
0252 002A 00492  movwf LOBYTE ; Place command into LOBYTE
0253 0C06 00493  movlw CMDISPCNT ; Place number of command bits into W
0254 002D 00494  movwf TEMP ; Use TEMP as command bit counter
0255 04E6 00495  bcf PORTB,ISPDATA ; Clear DATA line
0256 04C6 00496  bcf PORTB,ISPCLOCK ; Clear CLOCK line
0257 0C01 00497  movlw DATISPOUT ; Place tris value for data output into W
0258 0006 00498  tris PORTB ; Set tris to data output
0259 0500 Pl6cispccmdoutloop
0259 0403 00500  bcf STATUS,C ; Clear carry bit to start clean
025A 04E6 00501  bcf PORTB,ISPDATA ; Clear the DATA line to start
025B 032A 00502  rrf LOBYTE,F ; Update carry with next CMD bit to send
025C 0603 00503  btfsc STATUS,C ; Skip if bit is supposed to be 0
025D 05E6 00504  bsf PORTB,ISPDATA ; Command bit was a one - set DATA to one
025E 05C6 00505  bcf PORTB,ISPCLOCK ; Set CLOCK line to clock the data
025F 0000 00506  nop ; Wait one cycle
0260 04C6 00507  bcf PORTB,ISPDATA ; Clear DATA line
0261 02ED 00508  decfsz TEMP,F ; Decement bit counter TEMP, skip when done
0262 0A59 00509  goto Pl6cispccmdoutloop ; Jump back and send next cmd bit
0263 0000 00510  nop ; Wait one cycle
0264 04E6 00511  bcf PORTB,ISPDATA ; Clear DATA line
0265 04C6 00512  bcf PORTB,ISPCLOCK ; Clear CLOCK line
0266 0C81 00513  movlw DATISPIN ; Place tris value for data input into W
0267 0006 00514  tris PORTB ; set as input to avoid any contention
0268 0000 00515  nop ; Wait one cycle
0269 0000 00516  nop ; Wait one cycle
026A 0800 00517  nop ; Wait one cycle
026A 0800 00518  retlw 0 ; Done - return!
00519
00520 ; *******************************************************
00521 ; * programpartisp                                     *
00522 ; * Main ISP programming loop. Reads data starting at STARTCALBYTE *
00523 ; * and calls programming subroutines to program and verify this *
00524 ; * data into the application PIC.                     *
00525 ; * RAM used: LOADDR, HIADDR, LODATA, HIDATA, FSR, LOBYTE, HIBYTE* *
00526 ; *******************************************************

026B 0027 programpartisp
026B 0907 00527  call testmodisp ; Place PIC into test/program mode
026C 0064 00528  clrf FSR ; Point to bank 0
026D 0210 00529  movf STARTCALBYTE,W ; Upper order address of data to be stored into W
026E 0027 00530  movwf HIADDR ; place into counter
026F 0211 00531  movf STARTCALBYTE+1,W ; Lower order address of data to be stored
0270 0028 00532  movwf LOADDR ; place into counter
0271 00E8 00534 decf LOADDR,F   ; Subtract one from loop constant
0272 02A7 00535 incf HIADDR,F   ; Add one for loop constant
0273 0063 programsetptr
0274 0C06 00537 movlw CMDISPINCRADDR ; Increment address command load into W
0275 0952 00538 call commandisp ; Send command to PIC
0276 02E8 00539 decfz LOADDR,F   ; Decrement lower address
0277 0A73 00540 goto programsetptr ; Go back again
0278 02E7 00541 decfz HIADDR,F   ; Decrement high address
0279 0A73 00542 goto programsetptr ; Go back again
027A 0C03 00543 movlw .3   ; Place start pointer into W, offset address
027B 008B 00544 subwf TIMEHIGH,W   ; Restore byte count into W
027C 002F 00545 movwf BYTECOUNT   ; Place into byte counter
027D 0C12 00546 movlw STARTCALBYTE+2   ; Place start of REAL DATA address into W
027E 02E2 00547 movwf ADDR PTR   ; Update pointer
027F 0068 programisloop
0280 0C34 00549 movlw UPPER6BITS ; retlw instruction opcode placed into W
0281 0027 00550 movwf H IDATA ; Set up upper bits of program word
0282 0C10 00551 addrtosr ADDR PTR ; Set up FSR to point to next value
0283 06A4 00552 movlw STARTCALBYTE ; Place base address into W
0284 05C4 00553 bsf FSR,6   ; Shift bits 4,5 to 5,6
0285 04A4 00554 bsf FSR,5
0286 0684 00555 btfsc FSR,4
0287 05A4 00556 bsf FSR,5
0288 0584 00557 bcf FSR,5
0289 0200 00558 movf INDF,W   ; Place next cal param into W
028A 0028 00559 movf LODATA ; Move it out to LODATA
028B 0028 00560 movf LODATA,W ; Place LODATA into LOBYTE
028C 002A 00561 movf LOBYTE   ;
028D 0027 00562 movf H IDATA,W ; Place H IDATA into H IBYTE
028E 0029 00563 movf H IBYTE   ;
028F 066B 00564 clr PULSE CN T ; Clear pulse counter
0290 0069 pgmispctloop
0291 05B3 00565 bsf STATUS, VFY YES ; Set verify flag
0292 09B1 00566 call pgmvfyisp ; Program and verify this byte
0293 02AB 00567 incf PULSE CN T,F ; Increment pulse counter
0294 0C19 00568 movlw .25   ; Place 25 count into W
0295 008B 00569 subsf PULSE CN T,W ; Subtract pulse count from 25
0296 0643 00570 bsf STATUS,Z ; Skip if NOT 25 pulse counts
0297 0AA9 00571 goto pgmispfail ; Jump to program failed - only try 25 times
0298 0209 00572 movf H IBYTE,W ; Subtract programmed and read data
0299 0087 00573 subwf H IDATA,W ; Subtract programmed and read data
029A 0743 00574 btfss STATUS,Z ; Skip if programmed is OK
029B 0AA9 00575 goto pgmispctloop ; Miscompare - program it again!
029C 082A 00576 movf LOBYTE,W ; Subtract programmed and read data
029D 0743 00577 btfss STATUS,Z ; Skip if programmed is OK
029E 0AA9 00578 goto pgmispctloop ; Miscompare - program it again!
029F 0040 00579 clr W ; Clear W reg
02A0 01CB 00576 addwf PULSE CN T,W ; now do 3 times overprogramming pulses
02A1 01CB 00577 addwf PULSE CN T,W
02A2 01CB 00578 addwf PULSE CN T,W
02A3 002B 00579 movwf PULSE CN T   ; Add 3X pulsecount to pulsecount
02A4 00580 pgmisp3X
02A5 04E3 00581 bcf STATUS,VFY YES ; Clear verify flag
02A6 09B1 00582 call pgmvfyisp ; Program this byte
02A7 0BBE 00583 decfz PULSE CN T,F ; Decrement pulse counter, skip when done
02A8 0AA9 00584 goto pgmisp3X ; Loop back and program again!
02A9 00585 goto progrnxtbyte ; Done - jump to program next byte!
02AA 00586 pgmispfail
02AB 0446 00587 bcf PORTB,DONE LED ; Failure - clear green LED!
02AC 00588 progrnxtbyte
02AD 0C06 00589 movlw CMDISPINCRADDR ; Increment address command load into W
02AE 0952 00590 call commandisp ; Send command to PIC
02AC 02AE 00591  incf ADDRPRTR,F ; Increment pointer to next address
02AD 02EF 00592  deecsz BYTECOUNT,F ; See if we sent last byte
02AE 0A7E 00593  goto programisploop ; Jump back and send next byte
02AF 0900 00594  call poweroffisp ; Done - power off PIC and reset it!
02B0 00595  self
02B0 0AB0 00596  goto self ; Done with programming - wait here!
00597
00598
00599
00600 ; *******************************************************************
00601 ; * pgmvfyisp                                                       *
00602 ; * Program and/or Verify a word in program memory on the          *
00603 ; * application PIC. The data to be programmed is in HIDATA and    *
00604 ; * LODATA.                                                         *
00605 ; * RAM used: HIBYTE, LOBYTE, HIDATA, LODATA, TEMP                  *
00606 ; *******************************************************************
02B1 00607  pgmvfyisp
02B1 00608  loadcisp
02B1 0C02 00609  movlw CMDISPLLOAD ; Place load data command into W
02B2 0952 00610  call commandisp ; Send load data command to PIC
02B3 0000 00611  nop ; Wait one cycle
02B4 0000 00612  nop ; Wait one cycle
02B5 0000 00613  nop ; Wait one cycle
02B6 0208 00614  movf LODATA,w ; Place LODATA byte into W
02B7 0000 00615  nop ; Wait one cycle
02B8 0020 00616  movf HIDATA,w ; Place HIDATA byte into W
02B9 0029 00617  movwf HIBYTE ; Move it to HIBYTE reg
02BA 0915 00618  call P16cispout ; Send data to PIC
02BB 0C08 00619  movlw CMDISPPGMSTART ; Place start programming command into W
02BC 0952 00620  call commandisp ; Send start programming command to PIC
02BD 00621  delay100us
02BD 0C20 00622  movlw .32 ; Place 32 into W
02BE 0000 00623  nop ; Wait one cycle
02BF 002D 00624  movwf TEMP ; Move it to TEMP for delay counter
02C0 0025 00625  loopprgm
02C0 020D 00626  deecsz TEMP,F ; Decrement TEMP, skip when delay done
02C1 0AC0 00627  goto loopprgm ; Jump back and loop delay
02C2 0C0E 00628  movlw CMDISPPGMEND ; Place stop programming command into W
02C3 0952 00629  call commandisp ; Send end programming command to PIC
02C4 07E3 00630  btfss STATUS,VFYYES ; Skip if we are supposed to verify this time
02C5 0800 00631  retlw 0 ; Done - return!
02C6 0000 00632  nop ; Wait one cycle
02C7 00633  readcisp
02C7 0C04 00634  movlw CMDISPRDREAD ; Place read data command into W
02C8 0952 00635  call commandisp ; Send read data command to PIC
02C9 092C 00636  call P16cispin ; Read programmed data
02CA 0800 00637  retlw 0 ; Done - return!
MEMORY USAGE MAP (‘X’ = Used, ‘-’ = Unused)

0000 : XXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX
00C0 : XXXXX------------- ---------------- ---------------- ----------------
0200 : XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX
0240 : XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX
0280 : XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXX
02C0 : XXXX---- ----------------- ----------------- -----------------
07C0 : ----------------- ----------------- ----------------- ---------------X
0FC0 : ----------------- ----------------- ----------------- ---------------X

All other memory blocks unused.

Program Memory Words Used: 402
Program Memory Words Free: 1646

Errors : 0
Warnings : 0 reported, 0 suppressed
Messages : 2 reported, 0 suppressed
APPENDIX B:
MPASM 01.40.01 Intermediate ISPTEST.ASM 3-31-1997 10:55:57 PAGE 1

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJECT CODE</th>
<th>LINE SOURCE</th>
<th>TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>; Filename: ISPTEST.ASM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0002</td>
<td>; ************************************************************</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>; * Author: John Day</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>; * Sr. Field Applications Engineer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0005</td>
<td>; * Microchip Technology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0006</td>
<td>; * Revision: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0007</td>
<td>; * Date August 25, 1995</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0008</td>
<td>; * Part: PIC16CXX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0009</td>
<td>; * Compiled using MPASM V1.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>; ************************************************************</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>; * Include files:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0012</td>
<td>; * P16CXX.ASM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013</td>
<td>; ************************************************************</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0014</td>
<td>; * Fuses: OSC: XT (4.0 Mhz xtal)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0015</td>
<td>; * WDT: OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0016</td>
<td>; * CP: OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0017</td>
<td>; * PWRT: OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0018</td>
<td>; ************************************************************</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0019</td>
<td>; * This program is intended to be used as a code example to</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0020</td>
<td>; * show how to communicate with a manufacturing test jig that</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0021</td>
<td>; * allows this PIC16CXX device to self program. The RB6 and RB7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0022</td>
<td>; * lines of this PIC16CXX device are used to clock the data from</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0023</td>
<td>; * this device to the test jig (running ISPPRGM.ASM). Once the</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0024</td>
<td>; * PIC16C58 running ISPPRGM in the test jig receives the data,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0025</td>
<td>; * it places this device in test mode and programs these parameters.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0026</td>
<td>; * The code with comments &quot;TEST - &quot; is used to create some fakecalibration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0027</td>
<td>; * parameters that are first written to addresses STARTCALBYTE through</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0028</td>
<td>; * ENDCALBYTE and later used to call the self-programming algorithm.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0029</td>
<td>; * Replace this code with your parameter calculation procedure,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0030</td>
<td>; * placing each parameter into the STARTCALBYTE to ENDCALBYTE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0031</td>
<td>; * file register addresses (16 are used in this example). The address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0032</td>
<td>; * lookuptable is used by the main code later on for the final lookup</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0033</td>
<td>; * table</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0034</td>
<td>; ************************************************************</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0035</td>
<td>; Program Memory;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0036</td>
<td>; * 49 Words - communication with test jig</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0037</td>
<td>; * 17 Words - calibration look-up table (16 bytes of data)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0038</td>
<td>; * 13 Words - Test Code to generate Calibration Constants</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0039</td>
<td>; * RAM Memory;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0040</td>
<td>; * 16 Bytes - Temporary- Store 16 bytes of calibration constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0041</td>
<td>; * 4 Bytes - Temporary- Store 4 bytes of temp variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0042</td>
<td>; ************************************************************</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Warning[217]: Hex file format specified on command line.
AN656

00056 ; *******************************************************
00057 #define     CLOCK   6 ; clock line for ISP
00058 #define     DATA    7 ; data line for ISP
00059 ; Port pins RB0-5 are not used in this test program
00060
00061 ; *******************************************************
00062 ; * RAM register usage definition  *
00063 ; *******************************************************
00064 CSUMTOTAL   EQU 0Ch ; Address for checksum var
00065 COUNT      EQU 0Dh ; Address for COUNT var
00066 DATAREG    EQU 0Eh ; Address for Data output register var
00067 COUNTDLY   EQU 0Fh ; Address for clock delay counter
00068
00069 ; These two symbols are used for the start and end address
00070 ; in RAM where the calibration bytes are stored. There are 16 bytes
00071 ; to be stored in this example; however, you can increase or
00072 ; decrease the number of bytes by changing the STARTCALBYTE or ENDCALBYTE
00073 ; address values.
00074
00075 STARTCALBYTE    EQU 10h     ; Address pointer for start CAL byte
00076 ENDCALBYTE      EQU 2Fh     ; Address pointer for end CAL byte
00077
00078 ; Table length of lookup table (number of CAL parameters to be stored)
00079
00080 CALTABLELENGTH EQU  ENDCALBYTE - STARTCALBYTE + 1
00081
00082 ORG 0
00083 ; *******************************************************
00084 ; * testcode routine                                           *
00085 ; * TEST code - sets up RAM register with register address as data *
00086 ; * Uses file register STARTCALBYTE through ENDCALBYTE to store the*
00087 ; * calibration values that are to be programmed into the lookup    *
00088 ; * table by the test jig running ISPPRGM.                       *
00089 ; * Customer would place calibration code here and make sure that  *
00090 ; * calibration constants start at address STARTCALBYTE            *
00091 ; *******************************************************
00092 testcode
00093  movlw   STARTCALBYTE    ; TEST -
00094  movwf   FSR             ; TEST - Init FSR with start of RAM addres
00095
00096  looptestram
00097  movf    FSR,W           ; TEST - Place address into W
00098  movwf   INDF            ; TEST - Place address into RAM data byte
00099  incf    FSR,F           ; TEST - Move to next address
00100  movf    FSR,W           ; TEST - Place current address into W
00101  sublw   ENDCALBYTE+1    ; TEST - Subtract from end of RAM
00102  btfss   STATUS,Z        ; TEST - Skip if at END of ram
00103  goto    looptestram     ; TEST - Jump back and init next RAM byte
00104  clrw                    ; TEST - Clear W
00105  call    lookuptable     ; TEST - Get first CAL value from lookup table
00106  sublw   0FFh            ; TEST - Check if lookup CAL table is blank
00107  btfsc   STATUS,Z        ; TEST - Skip if table is NOT blank
00108  goto    calsend         ; TEST - Table blank - send out cal parameters
00109  mainloop
00110  goto    mainloop        ; TEST - Jump back to self since CAL is done
00111 ; *******************************************************
00112 ; * lookuptable                                                  *
00113 ; * Calibration constants look-up table. This is where the CAL    *
00114 ; * Constants will be stored via ISP protocol later. Note it is    *
00115 ; * blank, since these values will be programed by the test jig    *
00116 ; * running ISPPRG later.                                        *
00117 ; * Input Variable:  W stores index for table lookup             *
00118 ; * Output Variable:  W returns with the calibration constant     *
00119 ; * NOTE: Blank table when programed reads "FF" for all locations *
00120 ; *******************************************************
00121 lookuptable
000F 0782    00122   addwf  FCL,F ; Place the calibration constant table here!
00123
002F 00124    ORG  lookuptable + CALTABLELENGTH
002F 34FF    00125   retrlw  0FFh ; Return FF at last location for a blank table
00126
00127 ; *******************************************************
00128 ; * calsend subroutine                                 *
00129 ; * Send the calibration data stored in locations STARTCALBYTE  *
00130 ; * through ENDCALBYTE in RAM to the programming jig using a serial*
00131 ; * clock and data protocol                            *
00132 ; * Input Variables:  STARTCALBYTE through ENDCALBYTE  *
00133 ; *******************************************************
0030 00134   calsend
0030 018C    00135   clrf    CSUMTOTAL       ; Clear CSUMTOTAL reg for delay counter
0031 018D    00136   clrf    COUNT           ; Clear COUNT reg to delay counter
0032 0137    00137   delayloop                   ; Delay for 100 mS to wait for prog jig wakeup
0032 0B8D    00138   decfsz  COUNT,F         ; Decrement COUNT and skip when zero
0033 2832    00139   goto    delayloop       ; Go back and delay again
0034 0B8C    00140   decfsz  CSUMTOTAL,F     ; Decrement CSUMTOTAL and skip when zero
0035 2832    00141   goto    delayloop       ; Go back and delay again
0036 0186    00142   clrf    PORTB           ; Place "0" into port b latch register
0037 1683    00143   bsf     STATUS,RP0      ; Switch to bank 1
0038 303F    00144   movlw   b'00111111'     ; RB6,7 set to outputs
0039 0086    00145   movwf   TRISB           ; Move to TRIS registers
003A 1283    00146   bcf     STATUS,RP0      ; Switch to bank 0
003B 018C    00147   clrf    CSUMTOTAL       ; Clear CSUMTOTAL reg for delay counter
003C 3001    00148   movlw   high lookuptable+1 ; place MSB of first addr of cal table into W
003D 204D    00149   call    sendcalbyte     ; Send the high address out
003E 3010    00150   movlw   low lookuptable+1  ; place LSB of first addr of cal table into W
003F 204D    00151   call    sendcalbyte     ; Send low address out
0040 3010    00152   movlw   STARTCALBYTE    ; Place RAM start address of first cal byte
0041 0084    00153   movwf   FSR             ; Place this into FSR
0042 0184    00154   loopcal
0042 0800    00155   movf    INDF,W          ; Place data into W
0043 204D    00156   call    sendcalbyte     ; Send the byte out
0044 0A84    00157   incf    FSR,F           ; Move to the next cal byte
0045 008D    00158   movwf   COUNT           ; set up counter register
0046 3C30    00159   sublw   .8              ; Place 8 into W
0047 1D03    00160   btfss   STATUS,Z        ; Skip if we are done
0048 2842    00161   goto    loopcal         ; Go back for next byte
0049 080C    00162   movwf   CSUMTOTAL,W     ; place checksum total into W
004A 204D    00163   call    sendcalbyte     ; Send the checksum out
004B 0186    00164   clrf    PORTB           ; clear out port pins
004C 0165    00165   calsenddone
004C 284C    00166   goto    calsenddone   ; We are done - go home!
00167
00168 ; *******************************************************
00169 ; * sendcalbyte subroutine                              *
00170 ; * Send one byte of calibration data to the programming jig *
00171 ; * Input Variable:  W contains the byte to be sent     *
00172 ; *******************************************************
004D 00137   sendcalbyte
004D 008E    00145   movwf   DATAREG         ; Place send byte into data register
004E 078C    00146   addwf  CSUMTOTAL,F ; Update checksum total
004F 300B    00147   movlw  .8              ; Place 8 into W
0050 008D    00148   movwf   COUNT           ; set up counter register
0051 1706    00149   bsf    PORTB,CLOCK     ; Set clock line high
0052 205C    00150   call    delaysend       ; Wait for test jig to synch up
0053 0D8E    00151   rlf     DATAREG,F       ; Rotate to next bit
0054 1786    00152   bsf    PORTB,DATA      ; Assume data bit is high
0055 1C03    00153   btfss   STATUS,C        ; Skip if the data bit was high
0056 1386    00154   bcf    PORTB,DATA      ; Set data bit to low
0057 1306    00155   bcf    PORTB,CLOCK     ; Clear clock bit to clock data out
0058 205C    00156   call    delaysend       ; Wait for test jig to synch up
0059 0B8D 00187  decfsz COUNT,F ; Skip after 8 bits
005A 2851 00188  goto  loopsendcal ; Jump back and send next bit
005B 0008 00189  return ; We are done with this byte so return!
00190
00191 ; ******************************************************************
00192 ; * delaysend subroutine                                           *
00193 ; * Delay for 50 ms to wait for the programming jig to synch up    *
00194 ; ******************************************************************
005C 00195 delaysend
005C 3010 00196  movlw  10h             ; Delay for 16 loops
005D 008F 00197  movwf  COUNTDLY        ; Use COUNTDLY as delay count variable
005E 00198 loopdelaysend
005E 0B8F 00199  decfsz  COUNTDLY,F      ; Decrement COUNTDLY and skip when done
005F 285E 00200  goto  loopdelaysend   ; Jump back for more delay
0060 0008 00201  return
00202         END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX ---------------- ---------------X XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX X--------------- ----------------
2000 : -------X-------- ---------------- ---------------- ----------------
All other memory blocks unused.

Program Memory Words Used:    66
Program Memory Words Free:   958

Errors :     0
Warnings :  1 reported,     0 suppressed
Messages :  1 reported,     0 suppressed