Featuring Microchip's Exclusive 2-Wire Serial Programming Capability

In-Circuit Serial Programming™ Guide

MICROCHIP

In-Circuit Serial Programming (ICSP™) Guide

Table of Contents

SECTION 1
IN-CIRCUIT SERIAL PROGRAMMING (ICSP™) GUIDE INTRODUCTION

ICSP is a trademark of Microchip Technology Inc.

INTRODUCTION

In-Circuit Serial Programming (ICSP™) Guide

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 8-bit 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's 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 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 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 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, re-calibration 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 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 pre-programmed, 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.

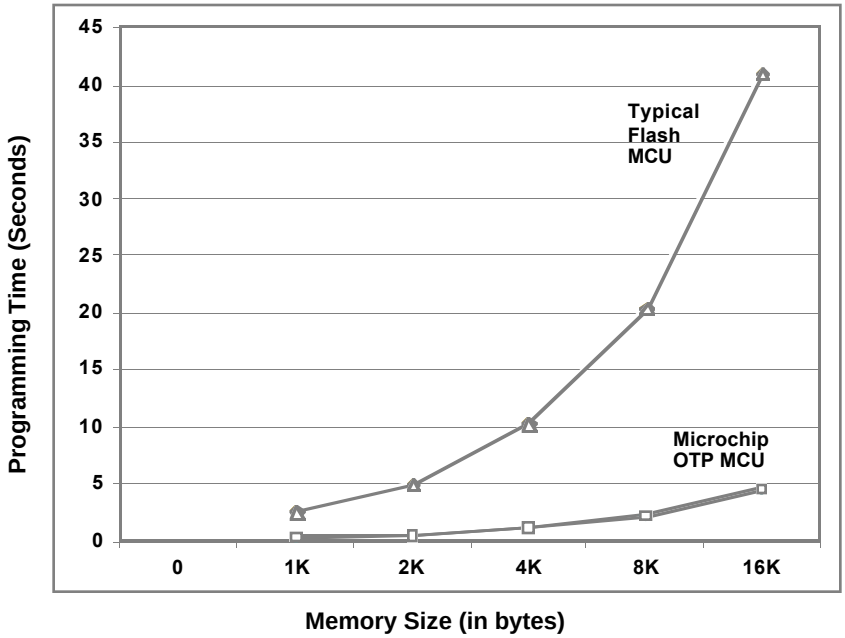
PROGRAMMING TIME CONSIDERATIONS

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.

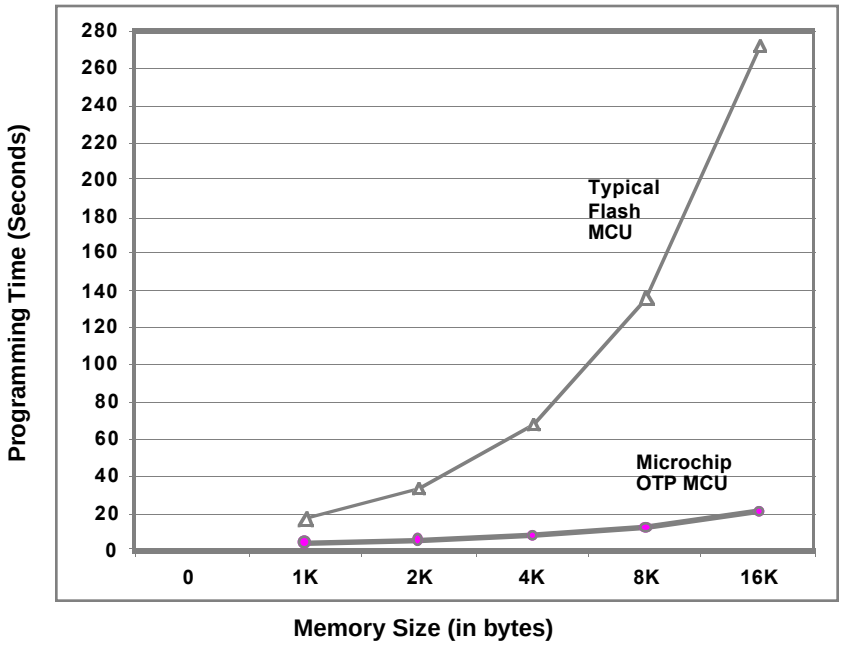
FIGURE 1: PROGRAMMING TIME FOR FLASH AND OTP MCUS (THEORETICAL MINIMUM TIMES)



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”.

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



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

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

Ramifications

The programming time differences between FLASH and OTP MCUs are not particularly 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
- PIC16CXXX OTP, Mid-Range Family
- PIC17CXXX OTP High-End Family
- PIC16F8X 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 mid-volume production. PRO MATE II runs under Windows® 95 and Windows 3.1, and can run under DOS in a Command Line Mode. PRO MATE II is also capable of Serialized Quick Turn Programming (SQTP), where each device can be programmed with up to 16 bytes of unique code.

PICSTART® Plus Development Programmer - supports ICSP and is recommended for development and prototyping only.

PRO MATE II ISP Kit - Microchip is currently developing a complete kit including connectors, cables and required interface boards to allow you to implement ICSP with PRO MATE II with minimal effort.

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.

NOTES:

SECTION 2
IN-CIRCUIT SERIAL PROGRAMMING (ICSP™) TECHNICAL BRIEFS

TB017

How to Implement ICSP™ Using PIC12C5XX OTP MCUs

Author: *Thomas Schmidt*
Microchip Technology Inc.

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 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 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.

IN-CIRCUIT SERIAL PROGRAMMING

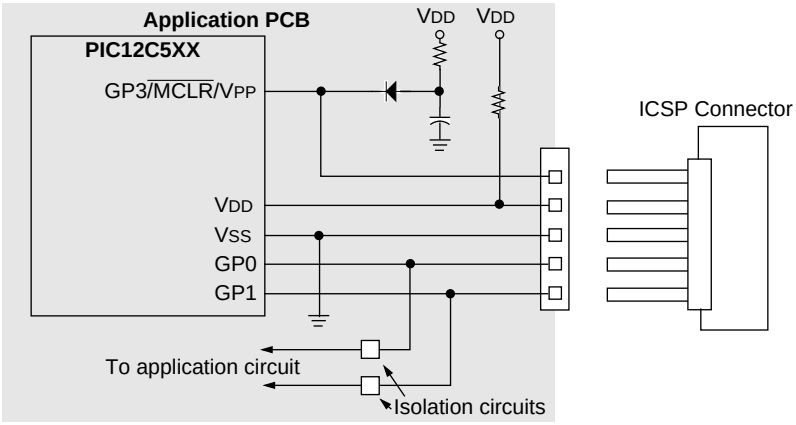
To implement ICSP into an application, the user needs to consider three main components of an ICSP system: Application Circuit, Programmer and Programming Environment.

Application Circuit

During the initial design phase of the application circuit, certain considerations have to be taken into account. Figure 1 shows and 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/$\overline{\text{MCLR}}$/V_{PP} 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 V_{DD}, GP3/$\overline{\text{MCLR}}$/V_{PP}, GP1, and GP0 pins.
4. Interface to the programmer.
5. Minimum and maximum operating voltage for V_{DD}.

FIGURE 1: TYPICAL APPLICATION CIRCUIT



PRO MATE and PICSTART are registered trademarks and PICmicro and ICSP are trademarks of Microchip Technology Inc.

Isolation of the GP3/$\overline{MCLR}$/VPP Pin from the Rest of the Circuit

PIC12C5XX devices have two ways of configuring the $\overline{MCLR}$ pin:

- $\overline{MCLR}$ can be connected either to an external RC circuit or
- $\overline{MCLR}$ is tied internally to VDD

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

For more information about configuring the GP3/$\overline{MCLR}$/VPP internally to VDD, please refer to the PIC12C5XX data sheet (DS40139).

Isolation of Pins GP1 and GP0 from the Rest of the Circuit

Pins GP1 and GP0 are used by the PICmicro for serial programming. GP1 is the clock line and GP0 is the data line.

GP1 is driven by the programmer. GP0 is a bi-directional pin that is driven by the programmer when programming and driven by the PICmicro 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 GP1 and GP0 from the rest of the circuit. This isolation circuit must account for GP1 being an input on the PICmicro and for GP0 being bi-directional pin.

For example, PRO MATE® II has an output impedance of 1 kΩ. If the design permits, these pins should not be used by the application. This is not the case with most designs. As a designer, you must consider what type of circuitry is connected to GP1 and GP0 and then make a decision on how to isolate these pins.

Total Capacitance on VDD, GP3/$\overline{MCLR}$/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 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 during programming. The other point of consideration is that the device must be verified at minimum and maximum operation 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 VDD levels are used to verify the memory contents of the PICmicro. 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 PIC12C5XX 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 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 uses spring loaded test pins (often referred 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

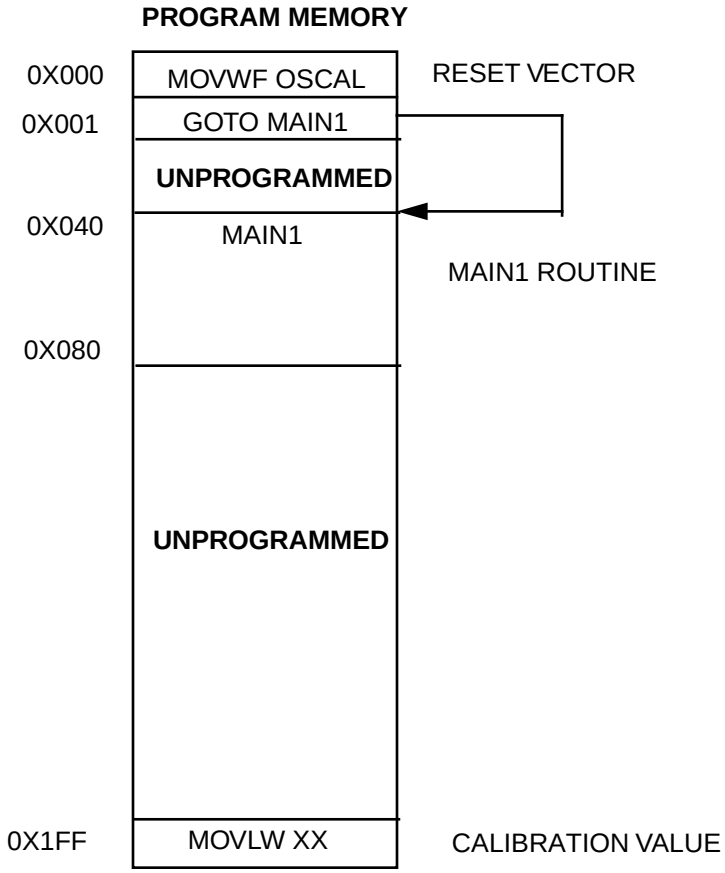
ICSP provides several other benefits such as calibration and serialization. If program memory permits, it would be cheaper and more reliable to store calibration constants in program memory instead of using an external serial EEPROM.

Field Programming of PICmicro OTP MCUs

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

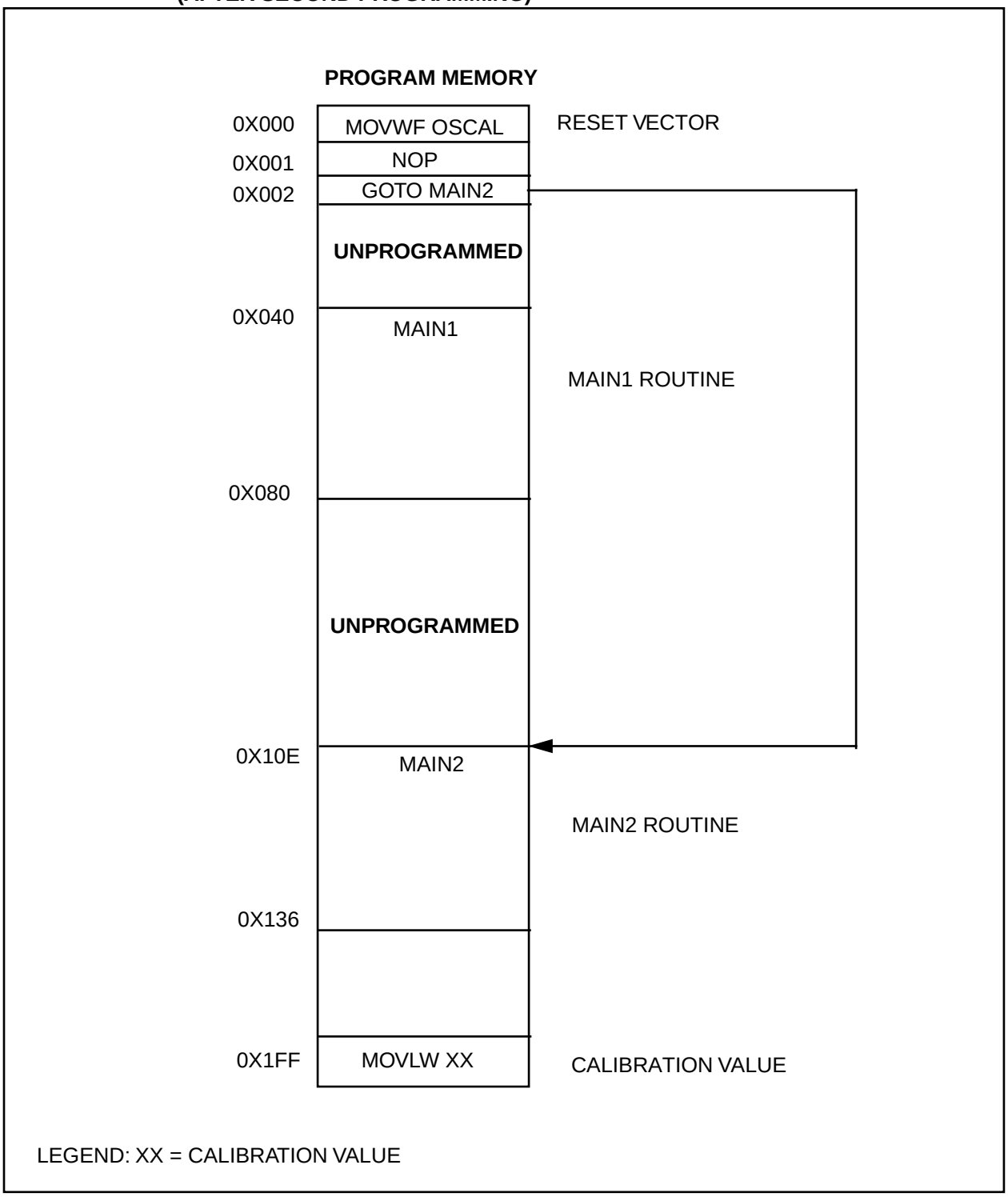
This method involves using jump tables for the reset and interrupt vectors. Example 1 shows the location of a main routine and the reset vector for the first time a device with 0.5K-words of program memory is programmed. Example 2 shows the location of a second main routine and its reset vector for the 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



LEGEND: XX = CALIBRATION VALUE

EXAMPLE 2: LOCATION OF THE SECOND MAIN ROUTINE AND IT INTERRUPT VECTOR (AFTER SECOND PROGRAMMING)

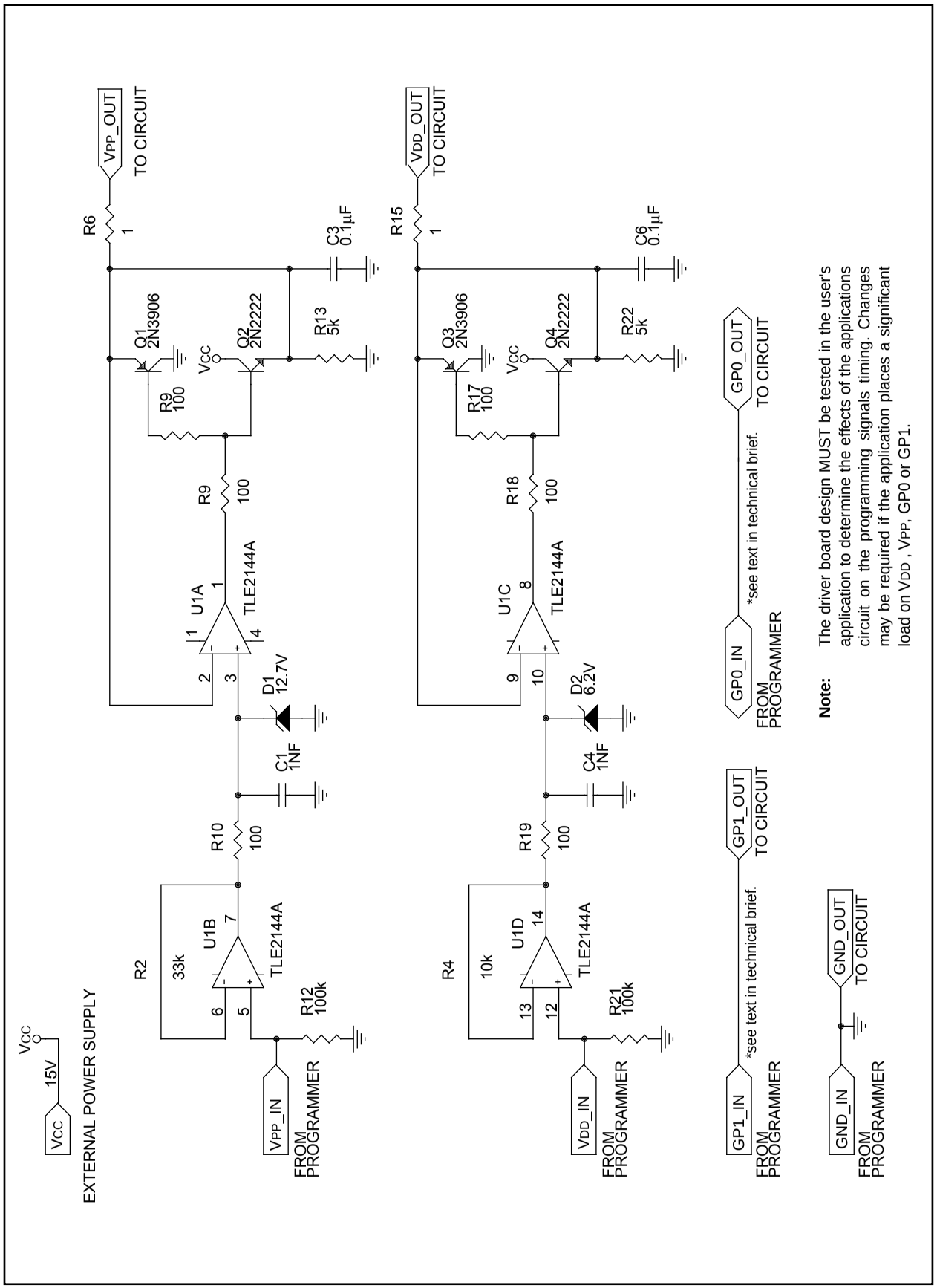


Since the program memory of the PIC12C5XX devices is organized in 256 x 12 word pages, placement of such information as look-up tables and CALL instructions must be taken into account. For further information, please refer to application note *AN581, Implementing Long Calls* and application note *AN556,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



NOTES:

TB013

How to Implement ICSP™ Using PIC16CXXX OTP MCUs

Author: Rodger Richey
Microchip Technology Inc.

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.

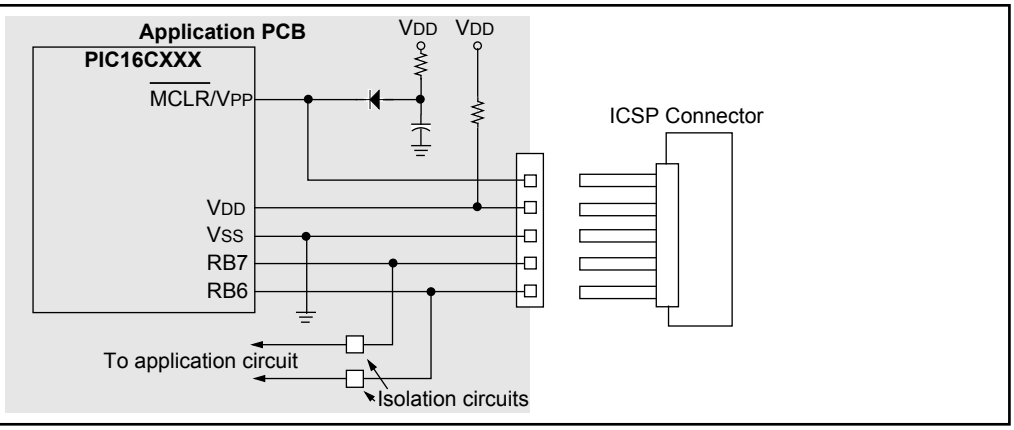
Application Circuit

The application circuit must be designed to allow all the programming signals to be directly connected to the PICmicro. 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 $\overline{MCLR}/V_{PP}$ 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 V_{DD}, $\overline{MCLR}/V_{PP}$, RB6, and RB7 pins.
4. Minimum and maximum operating voltage for V_{DD}.
5. PICmicro Oscillator.
6. Interface to the programmer.

The $\overline{MCLR}/V_{PP}$ pin is normally connected to an RC circuit. The pull-up resistor is tied to V_{DD} 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 $\overline{MCLR}/V_{PP}$. The diode should be a Schottky-type device. Another issue with $\overline{MCLR}/V_{PP}$ is that when the PICmicro 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 for serial programming. RB6 is the clock line and RB7 is the data line. RB6 is driven by the programmer. RB7 is a bi-directional pin that is driven by the programmer when programming, and driven by the PICmicro 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, and for RB7 being bi-directional (can be driven by both the PICmicro 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 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 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 option over the voltage range of the system.

This final issue deals with the oscillator circuit on the application board. The voltage on $\overline{MCLR}$/VPP must rise to the specified program mode entry voltage before the device executes any code. The crystal modes available on the PICmicro 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 $\overline{MCLR}$/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 $\overline{MCLR}$/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 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. 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 micrcontroller 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 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 in programming mode and programs the PICmicro 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 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 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. The second listing of Example 1 shows the code that is programmed into the PICmicro for the second time.

EXAMPLE 1: PROGRAMMING CYCLE LISTING FILES

```
       First Program Cycle                              Second Program Cycle
_____________________________________________________________________________________________
Prog   Opcode  Assembly                    |  Prog   Opcode  Assembly
Mem            Instruction                 |  Mem            Instruction
---------------------------------------------------------------------------------------------
0000   2808    goto  Main        ;Main loop  |  0000   0000    nop
0001   3FFF    <blank>           ;at 0x0008  |  0001   2860    goto  Main         ;Main now
0002   3FFF    <blank>                       |  0002   3FFF    <blank>            ;at 0x0060
0003   3FFF    <blank>                       |  0003   3FFF    <blank>
0004   2848    goto  ISR         ;ISR at     |  0004   0000    nop
0005   3FFF    <blank>           ;0x0048     |  0005   28A8    goto  ISR          ;ISR now at
0006   3FFF    <blank>                       |  0006   3FFF    <blank>            ;0x00A8
0007   3FFF    <blank>                       |  0007   3FFF    <blank>
0008   1683    bsf   STATUS,RP0              |  0008   1683    bsf   STATUS,RP0
0009   3007    movlw 0x07                    |  0009   3007    movlw 0x07
000A   009F    movwf ADCON1                  |  000A   009F    movwf ADCON1
 .                                           |   .
 .                                           |   .
 .                                           |   .
0048   1C0C    btfss PIR1,RBIF               |  0048   1C0C    btfss PIR1,RBIF
0049   284E    goto  EndISR                  |  0049   284E    goto  EndISR
004A   1806    btfsc PORTB,0                 |  004A   1806    btfsc PORTB,0
 .                                           |   .
 .                                           |   .
 .                                           |   .
0060   3FFF    <blank>                       |  0060   1683    bsf   STATUS,RP0
0061   3FFF    <blank>                       |  0061   3005    movlw 0x05
0062   3FFF    <blank>                       |  0062   009F    movwf ADCON1
 .                                           |   .
 .                                           |   .
 .                                           |   .
00A8   3FFF    <blank>                       |  00A8   1C0C    btfss PIR1,RBIF
00A9   3FFF    <blank>                       |  00A9   28AE    goto  EndISR
00AA   3FFF    <blank>                       |  00AA   1806    btfsc PORTB,0
 .                                           |   .
 .                                           |   .
 .                                           |   .
---------------------------------------------------------------------------------------------
```


The example shows that to program the PICmicro 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 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 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 PICmicros 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.

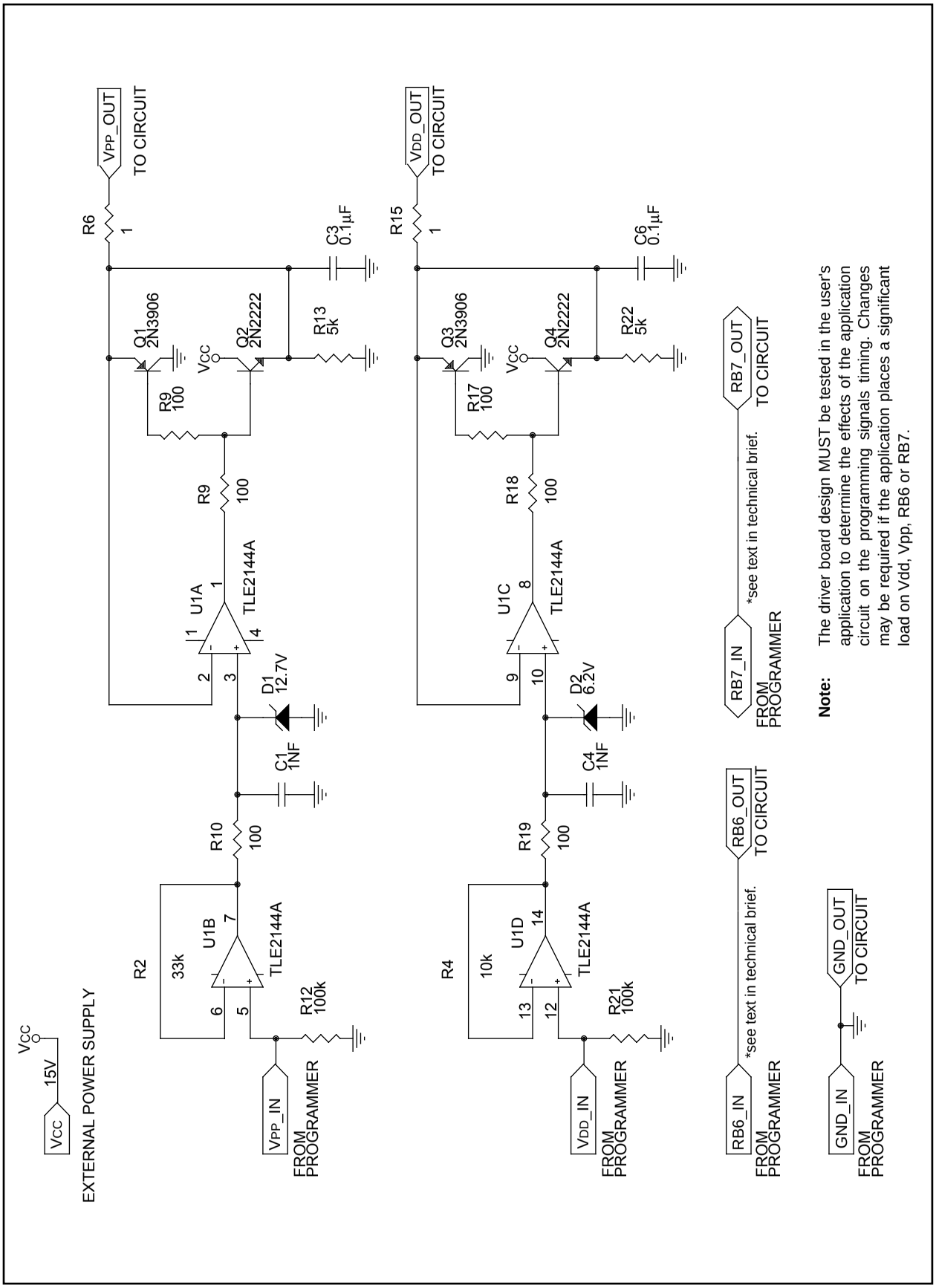
```
movlw <page>            movlw <page>
movwf PCLATH            movwf PCLATH
goto  Main              goto  ISR
```

Now your one time programmable PICmicro 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



TB015

How to Implement ICSP™ Using PIC17CXXX OTP MCUs

Author: *Stan D'Souza*
Microchip Technology Inc.

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.

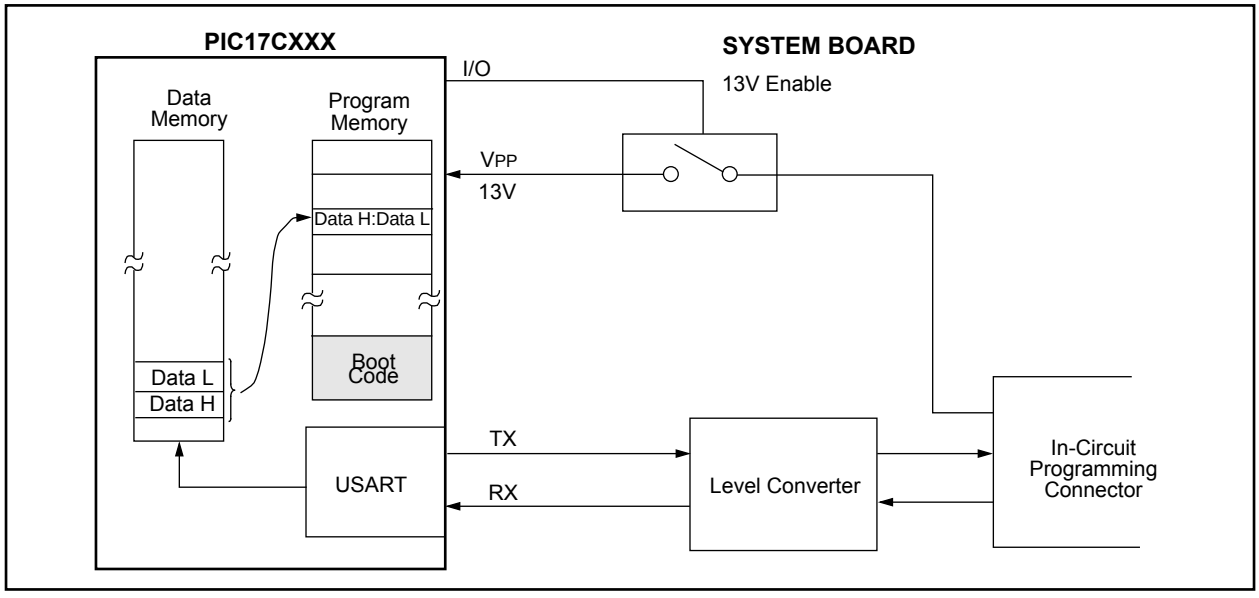
This technical brief will demonstrate the practical aspects associated with ICSP using the PIC17CXXX. It will also demonstrate some key capabilities of OTP devices when used in conjunction with ICSP.

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).

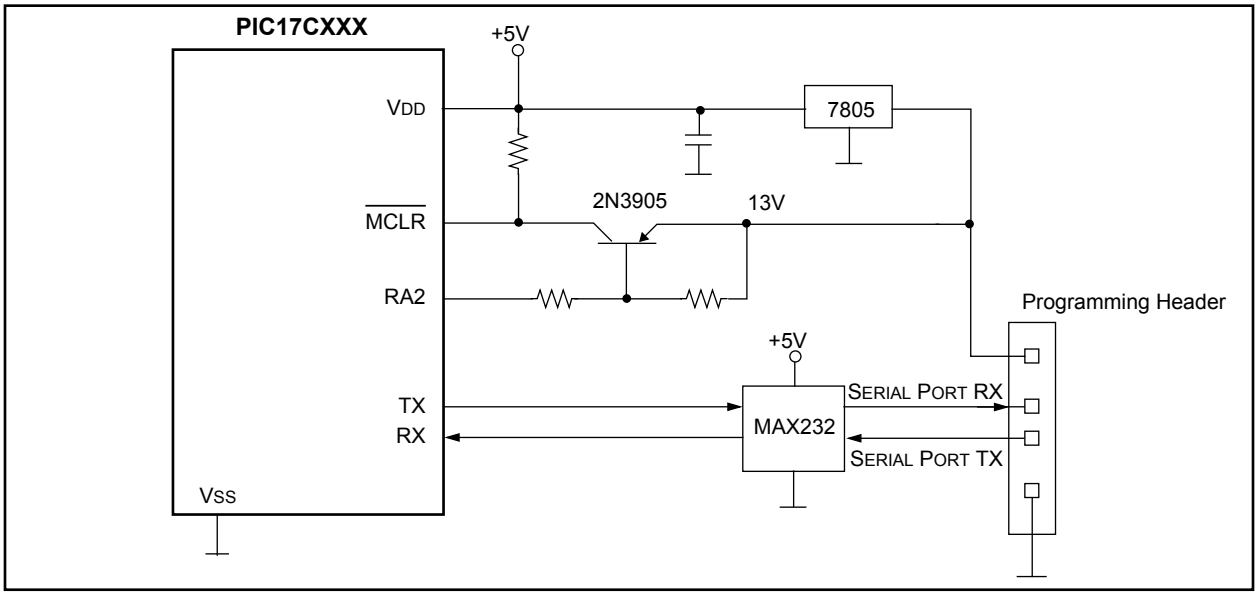
When doing ICSP, the PIC17CXXX runs a boot code, which configures the USART port and receives data serially through the RX line. This data is then programmed at the address specified in the serial data string. A high voltage (about 13V) is required for the EPROM cell to get programmed, and this is usually 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.

FIGURE 2: PIC17CXXX IN-CIRCUIT SERIAL PROGRAMMING SCHEMATIC



ICSP Boot Code

The boot code is normally programmed, into the PIC17CXXX 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 $\overline{MCLR}$. 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 3: BOOT CODE EXAMPLE FOR PIC17C42A

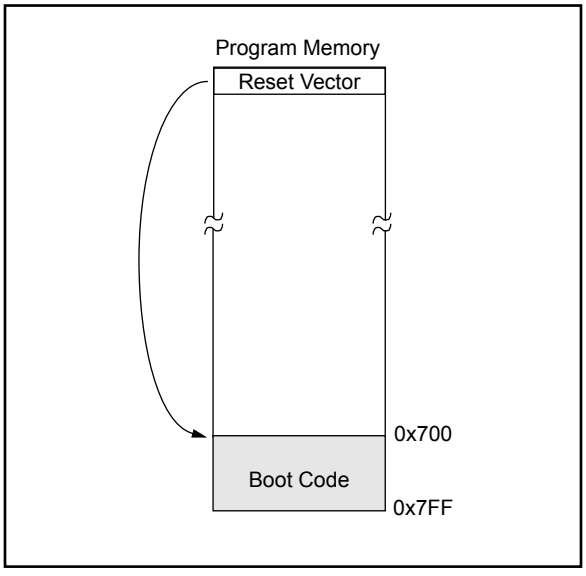
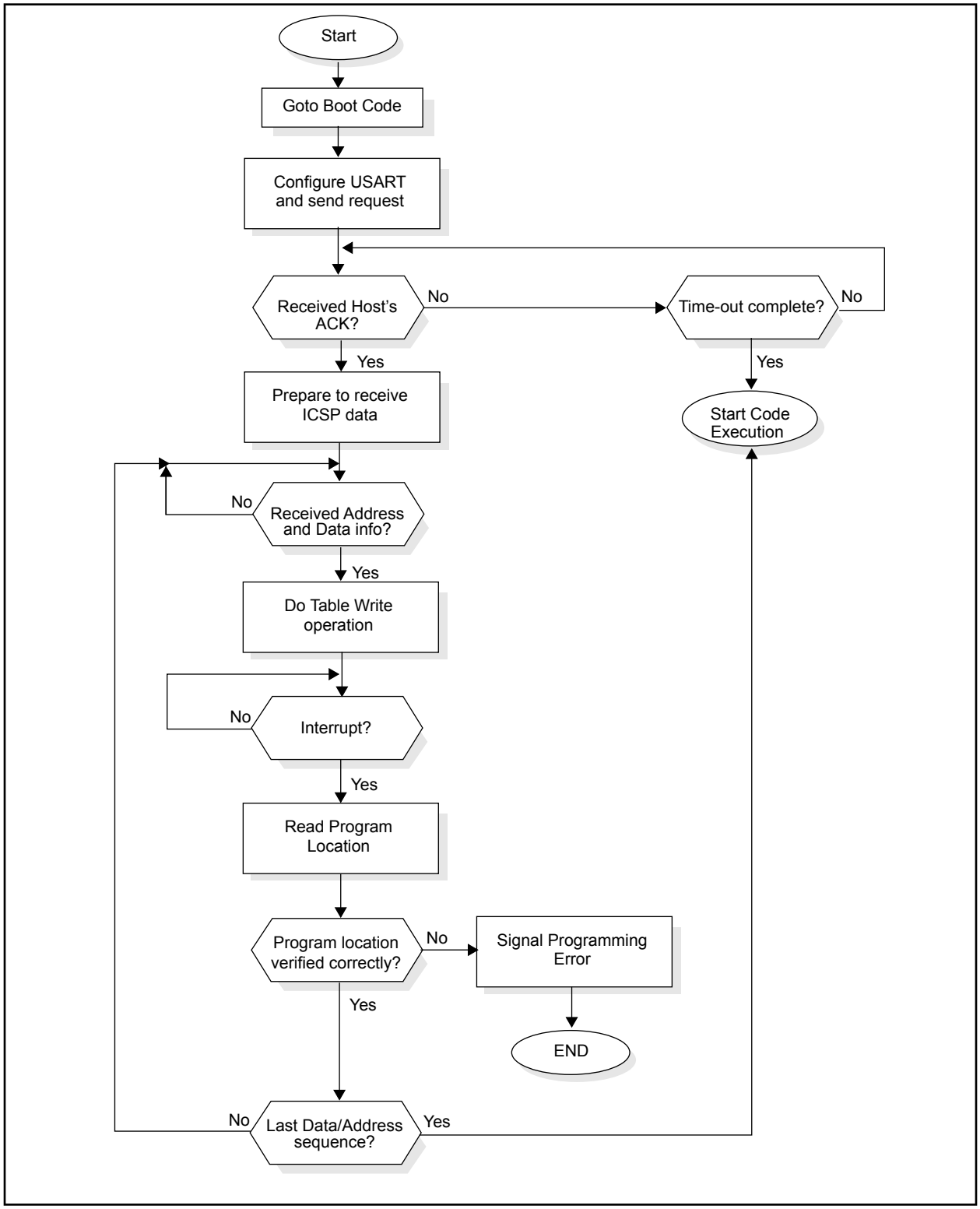


FIGURE 4: FLOWCHART FOR ICSP BOOT CODE



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:

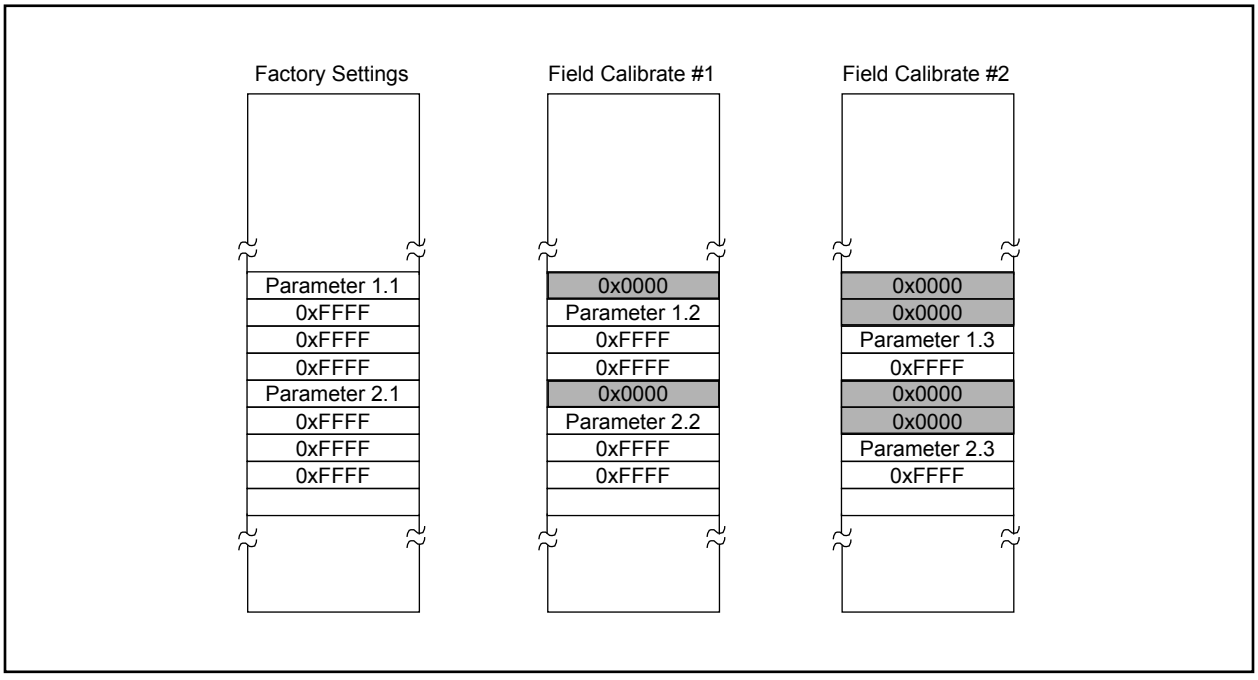
1. Un-programmed or blank locations have to be reserved at each calibration constant location in order to save new calibration parameters/constants.
2. The old calibration parameters/constants are all programmed to 0, so the user program will have to be "intelligent" and differentiate between blank (0xFFFF),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



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.

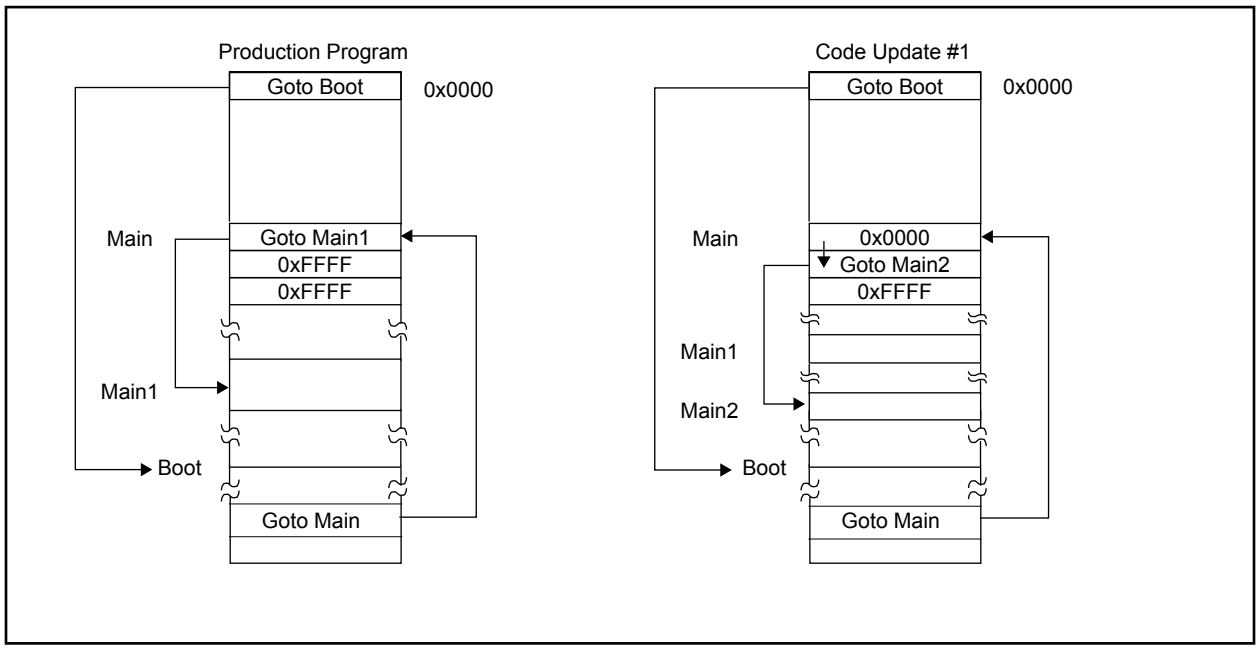
Here are some of the issues which need to be addressed:

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



NOTES:

TB016

How to Implement ICSP™ Using PIC16F8X FLASH MCUs

Author: *Rodger Richey*
Microchip Technology Inc.

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 turnarounds 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.

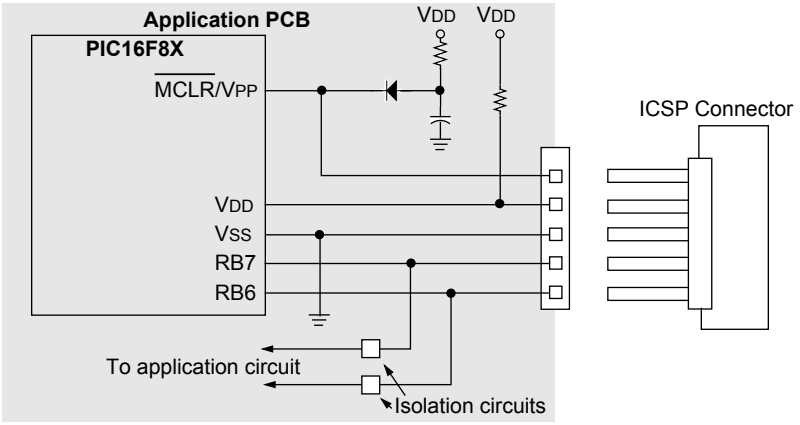
Application Circuit

The application circuit must be designed to allow all the programming signals to be directly connected to the PICmicro. 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 $\overline{MCLR}/V_{PP}$ 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 V_{DD}, $\overline{MCLR}/V_{PP}$, RB6, and RB7 pins.
4. Minimum and maximum operating voltage for V_{DD}.
5. PICmicro Oscillator.
6. Interface to the programmer.

The $\overline{MCLR}/V_{PP}$ pin is normally connected to an RC circuit. The pull-up resistor is tied to V_{DD} 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 $\overline{MCLR}/V_{PP}$. The diode should be a Schottky-type device. Another issue with $\overline{MCLR}/V_{PP}$ is that when the PICmicro 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 for serial programming. RB6 is the clock line and RB7 is the data line. RB6 is driven by the programmer. RB7 is a bi-directional pin that is driven by the programmer when programming, and driven by the PICmicro 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 and for RB7 being bi-directional (can be driven by both the PICmicro 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 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 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 option over the voltage range of the system.

This final issue deals with the oscillator circuit on the application board. The voltage on $\overline{MCLR}$/VPP must rise to the specified program mode entry voltage before the device executes any code. The crystal modes available on the PICmicro 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 $\overline{MCLR}$/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 $\overline{MCLR}$/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 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. 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 micrcontroller 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 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 in programming mode and programs the PICmicro 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.

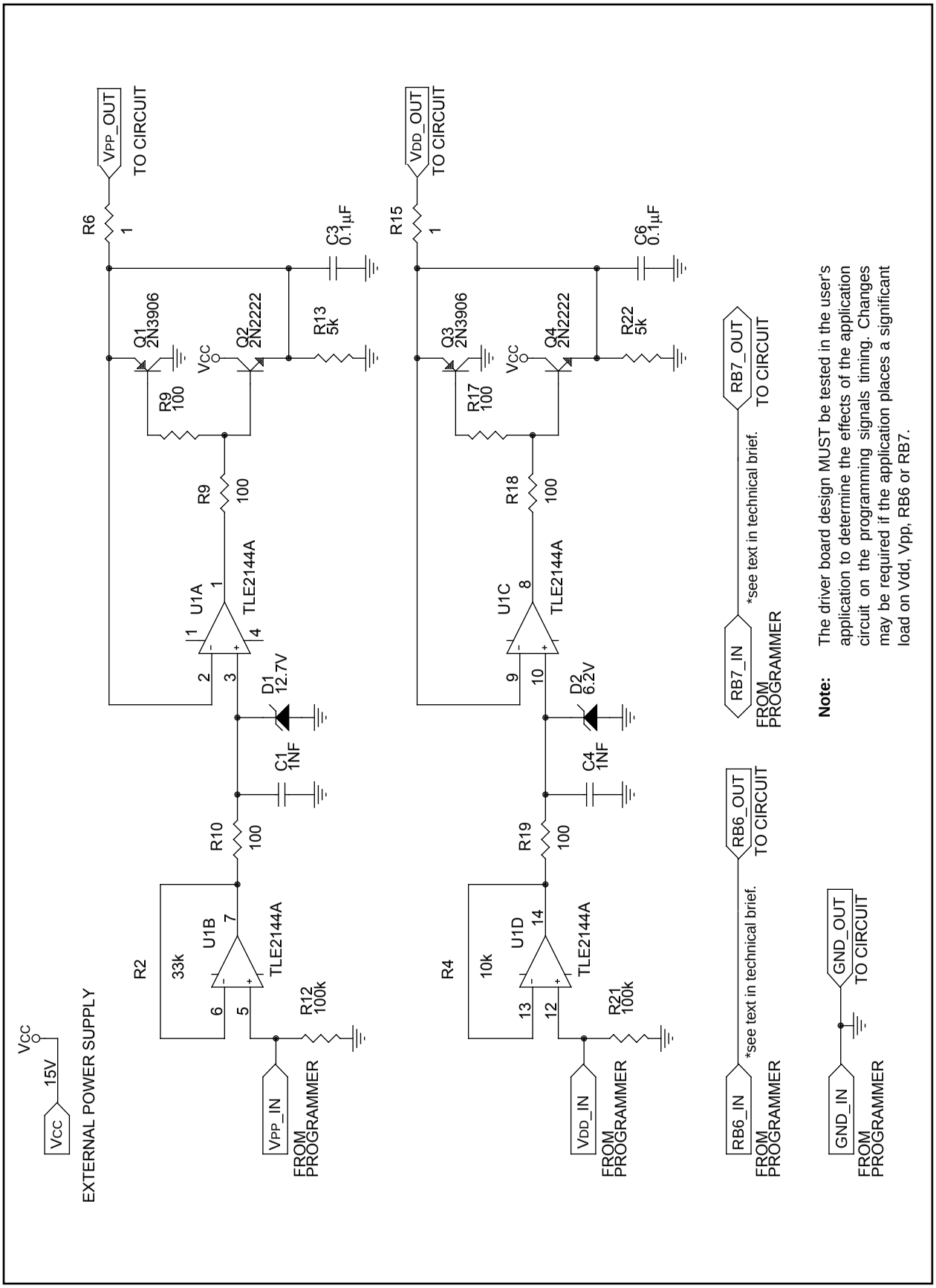
Field Programming of FLASH PICmicros

With the ISP interface circuitry already in place, these FLASH-based PICmicros 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. 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



SECTION 3
PICMICRO™ MICROCONTROLLER PROGRAMMING SPECIFICATIONS

PIC12C5XX

In-Circuit Serial Programming for PIC12C5XX OTP Microcontrollers

This document includes programming specifications for the following devices:

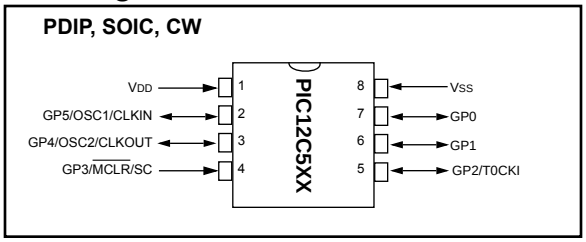
- PIC12C508
- PIC12C509

PROGRAMMING THE PIC12C5XX

The PIC12C5XX can be programmed using a serial method.

Pin Diagram

PDIP, SOIC, CW



PIN DESCRIPTIONS (DURING PROGRAMMING): PIC12C508/509

Pin Name	During Programming		
	Pin Name	Pin Type	Pin Description
GP1	CLOCK	I	Clock input
GP0	DATA	I/O	Data input/output
GP3/$\overline{MCLR}$/V_{PP}	V_{PP}	P	Programming Power
V_{DD}	V_{DD}	P	Power Supply
V_{SS}	V_{SS}	P	Ground

Legend: I = input, O = Output, P = Power

PROGRAM MODE ENTRY

The program/verify test mode is entered by holding pins DB0 and DB1 low while raising $\overline{MCLR}$ pin from V_{IL} to V_{IHH}. 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 configuration word. **GP0 and GP1 are Schmitt trigger inputs in this mode.**

Incrementing the PC once (using the increment address command) selects location 0x000 of the user program memory. Afterwards all other memory locations from 0x001-01FF (PIC12C508), 0x001-03FF (PIC12C509) 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 to determine where the special EPROM area is located for the various PIC12C5XX devices).

Programming Method

The programming technique is described in the following section.

Programming Method Details

Essentially, this technique includes the following steps:

1. Perform blank check at $V_{DD} = V_{DD}$min. Report failure. The device may not be properly erased.
2. Program location with pulses and verify after each pulse at $V_{DD} = V_{DDP}$:
   where $V_{DDP} = V_{DD}$ range required during programming (4.5V - 5.5V).

   a)Programming condition:

   V_{PP} = 13.0V to 13.25V

   $V_{DD} = V_{DDP}$ = 4.5V to 5.5V

   b)Verify condition:

   $V_{DD} = V_{DDP}$

   $V_{PP} \geq V_{DD}$ + 7.5V but not to exceed 13.25V

   If location fails to program after 25 pulses, then report error as a programming failure.

Note:	Device must be verified at minimum and maximum specified operating voltages as specified in the data sheet.

3. After the location has been programmed, apply 3X overprogramming, i.e., apply three times the number of pulses that were required to program the location. This will guarantee a solid programming margin.
4. Program all locations.
5. Verify all locations at $V_{DD} = V_{DD}$min.
6. Verify all locations at $V_{DD} = V_{DD}$max.

Programming Pulse Width

Program Memory Cells: When programming one word of EPROM, a programming pulse width (T_{PW}) of 100 μs is recommended.

The maximum number of programming attempts should be limited to 25 per word.

After the first successful verify, the same location should be over-programmed with 3X over-programming.

Configuration Word: The configuration word for oscillator selection, $\overline{WDT}$ (watchdog timer) disable, code protection, and $\overline{MCLR}$ enable, requires a programming pulse width (T_{PWF}) of 10 ms. A series of 100 μs pulses is preferred over a single 10 ms pulse.

FIGURE 1: PROGRAMMING METHOD FLOWCHART

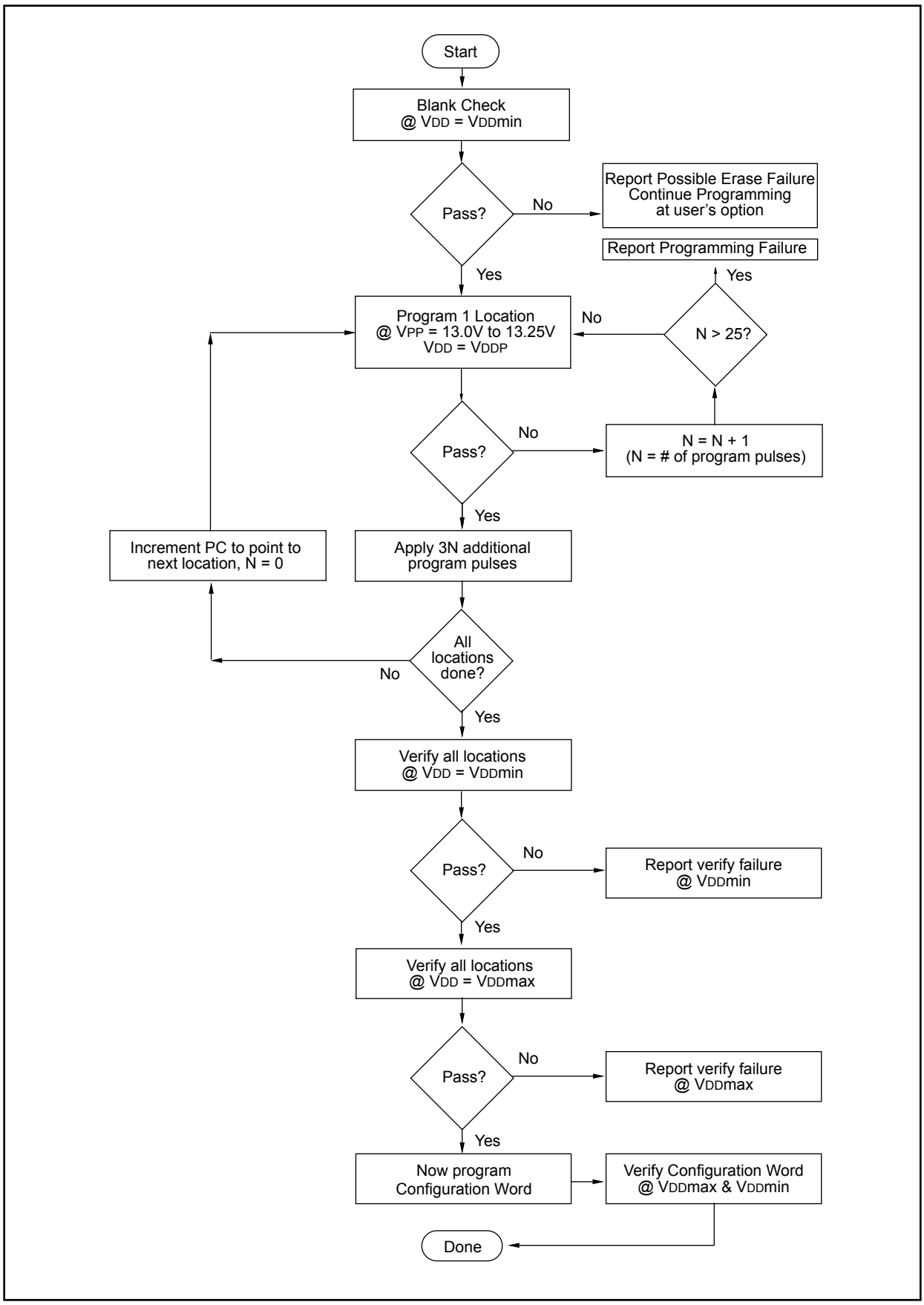
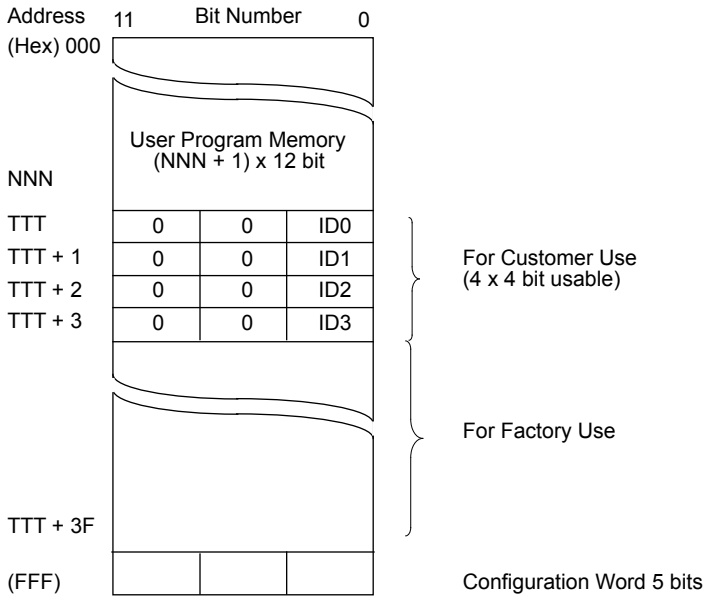


FIGURE 2: PIC12C5XX SERIES PROGRAM MEMORY MAP IN PROGRAM/VERIFY MODE



NNN Highest normal EPROM memory address. NNN = 0x1FF for PIC12C508.
NNN = 0x3FF for PIC12C509.
Note that some versions will have an oscillator calibration value programmed at NNN
TTT Start address of special EPROM area and ID Locations.

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 `MOVLW XX` instruction.

The ID Locations area is only enabled if the device is in a test or programming/verify mode. Thus, in normal operation mode only the memory location 0x000 to 0xNNN will be accessed and the Program Counter will just roll over from address 0xNNN to 0x000 when incremented.

The configuration word can only be accessed immediately after $\overline{MCLR}$ going from V_{IL} to V_{IHH}. The Program Counter will be set to all '1's upon $\overline{MCLR} = V_{IL}$. Thus, it has the value “0xFFF” when accessing the configuration EPROM. Incrementing the Program Counter once causes the Program Counter to roll over to all '0's. Incrementing the Program Counter 4K times after reset ($\overline{MCLR} = V_{IL}$) does not allow access to the configuration EPROM.

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.

Program/Verify Mode

The program/verify mode is entered by holding pins GP1 and GP0 low while raising $\overline{MCLR}$ pin from V_{IL} to V_{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. GP1 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 $\overline{MCLR}$ pin was initially at V_{IL}). This means that all I/O are in the reset state (High impedance inputs).

Note:	The $\overline{MCLR}$ pin should be raised from V_{IL} to V_{IHH} within 9 ms of V_{DD} rise. This is to ensure that the device does not have the PC incremented while in valid operation range.

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 µs 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 µs 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 µs is required between a command and a data word (or another command).

The commands that are available are listed in Table 1.

TABLE 1: COMMAND MAPPING

Command	Mapping (MSb ... LSb)						Data
Load Data	0	0	0	0	1	0	0, data(14), 0
Read Data	0	0	0	1	0	0	0, data(14), 0
Increment Address	0	0	0	1	1	0	
Begin programming	0	0	1	0	0	0	
End Programming	0	0	1	1	1	0	

Note: The clock must be disabled during In-Circuit Serial Programming.

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 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 4.

Increment Address

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

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.

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.

CODE PROTECTION

The program code written into the EPROM can be protected by writing to the CP0 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.

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 2: CODE PROTECTION

PIC12C508

To code protect:

- (CP enable pattern: XXXXXXXX0XXX)

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0xFFF)	Read enabled, Write Enabled	Read enabled, Write Enabled
[0x00:0x3F]	Read enabled, Write Enabled	Read enabled, Write Enabled
[0x40:0x1FF]	Read disabled (all 0’s), Write Disabled	Read enabled, Write Enabled
ID Locations (0x200 : 0x203)	Read enabled, Write Enabled	Read enabled, Write Enabled

PIC12C509

To code protect:

- (CP enable pattern: XXXXXXXX0XXX))

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0xFFF)	Read enabled, Write Enabled	Read enabled, Write Enabled
[0x00:0x3F]	Read enabled, Write Enabled	Read enabled, Write Enabled
[0x40:0x3FF]	Read disabled (all 0’s), Write Disabled	Read enabled, Write Enabled
ID Locations (0x400 : 0x403)	Read enabled, Write Enabled	Read enabled, Write Enabled

Checksum

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. 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 3.

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: CHECKSUM COMPUTATION

Device	Code Protect	Checksum*	Blank Value	0x723 at 0 and max address
PIC12C508	OFF	SUM[0x000:0x1FE] + CFGW & 0x001F	EE20	DC68
	ON	SUM[0x000:0x03F] + CFGW & 0x001F	EDF7	D363
PIC12C509	OFF	SUM[0x000:0x3FE] + CFGW & 0x001F	EC20	DA68
	ON	SUM[0x000:0x03F] + CFGW & 0x001F	EBF7	D163

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

PROGRAM/VERIFY MODE ELECTRICAL CHARACTERISTICS

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

Standard Operating Conditions
Operating Temperature: +10°C ≤ TA ≤ +40°C, unless otherwise stated, (20°C recommended)
Operating Voltage: 4.5V ≤ VDD 5.5V, unless otherwise stated.

Parameter No.	Sym.	Characteristic	Min.	Typ.	Max.	Units	Conditions
General							
PD1	VDDP	Supply voltage during programming	4.75	5.0	5.25	V	
PD2	IDDP	Supply current (from VDD) during programming	–	–	20	mA	
PD3	VDDV	Supply voltage during verify	VDDmin	–	VDDmax	V	Note 1
PD4	VIHH1	Voltage on $\overline{MCLR}$/VPP during programming	12.75	–	13.25	V	Note 2
PD5	VIHH2	Voltage on $\overline{MCLR}$/VPP during verify	VDD + 4.0	–	13.5	–	
PD6	IPP	Programming supply current (from VPP)	–	–	50	mA	
PD9	VIH1	(GP1, GP0) input high level	0.8 VDD	–	–	V	Schmitt Trigger input
PD8	VIL1	(GP1, GP0) input low level	0.2 VDD	–	–	V	Schmitt Trigger input
Serial Program Verify							
P1	TR	$\overline{MCLR}$/VPP rise time (VSS to VIHH) for test mode entry	–	–	8.0	ms	
P2	TF	$\overline{MCLR}$ Fall time	–	–	8.0	ms	
P3	TSET1	Data in setup time before clock ↓	100	–	–	ns	
P4	THLD1	Data in hold time after clock ↓	100	–	–	ns	
P5	TDLY1	Data input not driven to next clock input (delay required between command/data or command/command)	1.0	–	–	ms	
P6	TDLY2	Delay between clock ↓ to clock ↑ of next command or data	1.0	–	–	ms	
P7	TDLY3	Clock ↑ to date out valid (during read data)	200	–	–	ns	
P8	THLD0	Hold time after $\overline{MCLR}$ ↑	2	–	–	ms	

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 3: LOAD DATA COMMAND (PROGRAM/VERIFY)

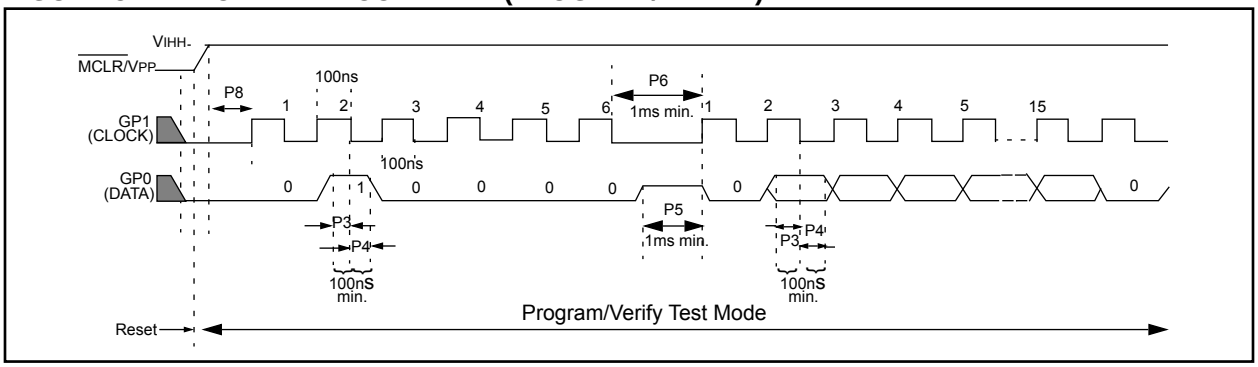


FIGURE 4: READ DATA COMMAND (PROGRAM/VERIFY)

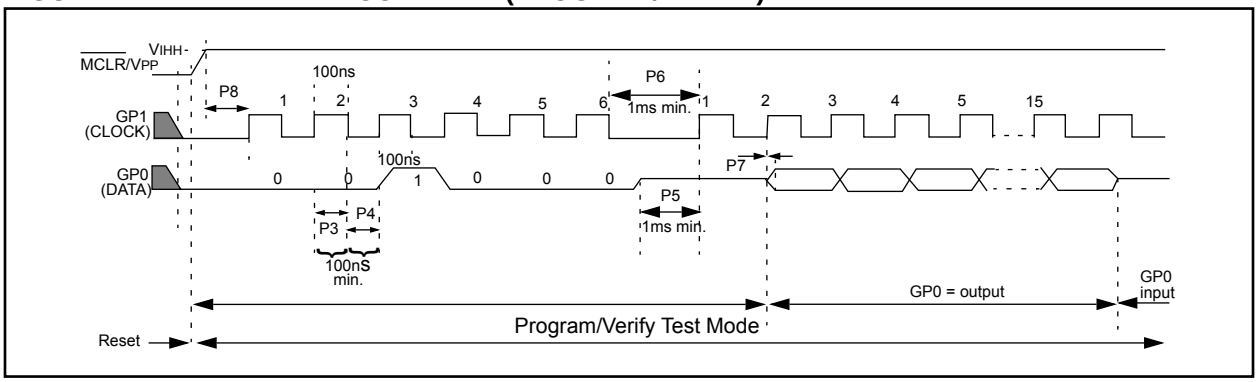
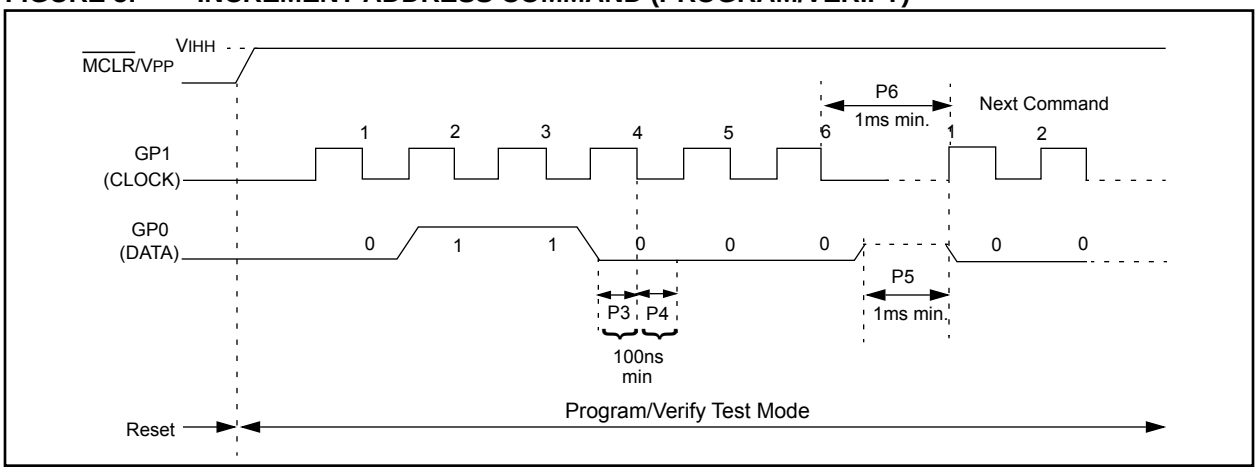


FIGURE 5: INCREMENT ADDRESS COMMAND (PROGRAM/VERIFY)



PIC14000

In-Circuit Serial Programming for PIC14XXX OTP Microcontrollers

This document includes the programming specifications for the following devices:

- PIC14000

PROGRAMMING THE PIC14000

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

Hardware Requirements

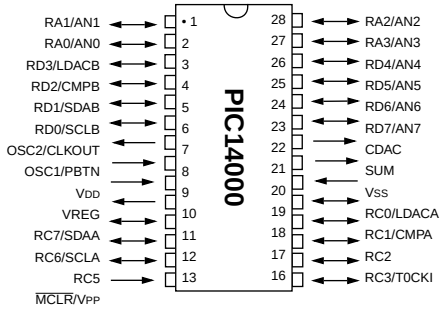
The PIC14000 requires two programmable power supplies, one for VDD (2.0V to 6.5V recommended) and one for VPP (12V to 14V).

Programming Mode

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

Pin Diagram

PDIP, SOIC, SSOP, Windowed CERDIP



Note: Peripheral pinout functions are not shown (see data sheets for full pinout information).

PIN DESCRIPTIONS (DURING PROGRAMMING): PIC14000

Pin Name	During Programming		
	Pin Name	Pin Type	Pin Description
RC6	CLOCK	I	Clock input
RC7	DATA	I/O	Data input/output
MCLR/VPP	VPP	P	Programming Power
VDD	VDD	P	Power Supply
VSS	VSS	P	Ground

Legend: I = Input, O = Output, P = Power

PROGRAM MODE ENTRY

User Program Memory Map

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

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

Area	Memory Space	Access to Memory
Program	0x000-0xFBF	PC<12:0>
Calibration	0xFC0 -0xFFF	PC<12:0>

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

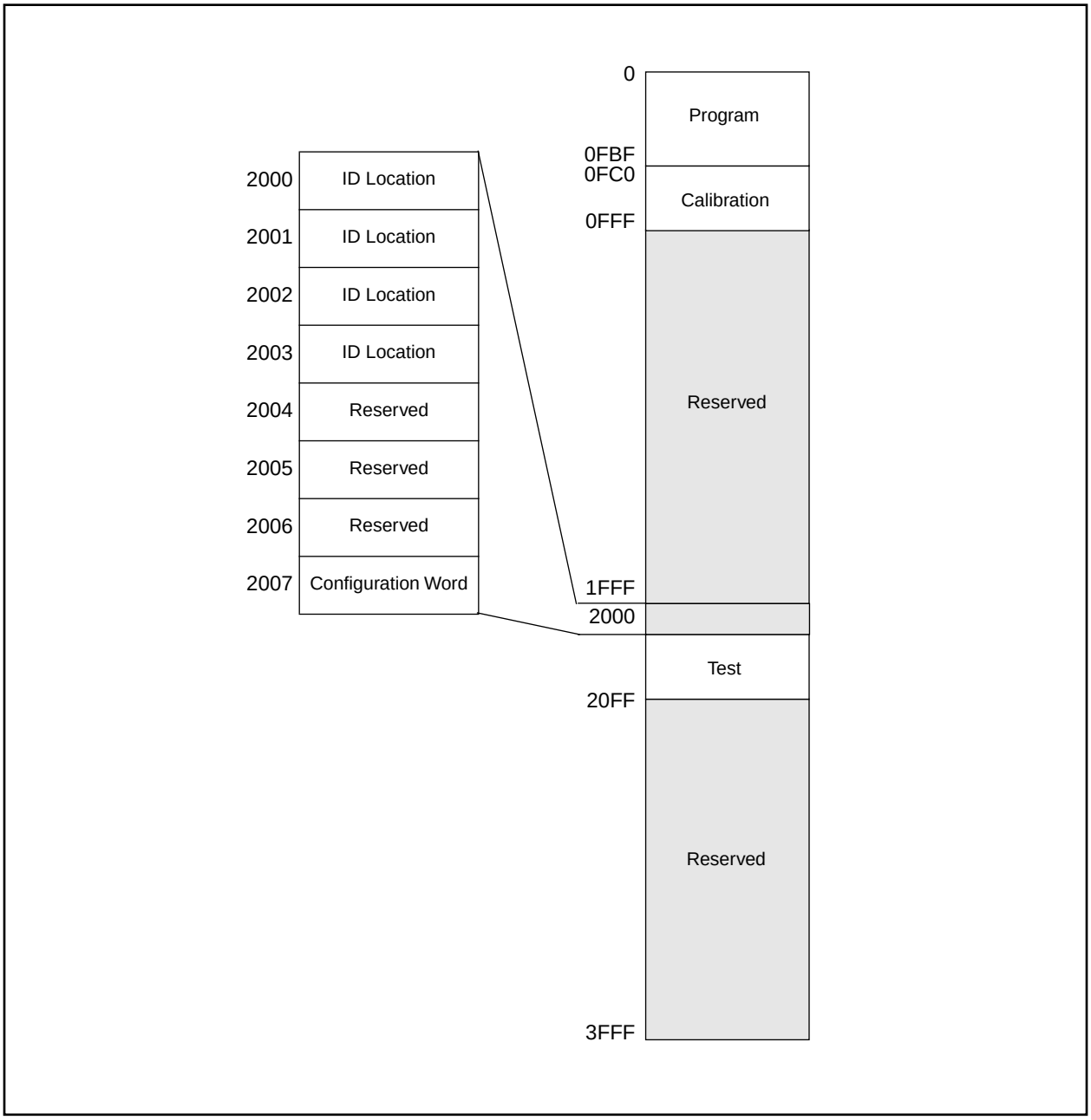
In programming mode the program memory space extends from 0x0000 to 0x3FFF, with the first half (0x0000-0x1FFF) being user program memory and the second half (0x2000-0x3FFF) being configuration memory. The PC will increment from 0x0000 to 0x1FFF and wrap to 0x0000, or 0x2000 to 0x3FFF and wrap around 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.

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 PC exceeds 0x2XFF (see Figure 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 3.

FIGURE 1: PROGRAM MEMORY MAPPING



Program/Verify Mode

The program/verify mode is entered by holding pins RC6 and RC7 low while raising $\overline{MCLR}$ pin from V_{IL} to V_{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. 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 $\overline{MCLR}$ pin was initially at V_{IL}). This means that all I/O are in the reset state (High impedance inputs).

Note:	The $\overline{MCLR}$ pin should be raised as quickly as possible from V_{IL} to V_{IHH}. This is to ensure that the device does not have the PC incremented while in valid operation range.

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 (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.

LOAD CONFIGURATION

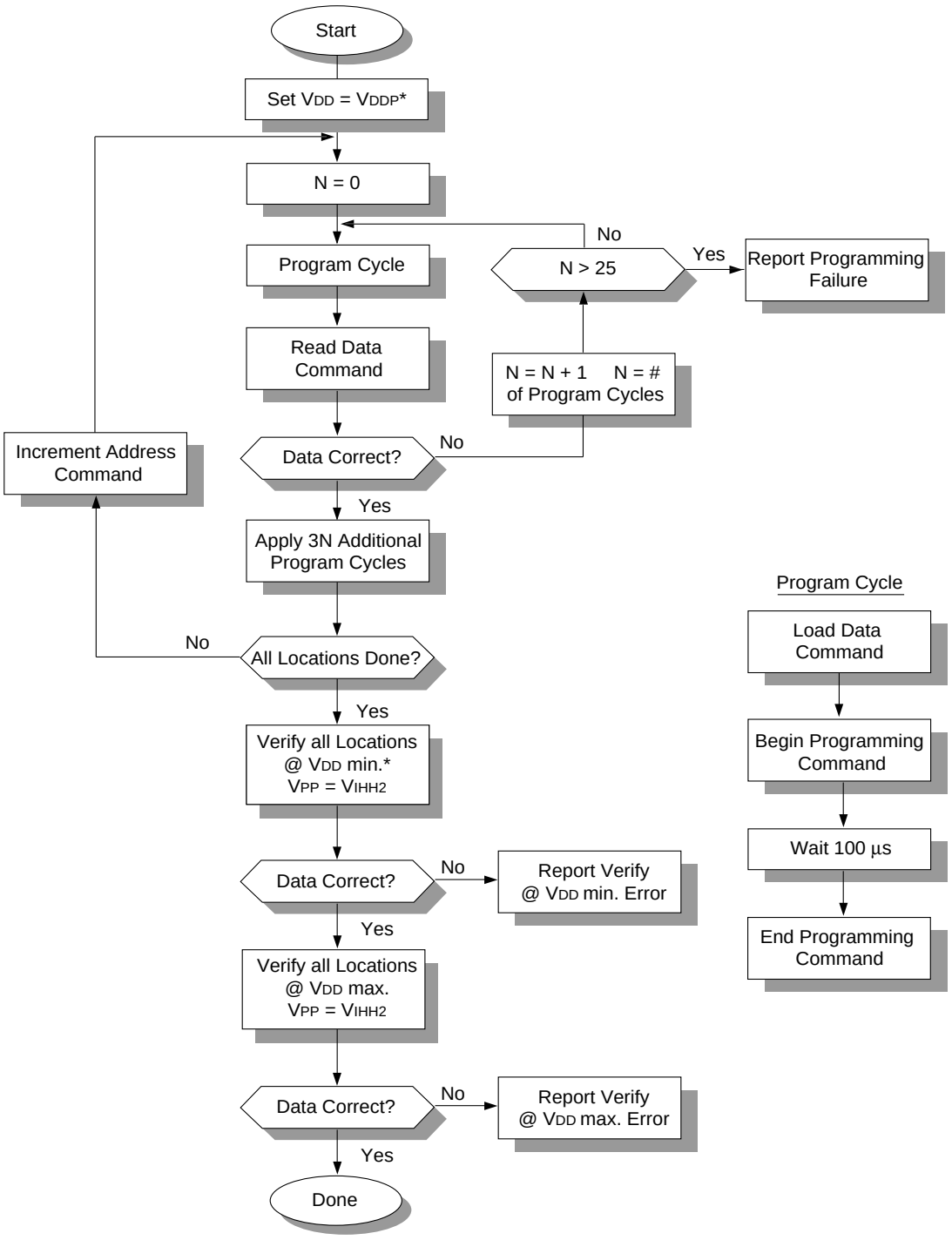
After receiving this command, the program counter (PC) will be set to 0x2000. By then applying 16 cycles to the clock pin, the chip will load 14-bits a “data word” as described above, to be programmed into the configuration memory. A description of the memory mapping schemes for normal operation and configuration mode operation is shown in Figure 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 $\overline{MCLR}$ low (V_{IL}).

TABLE 2: COMMAND MAPPING

Command	Mapping (MSB ... LSB)						Data
Load Configuration	0	0	0	0	0	0	0, data(14), 0
Load Data	0	0	0	0	1	0	0, data(14), 0
Read Data	0	0	0	1	0	0	0, data(14), 0
Increment Address	0	0	0	1	1	0	
Begin programming	0	0	1	0	0	0	
End Programming	0	0	1	1	1	0	

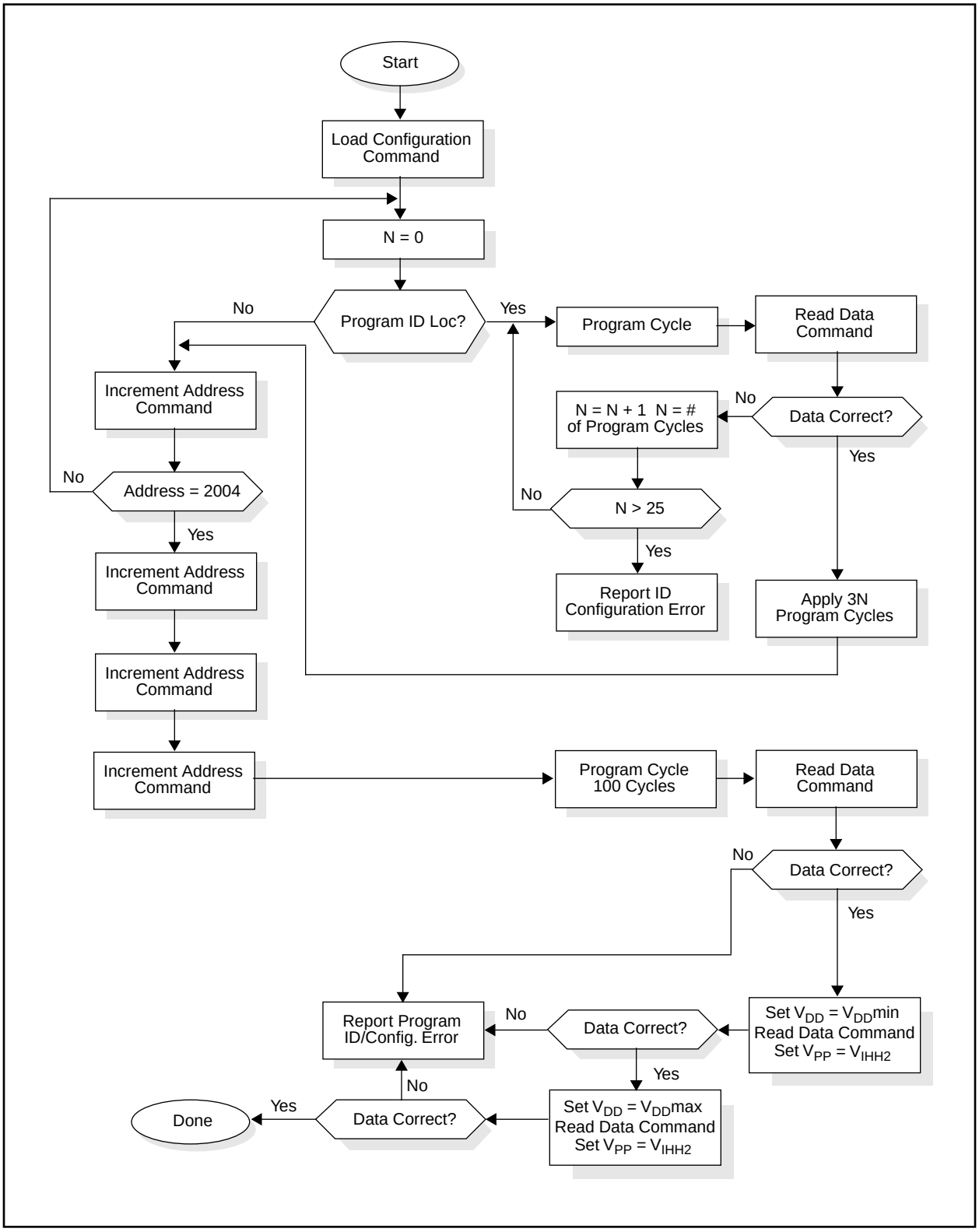
Note: The CPU clock must be disabled during in-circuit programming (to avoid incrementing the PC).

FIGURE 2: PROGRAM FLOW CHART - PIC14000 PROGRAM MEMORY AND CALIBRATION



* VDDP = VDD range for programming (typically 4.75V - 5.25V).
VDDmin = Minimum VDD for device operation.
VDDmax = Maximum VDD for device operation.

FIGURE 3: PROGRAM FLOW CHART - PIC14000 CONFIGURATION WORD & ID LOCATIONS



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.

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 6.

INCREMENT ADDRESS

The PC is incremented when this command is received. A timing diagram of this command is shown in Figure 7.

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.

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.

Programming Algorithm Requires Variable V_{DD}

The PIC14000 uses an intelligent algorithm. The algorithm calls for program verification at V_{DD}min as well as V_{DD}max. Verification at V_{DD}min guarantees good “erase margin”. Verification at V_{DD}max guarantees good “program margin”.

The actual programming must be done with V_{DD} in the V_{DDP} range (4.75 - 5.25V).

V_{DDP} = V_{CC} range required during programming.

V_{DD}min = minimum operating V_{DD} spec for the part.

V_{DD}max = maximum operating V_{DD} spec for the part.

Programmers must verify the PIC14000 at its specified V_{DD}max and V_{DD}min levels. Since Microchip may introduce future versions of the PIC14000 with a broader V_{DD} 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.

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 4 provides an overview of configuration bits.

FIGURE 4: CONFIGURATION WORD BIT MAP

Bit Number:	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PIC14000	CPC	CPP1	CPP0	CPP0	CPP1	CPC	CPC	F	CPP1	CPP0	$\overline{\text{PWRTE}}$	WDTE	F	FOSC

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: **$\overline{\text{PWRTE}}$**, 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 for cautions.

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.

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.

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).

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.

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 3: CODE PROTECT OPTIONS

- Protect calibration memory `0XXXX00XXXXXXX`
- Protect program memory `X0000XXX00XXXX`
- No code protection `1111111X11XXXX`

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

Legend: X = Don't care

Checksum

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, 0xFBF. 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:

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: CHECKSUM COMPUTATION

Code Protect	Checksum*	Blank Value	0x25E6 at 0 and max address
OFF	SUM[0000:0FBF] + CFGW & 0x3FBD	0x2FFD	0xFBCB
OFF OTP	SUM[0000:0FBF] + CFGW & 0x3FBD	0x0E7D	0xDA4B
ON	CFGW & 0x3FBD + SUM(IDs)	0x300A	0xFBD8

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

PROGRAM/VERIFY MODE ELECTRICAL CHARACTERISTICS

TABLE 5: AC/DC CHARACTERISTICS
AC/DC Timing Requirements for Program/Verify Mode

Standard Operating Conditions Operating Temperature: +10°C ≤ TA ≤ +40°C, unless otherwise stated, (25°C recommended) Operating Voltage: 4.5V ≤ VDD ≤ 5.5V, unless otherwise stated.							
Parameter No.	**Sym.**	**Characteristic**	**Min.**	**Typ.**	**Max.**	**Units**	**Conditions**
General							
PD1	VDDP	Supply voltage during programming	4.75	5.0	5.25	V	
PD2	IDDP	Supply current (from VDD) during programming	–	–	20	mA	
PD3	VDDV	Supply voltage during verify	VDDmin		VDDmax	V	Note 1
PD4	VIHH1	Voltage on $\overline{MCLR}$/VPP during programming	12.75	–	13.25	V	Note 2
PD5	VIHH2	Voltage on $\overline{MCLR}$/VPP during verify	VDD + 4.0		13.5		
PD6	IPP	Programming supply current (from VPP)	–	–	50	mA	
PD9	VIH1	(RC6, RC7) input high level	0.8 VDD	–	–	V	Schmitt Trigger input
PD8	VIL1	(RC6, RC7) input low level	0.2 VDD	–	–	V	Schmitt Trigger input
Serial Program Verify							
P1	TR	$\overline{MCLR}$/VPP rise time (VSS to VHH) for test mode entry	–	–	8.0	µs	
P2	Tf	$\overline{MCLR}$ Fall time	–	–	8.0	µs	
P3	Tset1	Data in setup time before clock ↓	100	–	–	ns	
P4	Thld1	Data in hold time after clock ↓	100	–	–	ns	
P5	Tdly1	Data input not driven to next clock input (delay required between command/data or command/command)	1.0	–	–	µs	
P6	Tdly2	Delay between clock ↓ to clock ↑ of next command or data	1.0	–	–	µs	
P7	Tdly3	Clock ↑ to date out valid (during read data)	200	–	–	ns	
P8	Thld0	Hold time after $\overline{MCLR}$ ↑	2	–	–	µs	

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 5: LOAD DATA COMMAND (PROGRAM/VERIFY)

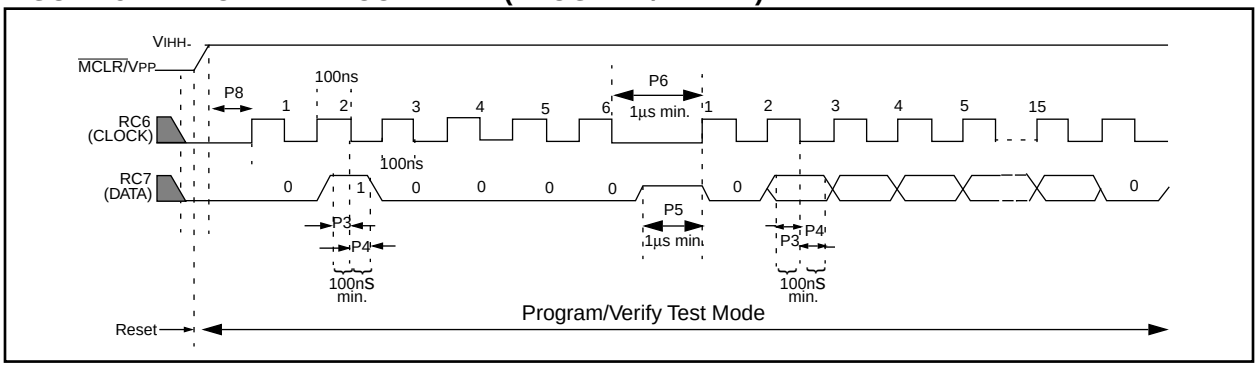


FIGURE 6: READ DATA COMMAND (PROGRAM/VERIFY)

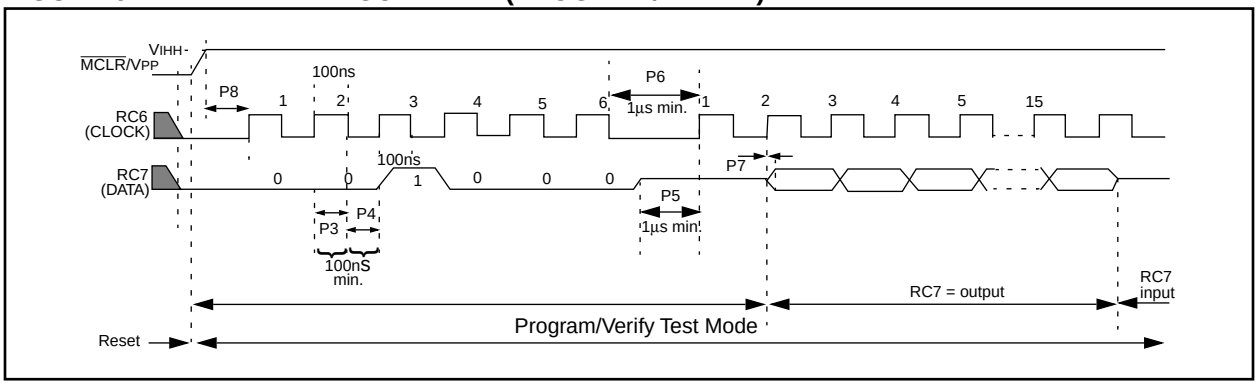
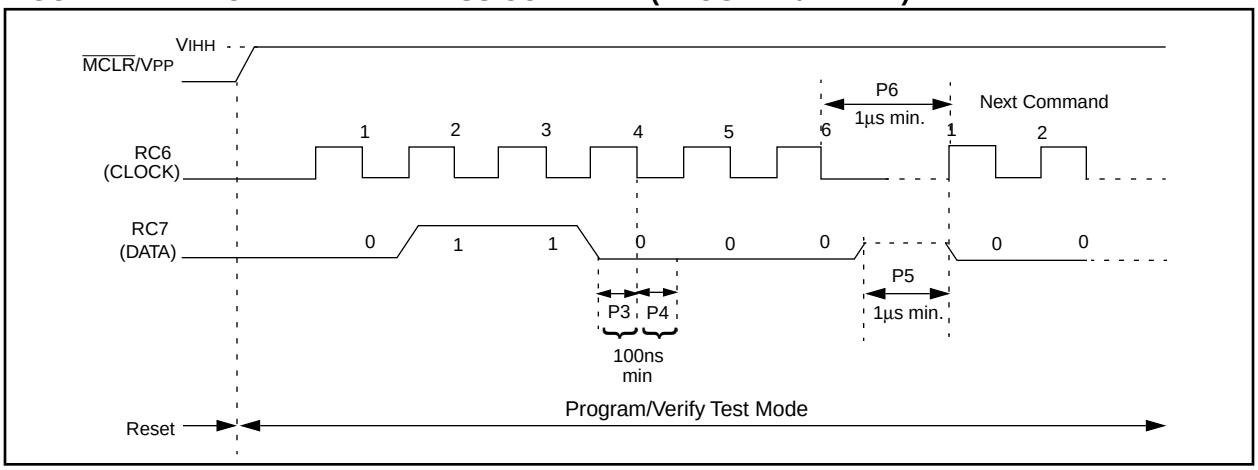


FIGURE 7: INCREMENT ADDRESS COMMAND (PROGRAM/VERIFY)



PIC16C55X

In-Circuit Serial Programming for PIC16C55X OTP Microcontrollers

This document includes the programming specifications for the following devices:

- PIC16C554
- PIC16C556
- PIC16C558

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.

Hardware Requirements

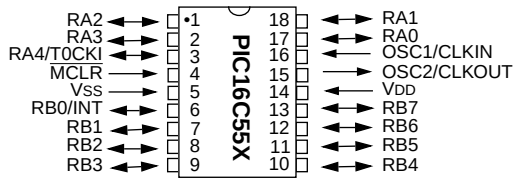
The PIC16C55X requires two programmable power supplies, one for VDD (2.0V to 6.5V recommended) and one for VPP (12V to 14V). Both supplies should have a minimum resolution of 0.25V.

Programming Mode

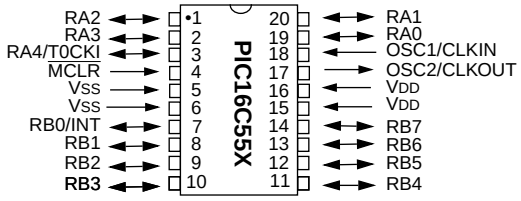
The programming mode for the PIC16C55X allows programming of user program memory, special locations used for ID, and the configuration word for the PIC16C55X.

Pin Diagrams

PDIP, SOIC, Windowed CERDIP



SSOP



Note: Peripheral pinout functions are not shown (see data sheets for full pinout information).

PIN DESCRIPTIONS (DURING PROGRAMMING): PIC16C554/556/558

Pin Name	During Programming		
	Pin Name	Pin Type	Pin Description
RB6	CLOCK	I	Clock input
RB7	DATA	I/O	Data input/output
MCLR/VPP	VPP	P	Programming Power
VDD	VDD	P	Power Supply
VSS	VSS	P	Ground

Legend: I = Input, O = Output, P = Power

PROGRAM MODE ENTRY

User Program Memory Map

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

TABLE 1: IMPLEMENTATION OF PROGRAM MEMORY IN THE PIC16C55X

Device	Program Memory Size	Access to Program Memory
PIC16C554	0x000 - 0x1FF (0.5K)	PC<8:0>
PIC16C556	0x000 - 0x3FF (1K)	PC<9:0>
PIC16C558	0x000 - 0x7FF (2K)	PC<10:0>

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 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.

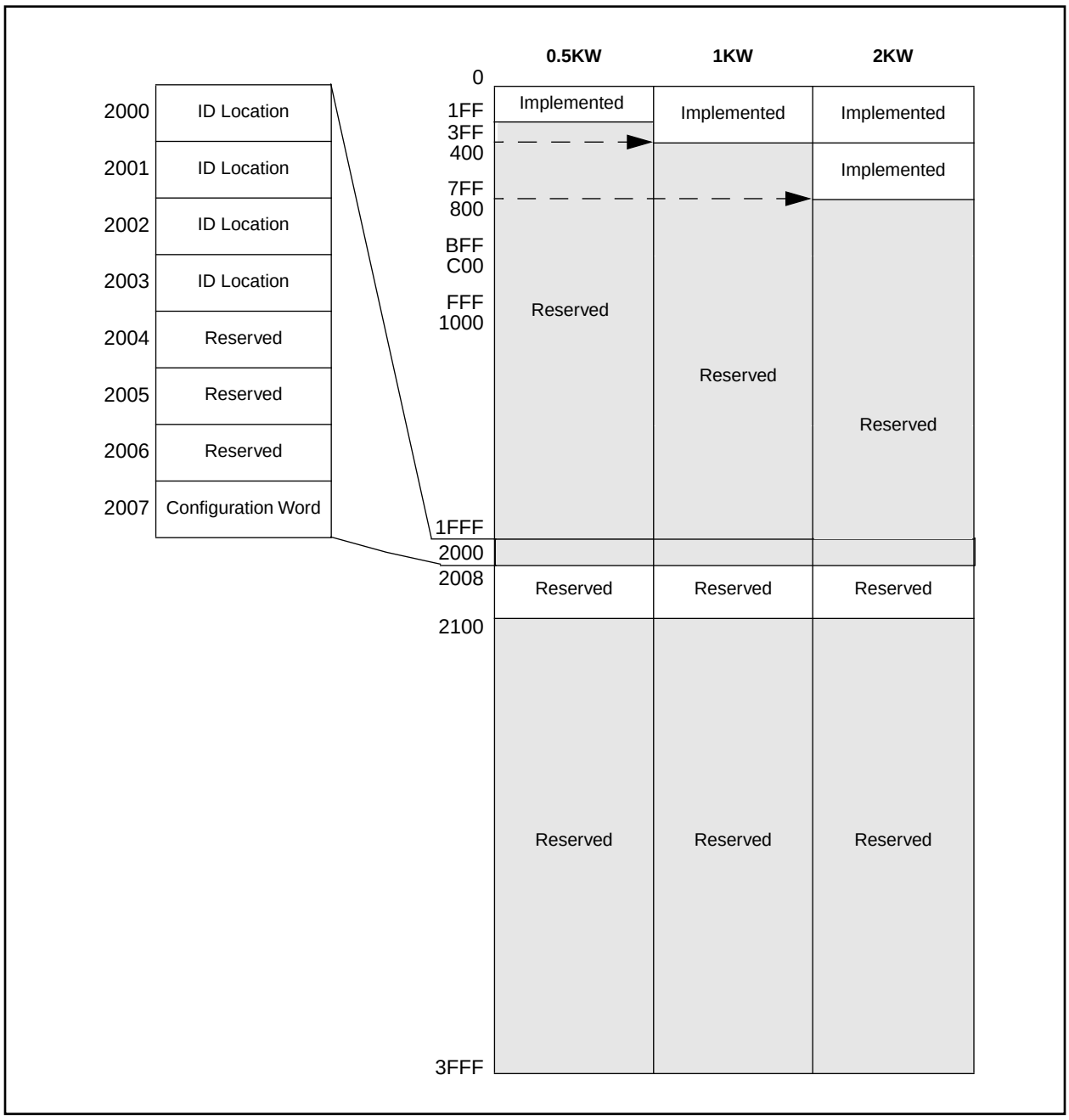
In the configuration memory space, 0x2000-0x20FF are utilized. When in a configuration memory, as in the user memory, the 0x2000-0x2XFF segment is repeatedly accessed as the PC exceeds 0x2XFF (see Figure 1).

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 3.

FIGURE 1: PROGRAM MEMORY MAPPING



Program/Verify Mode

The program/verify mode is entered by holding pins RB6 and RB7 low while raising $\overline{MCLR}$ pin from V_{IL} to V_{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 $\overline{MCLR}$ pin was initially at V_{IL}). This means that all I/O are in the reset state (High impedance inputs).

Note:	The $\overline{MCLR}$ pin should be raised as quickly as possible from V_{IL} to V_{IHH}. this is to ensure that the device does not have the PC incremented while in valid operation range.

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 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 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 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.

LOAD CONFIGURATION

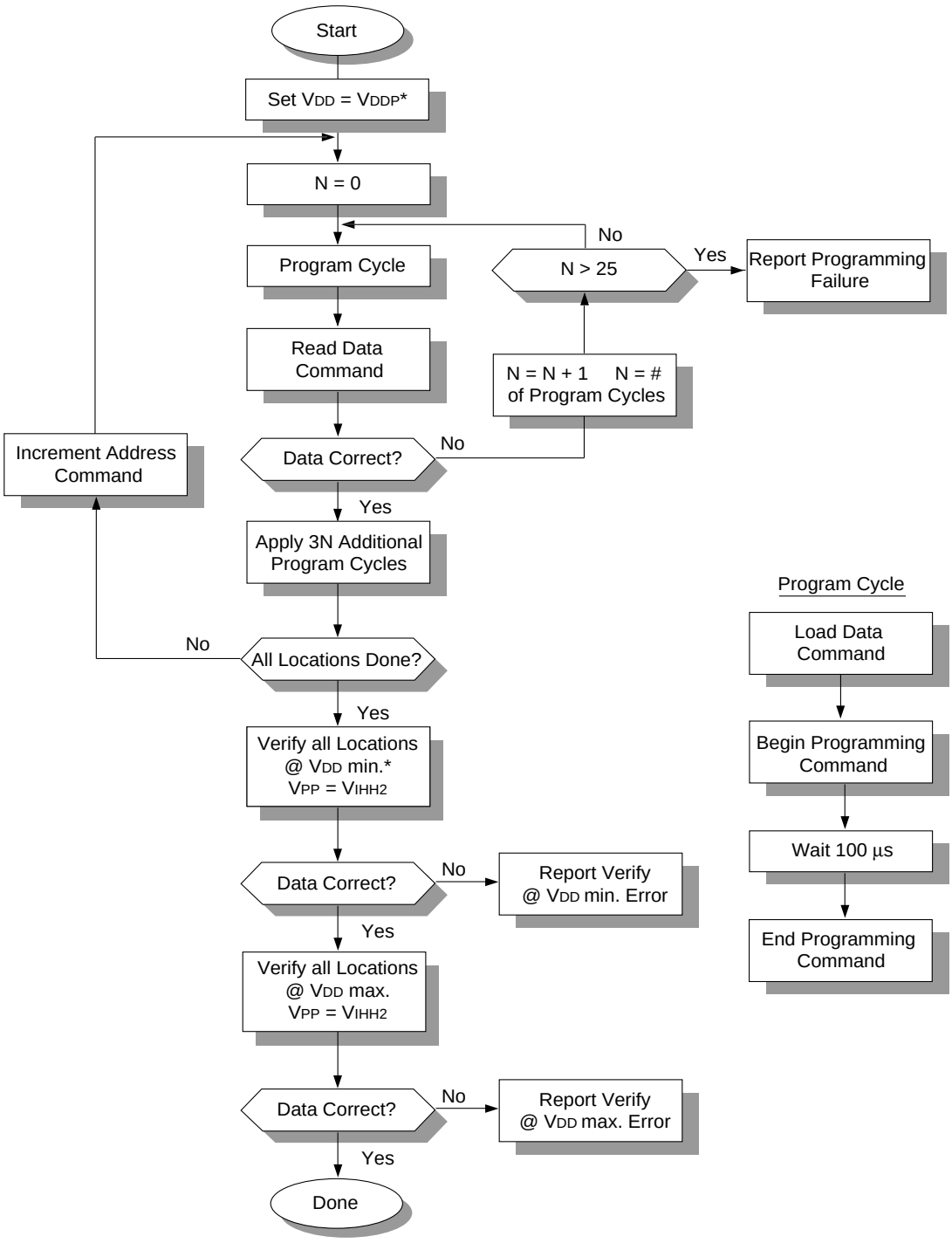
After receiving this command, the program counter (PC) will be set to 0x2000. By then applying 16 cycles to the clock pin, the chip will load 14-bits a “data word” as described above, to be programmed into the configuration memory. A description of the memory mapping schemes for normal operation and configuration mode operation is shown in Figure 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 $\overline{MCLR}$ low (V_{IL}).

TABLE 2: COMMAND MAPPING

Command	Mapping (MSB ... LSB)						Data
Load Configuration	0	0	0	0	0	0	0, data(14), 0
Load Data	0	0	0	0	1	0	0, data(14), 0
Read Data	0	0	0	1	0	0	0, data(14), 0
Increment Address	0	0	0	1	1	0	
Begin programming	0	0	1	0	0	0	
End Programming	0	0	1	1	1	0	

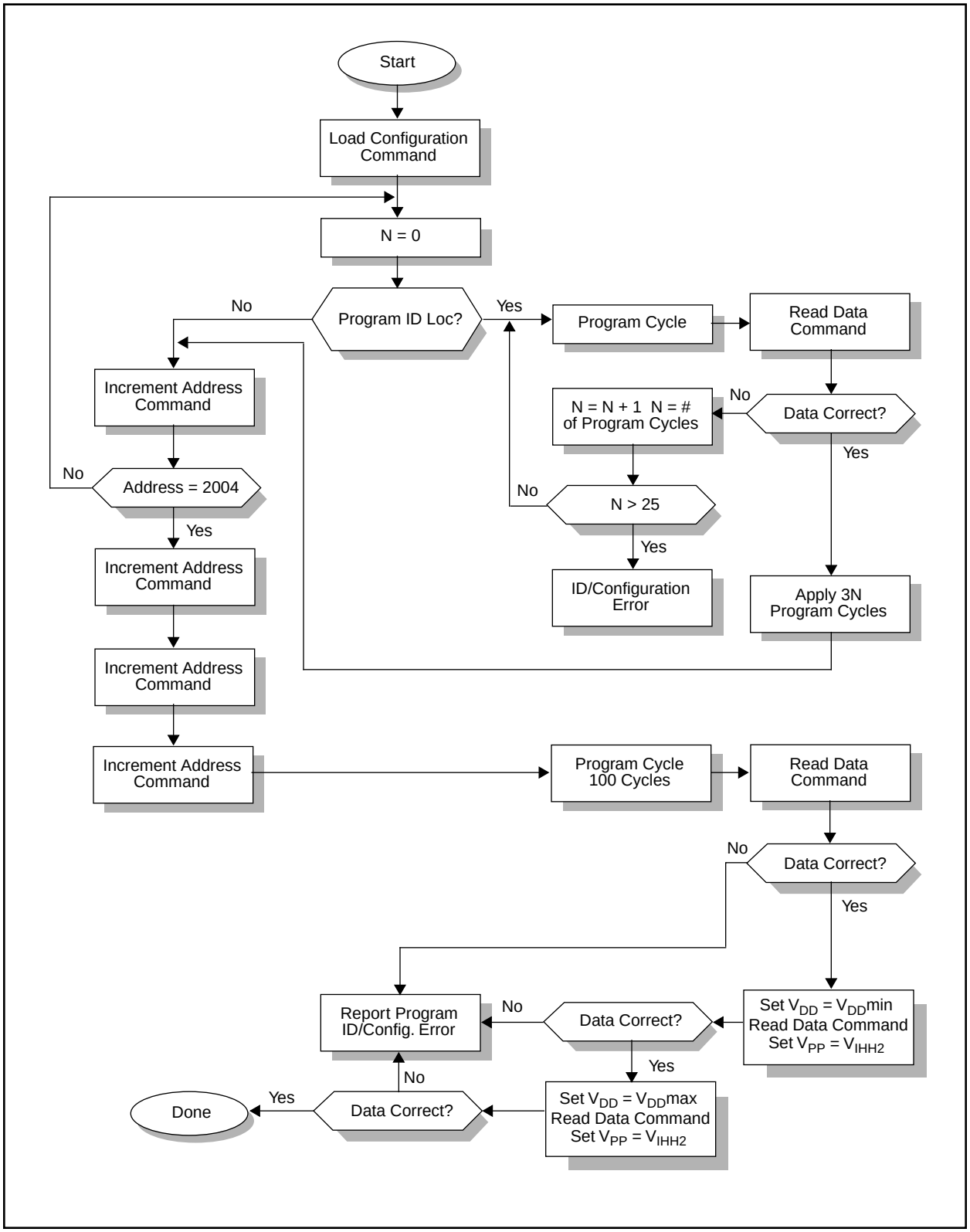
Note: The CPU clock must be disabled during in-circuit programming.

FIGURE 2: PROGRAM FLOW CHART - PIC16C55X PROGRAM MEMORY



* VDDP = VDD range for programming (typically 4.75V - 5.25V).
VDDmin = Minimum VDD for device operation.
VDDmax = Maximum VDD for device operation.

FIGURE 3: PROGRAM FLOW CHART - PIC16C55X CONFIGURATION WORD & ID LOCATIONS



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.

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 6.

INCREMENT ADDRESS

The PC is incremented when this command is received. A timing diagram of this command is shown in Figure 7.

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.

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.

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.

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 4 provides an overview of configuration bits.

FIGURE 4: CONFIGURATION WORD BIT MAP

Bit Number:	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PIC16C554/556/558	CP1	CP0	CP1	CP0	CP1	CP0	—	—	CP1	CP0	$\overline{\text{PWRTE}}$	WDTE	FOSC1	FOSC0

bit 6-7:**Reserved for future use**

bit 5-4: **CP1:CP0**, Code Protect

bit 8-13

Device	CP1	CP0	Code Protection
PIC16C554	0	0	All memory protected
	0	1	Do not use
	1	0	Do not use
	1	1	Code protection off
PIC16C556	0	0	All memory protected
	0	1	Upper 1/2 memory protected
	1	0	Do not use
	1	1	Code protection off
PIC16C558	0	0	All memory protected
	0	1	Upper 3/4 memory protected
	1	0	Upper 1/2 memory protected
	1	1	Code protection off

bit 3: **$\overline{\text{PWRTE}}$**, 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

CODE PROTECTION

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

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.

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 3: CONFIGURATION WORD

PIC16C554

To code protect:

- Protect all memory `0000001X00XXXX`
- No code protection `1111111X11XXXX`

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0x2007)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Unprotected memory segment	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Protected memory segment	Read All 0’s, Write Disabled	Read Unscrambled, Write Enabled
ID Locations (0x2000 : 0x2003)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled

PIC16C556

To code protect:

- Protect all memory `0000001X00XXXX`
- Protect upper 1/2 memory `0101011X01XXXX`
- No code protection `1111111X11XXXX`

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0x2007)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Unprotected memory segment	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Protected memory segment	Read All 0’s, Write Disabled	Read Unscrambled, Write Enabled
ID Locations (0x2000 : 0x2003)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled

PIC16C558

To code protect:

- Protect all memory `0000001X00XXXX`
- Protect upper 3/4 memory `0101011X01XXXX`
- Protect upper 1/2 memory `1010101X10XXXX`
- No code protection `1111111X11XXXX`

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0x2007)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Unprotected memory segment	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Protected memory segment	Read All 0’s, Write Disabled	Read Unscrambled, Write Enabled
ID Locations (0x2000 : 0x2003)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled

Checksum

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 4.

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: CHECKSUM COMPUTATION

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

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

PROGRAM/VERIFY MODE ELECTRICAL CHARACTERISTICS

TABLE 5: AC/DC CHARACTERISTICS
TIMING REQUIREMENTS FOR PROGRAM/VERIFY TEST MODE

Standard Operating Conditions Operating Temperature: +10°C ≤ TA ≤ +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**
General							
PD1	VDDP	Supply voltage during programming	4.75	5.0	5.25	V	
PD2	IDDP	Supply current (from VDD) during programming			20	mA	
PD3	VDDV	Supply voltage during verify	VDDmin		VDDmax	V	Note 1
PD4	VIHH1	Voltage on $\overline{MCLR}$/VPP during programming	12.75		13.25	V	Note 2
PD5	VIHH2	Voltage on $\overline{MCLR}$/VPP during verify	VDD + 4.0		13.5		
PD6	IPP	Programming supply current (from VPP)			50	mA	
PD9	VIH1	(RB6, RB7) input high level	0.8 VDD			V	Schmitt Trigger input
PD8	VIL1	(RB6, RB7) input low level	0.2 VDD			V	Schmitt Trigger input
Serial Program Verify							
P1	TR	$\overline{MCLR}$/VPP rise time (VSS to VHH) for test mode entry			8.0	µs	
P2	Tf	$\overline{MCLR}$ Fall time			8.0	µs	
P3	Tset1	Data in setup time before clock ↓	100			ns	
P4	Thld1	Data in hold time after clock ↓	100			ns	
P5	Tdly1	Data input not driven to next clock input (delay required between command/data or command/command)	1.0			µs	
P6	Tdly2	Delay between clock ↓ to clock ↑ of next command or data	1.0			µs	
P7	Tdly3	Clock ↑ to date out valid (during read data)	200			ns	
P8	Thld0	Hold time after $\overline{MCLR}$ ↑	2			µs	

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 5: LOAD DATA COMMAND (PROGRAM/VERIFY)

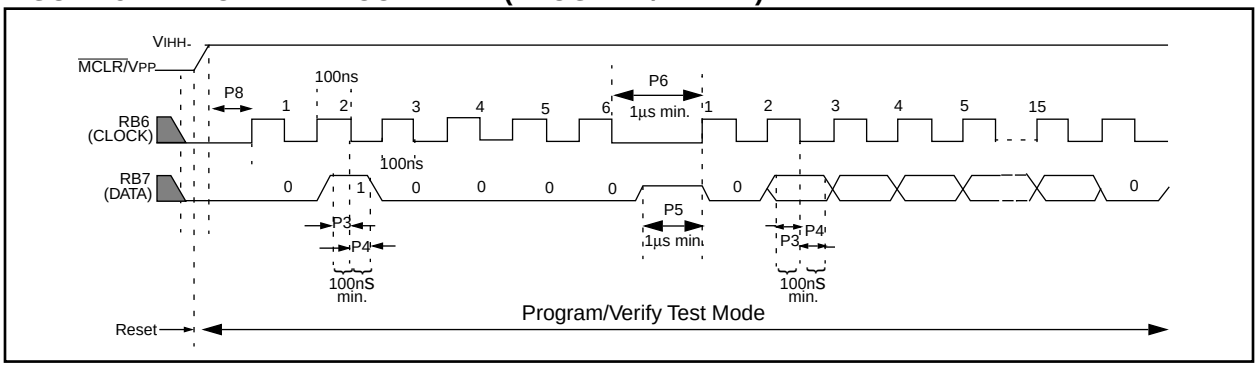


FIGURE 6: READ DATA COMMAND (PROGRAM/VERIFY)

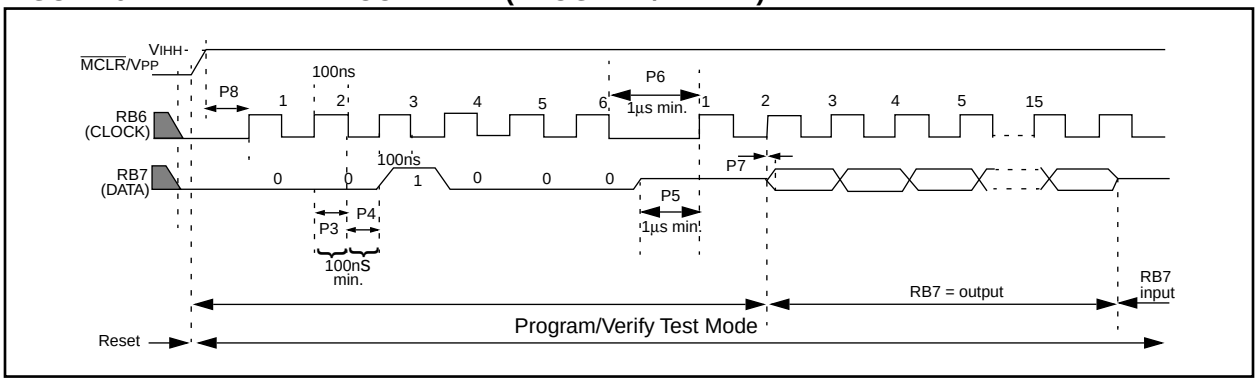
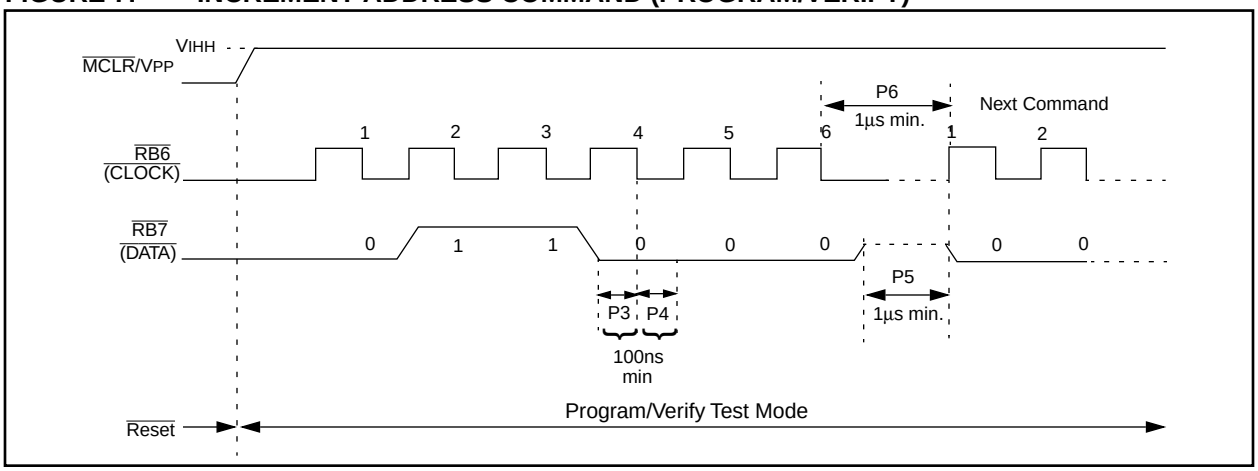


FIGURE 7: INCREMENT ADDRESS COMMAND (PROGRAM/VERIFY)



PIC16C6X/7X/9XX

In-Circuit Serial Programming for PIC16C6X/7X/9XX OTP Microcontrollers

This document includes the programming specifications for the following devices:

- PIC16C61
- PIC16C62
- PIC16C62A
- PIC16C63
- PIC16C64
- PIC16C64A
- PIC16C65
- PIC16C65A
- PIC16C71
- PIC16C710
- PIC16C711
- PIC16C72
- PIC16C73
- PIC16C73A
- PIC16C74
- PIC16C74A
- PIC16C66
- PIC16C67
- PIC16C76
- PIC16C77
- PIC16C620
- PIC16C621
- PIC16C622
- PIC16C923
- PIC16C924

PROGRAMMING THE PIC16C6X/7X/9XX

The PIC16C6X/7X/9XX can be programmed using a serial method. In serial mode the PIC16C6X/7X/9XX can be programmed while in the users system. This allows for increased design flexibility. This programming specification applies to PIC16C6X/7X/9XX devices in all packages.

Hardware Requirements

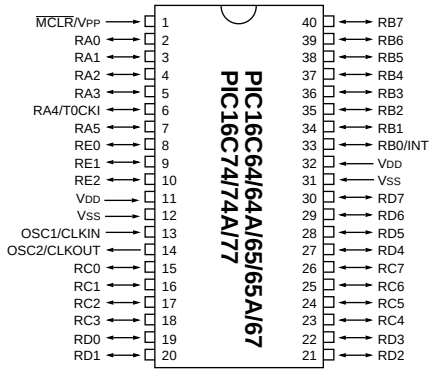
The PIC16C6X/7X/9XX requires two programmable power supplies, one for VDD (2.0V to 6.5V recommended) and one for VPP (12V to 14V). Both supplies should have a minimum resolution of 0.25V.

Programming Mode

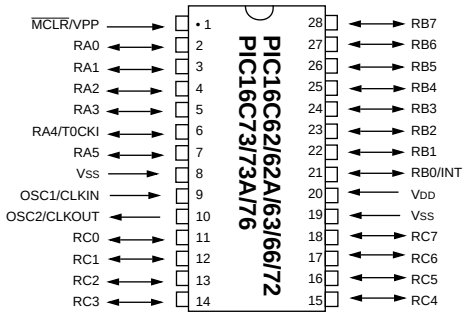
The programming mode for the PIC16C6X/7X/9XX allows programming of user program memory, special locations used for ID, and the configuration word for the PIC16C6X/7X/9XX.

Pin Diagrams

PDIP, Windowed CERDIP



PDIP, SOIC, Windowed CERDIP (300 mil)



Note: Peripheral pinout functions are not shown (see data sheets for full pinout information).

PIN DESCRIPTIONS (DURING PROGRAMMING):
PIC16C61/620/621/622/62/62A/63/64/64A/65/65A/66/67/71/73/73A/74/74A/76/77/710/711/923/924

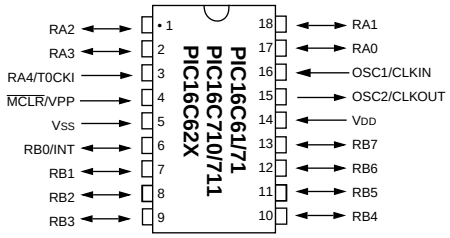
Pin Name	During Programming		
	Pin Name	Pin Type	Pin Description
RB6	CLOCK	I	Clock input
RB7	DATA	I/O	Data input/output
MCLR/VPP	VPP	P	Programming Power
VDD	VDD	P	Power Supply
VSS	VSS	P	Ground

Legend: I = input, O = Output, P = Power

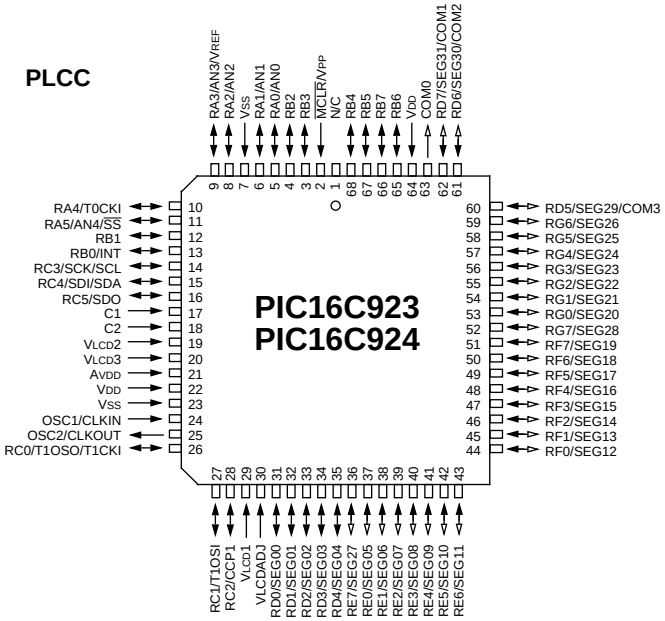
ICSP is a trademark of Microchip Technology Inc.

Pin Diagrams (Con't)

PDIP, SOIC, Windowed CERDIP



PLCC



PROGRAM MODE ENTRY

User Program Memory Map

The user memory space extends from 0x0000 to 0x1FFF (8K). Table 1 shows actual implementation of program memory in the PIC16C6X/7X/9XX family.

TABLE 1: IMPLEMENTATION OF PROGRAM MEMORY IN THE PIC16C6X/7X/9XX

Device	Program Memory Size
PIC16C61	0x000-0x3FF (1K)
PIC16C620	0x000 -0x1FF (0.5K)
PIC16C621	0x000 - 0x3FF (1K)
PIC16C622	0x000 -0x7FF (2K)
PIC16C62/62A	0x000 - 0x7FF (2K)
PIC16C63	0x000 - 0xFFF (4K)
PIC16C64/64A	0x000 - 0x7FF (2K)
PIC16C65/65A	0x000 - 0xFFF (4K)
PIC16C71	0x000 - 0x3FF (1K)
PIC16C710	0x000 - 0x1FF (0.5K)
PIC16C711	0x000 - 0x3FF (1K)
PIC16C72	0x000 - 0x7FF (2K)
PIC16C73/73A	0x000 - 0xFFF (4K)
PIC16C74/74A	0x000 - 0xFFF (4K)
PIC16C66	0x000 - 0x1FFF (8K)
PIC16C67	0x000 - 0x1FFF (8K)
PIC16C76	0x000 - 0x1FFF (8K)
PIC16C77	0x000 - 0x1FFF (8K)
PIC16C923/924	0x000 - 0xFFF (4K)

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 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.

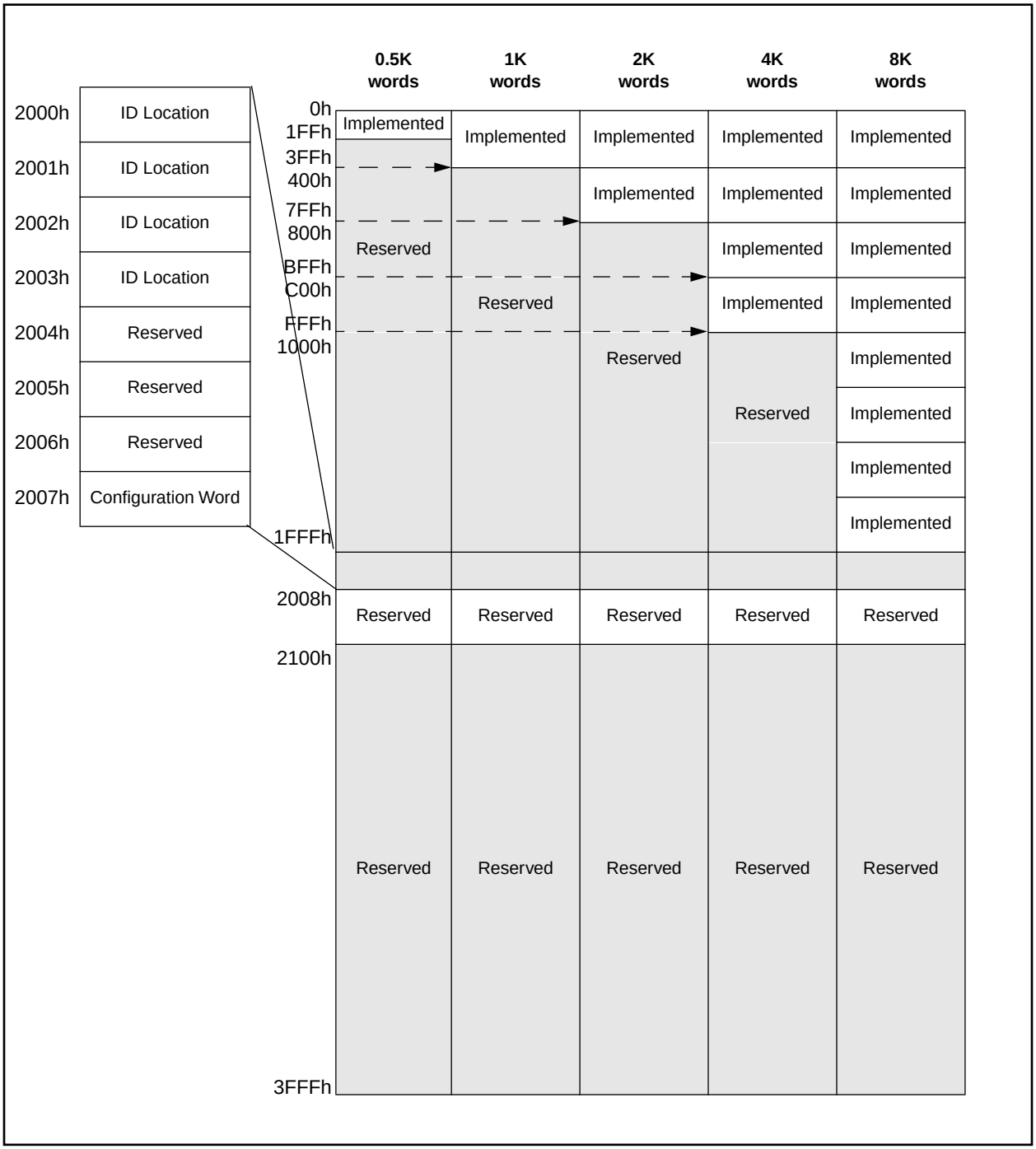
In the configuration memory space, 0x2000-0x207F or 0x2000-0x20FF are utilized. When in a configuration memory, as in the user memory, the 0x2000-0x2XFF segment is repeatedly accessed as PC exceeds 0x2XFF (see Figure 1).

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 1bbb 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 3.

FIGURE 1: PROGRAM MEMORY MAPPING



Program/Verify Mode

The program/verify mode is entered by holding pins RB6 and RB7 low while raising $\overline{MCLR}$ pin from V_{IL} to V_{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 $\overline{MCLR}$ pin was initially at V_{IL}). This means that all I/O are in the reset state (High impedance inputs).

Note:	The $\overline{MCLR}$ pin should be raised as quickly as possible from V_{IL} to V_{IHH}. this is to ensure that the device does not have the PC incremented while in valid operation range.

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 1 µs 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 µs 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 µs is required between a command and a data word (or another command).

The commands that are available are listed in Table 2.

LOAD CONFIGURATION

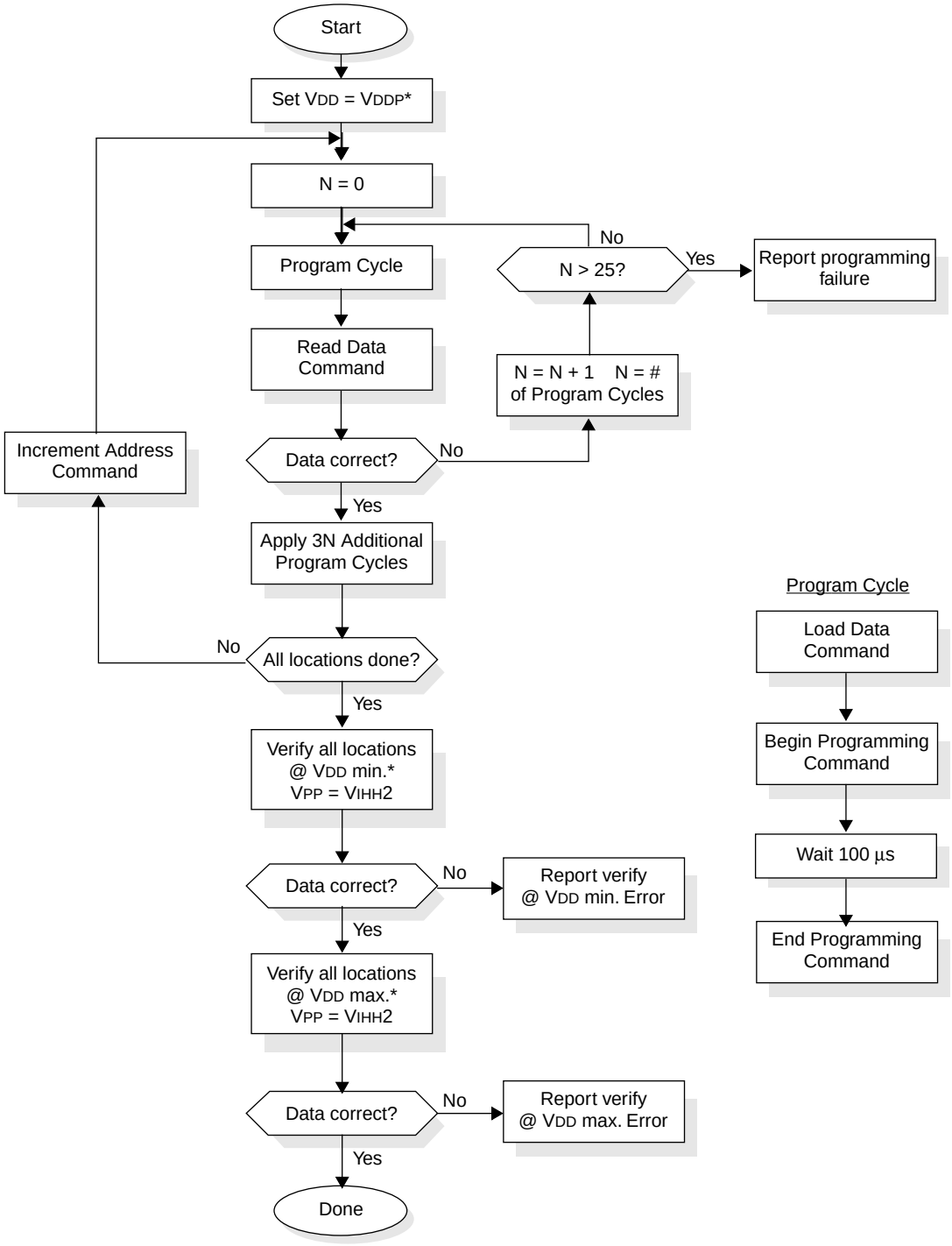
After receiving this command, the program counter (PC) will be set to 0x2000. By then applying 16 cycles to the clock pin, the chip will load 14-bits a “data word” as described above, to be programmed into the configuration memory. A description of the memory mapping schemes for normal operation and configuration mode operation is shown in Figure 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 $\overline{MCLR}$ low (V_{IL}).

TABLE 2: COMMAND MAPPING

Command	Mapping (MSb ... LSb)						Data
Load Configuration	0	0	0	0	0	0	0, data(14), 0
Load Data	0	0	0	0	1	0	0, data(14), 0
Read Data	0	0	0	1	0	0	0, data(14), 0
Increment Address	0	0	0	1	1	0	
Begin programming	0	0	1	0	0	0	
End Programming	0	0	1	1	1	0	

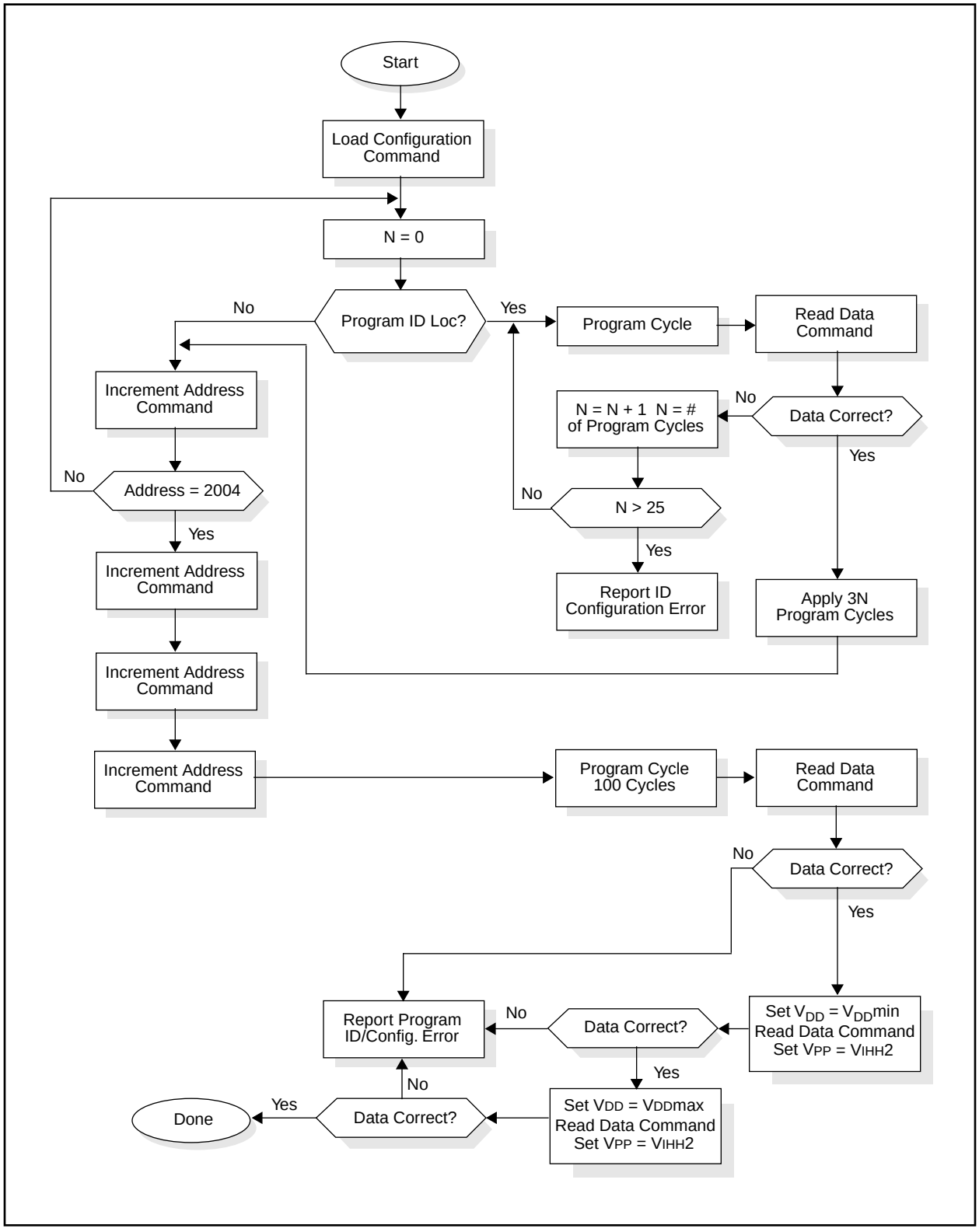
Note: The clock must be disabled during In-Circuit Serial Programming.

FIGURE 2: PROGRAM FLOW CHART - PIC16C6X/7X/9XX PROGRAM MEMORY



* VDDP = VDD range for programming (typically 4.75V - 5.25V).
VDDmin = Minimum VDD for device operation.
VDDmax = Maximum VDD for device operation.

FIGURE 3: PROGRAM FLOW CHART - PIC16C6X/7X/9XX CONFIGURATION WORD & ID LOCATIONS



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.

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 6.

INCREMENT ADDRESS

The PC is incremented when this command is received. A timing diagram of this command is shown in Figure 7.

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.

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.

Programming Algorithm Requires Variable V_{DD}

The PIC16C6X/7X/9XX uses an intelligent algorithm. The algorithm calls for program verification at V_{DD}min as well as V_{DD}max. Verification at V_{DD}min guarantees good “erase margin”. Verification at V_{DD}max guarantees good “program margin”.

The actual programming must be done with V_{DD} in the V_{DDP} range (4.75 - 5.25V).

V_{DDP} = V_{CC} range required during programming.

V_{DD} min. = minimum operating V_{DD} spec for the part.

V_{DD}max = maximum operating V_{DD} spec for the part.

Programmers must verify the PIC16C6X/7X/9XX at its specified V_{DD}max and V_{DD}min levels. Since Microchip may introduce future versions of the PIC16C6X/7X/9XX with a broader V_{DD} 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.

CONFIGURATION WORD

The PIC16C6X/7X/9XX family members have several configuration bits. These bits can be programmed (reads '0') or left unprogrammed (reads '1') to select various device configurations. Figure 4 provides an overview of configuration bits.

FIGURE 4: CONFIGURATION WORD BIT MAP

Bit Number:	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PIC16C61/71	—	—	—	—	—	—	—	—	—	CP0	PWRTE	WDTE	FOSC1	FOSC0
PIC16C62/64/65/73/74	—	—	—	—	—	—	—	0	CP1	CP0	PWRTE	WDTE	FOSC1	FOSC0
PIC16C710/711	CP0	CP0	CP0	CP0	CP0	CP0	CP0	BODEN	CP0	CP0	$\overline{\text{PWRTE}}$	WDTE	FOSC1	FOSC0
PIC16C62A/CR62/63/64A/CR64/65A/66/67/72/73A/74A/76/77/620/621/622	CP1	CP0	CP1	CP0	CP1	CP0	—	BODEN	CP1	CP0	$\overline{\text{PWRTE}}$	WDTE	FOSC1	FOSC0
PIC16C9XX	CP1	CP0	CP1	CP0	CP1	CP0	—	—	CP1	CP0	$\overline{\text{PWRTE}}$	WDTE	FOSC1	FOSC0

bit 6: **Reserved**, '–' write as '1' for PIC16C6X/7X/9XX

bit 5-4: **CP1:CP0**, Code Protect

Device	CP1	CP0	Code Protection
PIC16C622 PIC16C62/62A/63 PIC16C64/64A PIC16C65/65A/66/67 PIC16C72 PIC16C73/73A PIC16C74/74A/76/77 PIC16C9XX	0	0	All memory protected
	0	1	Upper 3/4 memory protected
	1	0	Upper 1/2 memory protected
	1	1	Code protection off
PIC16C61/71 PIC16C710/711	—	0	On
	—	1	Off
PIC16C620	0	0	All memory protected
	0	1	Do not use
	1	0	Do not use
	1	1	Code protection off
PIC16C621	0	0	All memory protected
	0	1	Upper 1/2 memory protected
	1	0	Do not use
	1	1	Code protection off

bit 6: **BODEN**, Brown Out Enable Bit

bit 4: **$\overline{\text{PWRTE}}$/PWRTE**, Power Up Timer Enable Bit
PIC16C61/62/64/65/71/73/74:
1 = Power up timer enabled
0 = Power up timer disabled
PIC16C620/621/622/62A/63/65A/66/67/72/73A/74A/76/77:
0 = Power up timer enabled
1 = Power up timer disabled

bit 3-2: **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

CODE PROTECTION

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

In PIC16C61/71 it is still possible to program locations 0x000 through 0x03F, after code protection. For all other devices, writing to all protected memory is disabled.

Programming Locations 0x0000 to 0x03F after Code Protection

For PIC16C61/71 devices, once code protection is enabled, all program memory locations read out in a scrambled fashion. The ID locations and the configuration word also read out in a scrambled fashion. Further programming is disabled for locations 0x040 and above. It is possible to program the ID locations and the configuration word.

For PIC16C61/71 devices, program memory locations 0x000 through 0x03F are essentially unprotected, i.e., these locations can be further programmed after code protection is enabled. However, since the data reads out in a scrambled fashion, to correctly overprogram these locations, the programmer must program seven bits at a time. For example, to program 0x3AD2 ("`11 1010 1101 0010`") in a blank location, first program the location with "`11 1111 1101 0010`" and verify scrambled output to be "`xx xxxx x101 0010`". Next, program the location with "`11 1010 1101 0010`" and verify scrambled output to be "`xx xxxx x101 1000`".

For all other PIC16C6X/7X/9XX 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.

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 3: CONFIGURATION WORD

PIC16C61

To code protect:

- Protect all memory `1111111110xxxx`

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0x2007)	Read Scrambled, Write Enabled	Read Unscrambled, Write Enabled
Unprotected memory segment	Read Scrambled, Write Enabled	Read Unscrambled, Write Enabled
Protected memory segment	Read Scrambled, Write Disabled	Read Unscrambled, Write Enabled
ID Locations (0x2000 : 0x2003)	Read Scrambled, Write Enabled	Read Unscrambled, Write Enabled

PIC16C620

To code protect:

- Protect all memory `0000001x00xxxx`
- No code protection `1111111x11xxxx`

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

PIC16C621

To code protect:

- Protect all memory `0000001X00XXXX`
- Protect upper 1/2 memory `0101011X01XXXX`
- No code protection `1111111X11XXXX`

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

PIC16C622

To code protect:

- Protect all memory `0000001X00XXXX`
- Protect upper 3/4 memory `0101011X01XXXX`
- Protect upper 1/2 memory `1010101X10XXXX`
- No code protection `1111111X11XXXX`

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

PIC16C62

To code protect:

- Protect all memory `1111111000XXXX`
- Protect upper 3/4 memory `1111111001XXXX`
- Protect upper 1/2 memory `1111111010XXXX`
- No code protection `1111111011XXXX`

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0x2007)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Unprotected memory segment	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Protected memory segment	Read Scrambled, Write Disabled	Read Unscrambled, Write Enabled
ID Locations (0x2000 : 0x2003)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled

PIC16C62A

To code protect:

- Protect all memory `0000001X00XXXX`
- Protect upper 3/4 memory `0101011X01XXXX`
- Protect upper 1/2 memory `1010101X10XXXX`
- No code protection `1111111X11XXXX`

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

PIC16C63

To code protect:

- Protect all memory `0000001x00xxxx`
- Protect upper 3/4 memory `0101011x01xxxx`
- Protect upper 1/2 memory `1010101x10xxxx`
- No code protection `1111111x11xxxx`

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

PIC16C64

To code protect:

- Protect all memory `1111111000xxxx`
- Protect upper 3/4 memory `1111111001xxxx`
- Protect upper 1/2 memory `1111111010xxxx`
- No code protection `1111111011xxxx`

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0x2007)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Unprotected memory segment	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Protected memory segment	Read Scrambled, Write Disabled	Read Unscrambled, Write Enabled
ID Locations (0x2000 : 0x2003)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled

PIC16C64A

To code protect:

- Protect all memory `0000001x00xxxx`
- Protect upper 3/4 memory `0101011x01xxxx`
- Protect upper 1/2 memory `1010101x10xxxx`
- No code protection `1111111x11xxxx`

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

PIC16C65

To code protect:

- Protect all memory `1111111000xxxx`
- Protect upper 3/4 memory `1111111001xxxx`
- Protect upper 1/2 memory `1111111010xxxx`
- No code protection `1111111011xxxx`

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0x2007)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Unprotected memory segment	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Protected memory segment	Read Scrambled, Write Disabled	Read Unscrambled, Write Enabled
ID Locations (0x2000 : 0x2003)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled

PIC16C65A

To code protect:

- Protect all memory `0000001x00xxxx`
- Protect upper 3/4 memory `0101011x01xxxx`
- Protect upper 1/2 memory `1010101x10xxxx`
- No code protection `1111111x11xxxx`

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

PIC16C66

To code protect:

- Protect all memory `0000001x00xxxx`
- Protect upper 3/4 memory `0101011x01xxxx`
- Protect upper 1/2 memory `1010101x10xxxx`
- No code protection `1111111x11xxxx`

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

PIC16C67

To code protect:

- Protect all memory `0000001x00xxxx`
- Protect upper 3/4 memory `0101011x01xxxx`
- Protect upper 1/2 memory `1010101x10xxxx`
- No code protection `1111111x11xxxx`

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

PIC16C710

To code protect:

- Protect all memory `0000000x00xxxx`

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

PIC16C71

To code protect:

- Protect all memory `1111111110xxxx`

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0x2007)	Read Scrambled, Write Enabled	Read Unscrambled, Write Enabled
Unprotected memory segment	Read Scrambled, Write Enabled	Read Unscrambled, Write Enabled
Protected memory segment	Read Scrambled, Write Disabled	Read Unscrambled, Write Enabled
ID Locations (0x2000 : 0x2003)	Read Scrambled, Write Enabled	Read Unscrambled, Write Enabled

PIC16C711

To code protect:

- Protect all memory `0000000x00xxxx`

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

PIC16C72

To code protect:

- Protect all memory `0000001x00xxxx`
- Protect upper 3/4 memory `0101011x01xxxx`
- Protect upper 1/2 memory `1010101x10xxxx`
- No code protection `1111111x11xxxx`

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

PIC16C73

To code protect:

- Protect all memory `1111111000xxxx`
- Protect upper 3/4 memory `1111111001xxxx`
- Protect upper 1/2 memory `1111111010xxxx`
- No code protection `1111111011xxxx`

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0x2007)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Unprotected memory segment	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Protected memory segment	Read Scrambled, Write Disabled	Read Unscrambled, Write Enabled
ID Locations (0x2000 : 0x2003)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled

PIC16C73A

To code protect:

- Protect all memory `0000001x00xxxx`
- Protect upper 3/4 memory `0101011x01xxxx`
- Protect upper 1/2 memory `1010101x10xxxx`
- No code protection `1111111x11xxxx`

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

PIC16C74

To code protect:

- Protect all memory `1111111000xxxx`
- Protect upper 3/4 memory `1111111001xxxx`
- Protect upper 1/2 memory `1111111010xxxx`
- No code protection `1111111011xxxx`

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0x2007)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Unprotected memory segment	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
Protected memory segment	Read Scrambled, Write Disabled	Read Unscrambled, Write Enabled
ID Locations (0x2000 : 0x2003)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled

PIC16C74A

To code protect:

- Protect all memory `0000001x00xxxx`
- Protect upper 3/4 memory `0101011x01xxxx`
- Protect upper 1/2 memory `1010101x10xxxx`
- No code protection `1111111x11xxxx`

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

PIC16C76

To code protect:

- Protect all memory `0000001x00xxxx`
- Protect upper 3/4 memory `0101011x01xxxx`
- Protect upper 1/2 memory `1010101x10xxxx`
- No code protection `1111111x11xxxx`

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

PIC16C77

To code protect:

- Protect all memory `0000001x00xxxx`
- Protect upper 3/4 memory `0101011x01xxxx`
- Protect upper 1/2 memory `1010101x10xxxx`
- No code protection `1111111x11xxxx`

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

PIC16C923

To code protect:

- Protect all memory `0000001x00xxxx`
- Protect upper 3/4 memory `0101011x01xxxx`
- Protect upper 1/2 memory `1010101x10xxxx`
- No code protection `1111111x11xxxx`

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

PIC16C924

To code protect:

- Protect all memory `0000001x00xxxx`
- Protect upper 3/4 memory `0101011x01xxxx`
- Protect upper 1/2 memory `1010101x10xxxx`
- No code protection `1111111x11xxxx`

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

Legend: X = Don't care

Checksum

CHECKSUM CALCULATIONS

Checksum is calculated by reading the contents of the PIC16C6X/7X/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 PIC16C8X devices is shown in Table 4.

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: CHECKSUM COMPUTATION

Device	Code Protect	Checksum*	Blank Value	0x25E6 at 0 and max address
PIC16C61	OFF	SUM[0x000:0x3FF] + CFGW & 0x001F + 0x3FE0	0x3BFF	0x07CD
	ON	SUM_XNOR7[0x000:0x3FF] + (CFGW & 0x001F | 0x0060)	0xFC6F	0xFC15
PIC16C620	OFF	SUM[0x000:0x1FF] + CFGW & 0x3F7F	0x3D7F	0x094D
	ON	SUM_ID + CFGW & 0x3F7F	0x3DCE	0x099C
PIC16C621	OFF	SUM[0x000:0x3FF] + CFGW & 0x3F7F	0x3B7F	0x074D
	1/2	SUM[0x000:0x1FF] + CFGW & 0x3F7F + SUM_ID	0x4EDE	0x0093
	ALL	CFGW & 0x3F7F + SUM_ID	0x3BCE	0x079C
PIC16C622	OFF	SUM[0x000:0x7FF] + CFGW & 0x3F7F	0x377F	0x034D
	1/2	SUM[0x000:0x3FF] + CFGW & 0x3F7F + SUM_ID	0x5DEE	0x0FA3
	3/4	SUM[0x000:0x1FF] + CFGW & 0x3F7F + SUM_ID	0x4ADE	0xFC93
	ALL	CFGW & 0x3F7F + SUM_ID	0x37CE	0x039C
PIC16C62	OFF	SUM[0x000:0x7FF] + CFGW & 0x003F + 0x3F80	0x37BF	0x038D
	1/2	SUM[0x000:0x3FF] + SUM_XNOR7[0x400:0x7FF] + CFGW & 0x003F + 0x3F80	0x37AF	0x1D69
	3/4	SUM[0x000:0x1FF] + SUM_XNOR7[0x200:0x7FF] + CFGW & 0x003F + 0x3F80	0x379F	0x1D59
	ALL	SUM_XNOR7[0x000:0x7FF] + CFGW & 0x003F + 0x3F80	0x378F	0x3735
PIC16C62A	OFF	SUM[0x000:0x7FF] + CFGW & 0x3F7F	0x377F	0x034D
	1/2	SUM[0x000:0x3FF] + CFGW & 0x3F7F + SUM_ID	0x5DEE	0x0FA3
	3/4	SUM[0x000:0x1FF] + CFGW & 0x3F7F + SUM_ID	0x4ADE	0xFC93
	ALL	CFGW & 0x3F7F + SUM_ID	0x37CE	0x039C
PIC16C63	OFF	SUM[0x000:0xFFF] + CFGW & 0x3F7F	0x2F7F	0xFB4D
	1/2	SUM[0x000:0x7FF] + CFGW & 0x3F7F + SUM_ID	0x51EE	0x03A3
	3/4	SUM[0x000:0x3FF] + CFGW & 0x3F7F + SUM_ID	0x40DE	0xF293
	ALL	CFGW & 0x3F7F + SUM_ID	0x2FCE	0xFB9C

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, location_a = 0x123 and location_b = 0x456, then SUM_XNOR7 [location_a : location_b] = 0x001F.
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

TABLE 4: CHECKSUM COMPUTATION (CONTINUED)

Device	Code Protect	Checksum*	Blank Value	0x25E6 at 0 and max address
PIC16C64	OFF	SUM[0x000:0x7FF] + CFGW & 0x003F + 0x3F80	0x37BF	0x038D
	1/2	SUM[0x000:0x3FF] + SUM_XNOR7[0x400:0x7FF] + CFGW & 0x003F + 0x3F80	0x37AF	0x1D69
	3/4	SUM[0x000:0x1FF] + SUM_XNOR7[0x200:0x7FF] + CFGW & 0x003F + 0x3F80	0x379F	0x1D59
	ALL	SUM_XNOR7[0x000:0x7FF] + CFGW & 0x003F + 0x3F80	0x378F	0x3735
PIC16C64A	OFF	SUM[0x000:0x7FF] + CFGW & 0x3F7F	0x377F	0x034D
	1/2	SUM[0x000:0x3FF] + CFGW & 0x3F7F + SUM_ID	0x5DEE	0x0FA3
	3/4	SUM[0x000:0x1FF] + CFGW & 0x3F7F + SUM_ID	0x4ADE	0xFC93
	ALL	CFGW & 0x3F7F + SUM_ID	0x37CE	0x039C
PIC16C65	OFF	SUM[0x000:0xFFF] + CFGW & 0x003F + 0x3F80	0x2FBF	0xFB8D
	1/2	SUM[0x000:0x7FF] + SUM_XNOR7[0x800:FFF] + CFGW & 0x003F + 0x3F80	0x2FAF	0x1569
	3/4	SUM[0x000:0x3FF] + SUM_XNOR7[0x400:FFF] + CFGW & 0x003F + 0x3F80	0x2F9F	0x1559
	ALL	SUM_XNOR7[0x000:0xFFF] + CFGW & 0x003F + 0x3F80	0x2F8F	0x2F35
PIC16C65A	OFF	SUM[0x000:0xFFF] + CFGW & 0x3F7F	0x2F7F	0xFB4D
	1/2	SUM[0x000:0x7FF] + CFGW & 0x3F7F + SUM_ID	0x51EE	0x03A3
	3/4	SUM[0x000:0x3FF] + CFGW & 0x3F7F + SUM_ID	0x40DE	0xF293
	ALL	CFGW & 0x3F7F + SUM_ID	0x2FCE	0xFB9C
PIC16C66	OFF	SUM[0x000:0x1FFF] + CFGW & 0x3F7F	0x1F7F	0xEB4D
	1/2	SUM[0x000:0xFFF] + CFGW & 0x3F7F + SUM_ID	0x39EE	0xEBA3
	3/4	SUM[0x000:0x7FF] + CFGW & 0x3F7F + SUM_ID	0x2CDE	0xDE93
	ALL	CFGW & 0x3F7F + SUM_ID	0x1FCE	0xEB9C
PIC16C67	OFF	SUM[0x000:0x1FFF] + CFGW & 0x3F7F	0x1F7F	0xEB4D
	1/2	SUM[0x000:0xFFF] + CFGW & 0x3F7F + SUM_ID	0x39EE	0xEBA3
	3/4	SUM[0x000:0x7FF] + CFGW & 0x3F7F + SUM_ID	0x2CDE	0xDE93
	ALL	CFGW & 0x3F7F + SUM_ID	0x1FCE	0xEB9C
PIC16C710	OFF	SUM[0x000:0x1FF] + CFGW & 0x3FFF	0x3DFF	0x09CD
	ON	SUM[0x00:0x3F] + CFGW & 0x3FFF + SUM_ID	0x3E0E	0xEFC3
PIC16C71	OFF	SUM[0x000:0x3FF] + CFGW & 0x001F + 0x3FE0	0x3BFF	0x07CD
	ON	SUM_XNOR7[0x000:0x3FF] + (CFGW & 0x001F | 0x0060)	0xFC6F	0xFC15
PIC16C711	OFF	SUM[0x000:0x3FF] + CFGW & 0x3FFF	0x3BFF	0x07CD
	ON	SUM[0x00:0x3F] + CFGW & 0x3FFF + SUM_ID	0x3C0E	0xEDC3
PIC16C72	OFF	SUM[0x000:0x7FF] + CFGW & 0x3F7F	0x377F	0x034D
	1/2	SUM[0x000:0x3FF] + CFGW & 0x3F7F + SUM_ID	0x5DEE	0x0FA3
	3/4	SUM[0x000:0x1FF] + CFGW & 0x3F7F + SUM_ID	0x4ADE	0xFC93
	ALL	CFGW & 0x3F7F + SUM_ID	0x37CE	0x039C
PIC16C73	OFF	SUM[0x000:0xFFF] + CFGW & 0x003F + 0x3F80	0x2FBF	0xFB8D
	1/2	SUM[0x000:0x7FF] + SUM_XNOR7[0x800:FFF] + CFGW & 0x003F + 0x3F80	0x2FAF	0x1569
	3/4	SUM[0x000:0x3FF] + SUM_XNOR7[0x400:FFF] + CFGW & 0x003F + 0x3F80	0x2F9F	0x1559
	ALL	SUM_XNOR7[0x000:0xFFF] + CFGW & 0x003F + 0x3F80	0x2F8F	0x2F35

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, location_a = 0x123 and location_b = 0x456, then SUM_XNOR7 [location_a : location_b] = 0x001F.

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

TABLE 4: CHECKSUM COMPUTATION (CONTINUED)

Device	Code Protect	Checksum*	Blank Value	0x25E6 at 0 and max address
PIC16C73A	OFF	SUM[0x000:0xFFF] + CFGW & 0x3F7F	0x2F7F	0xFB4D
	1/2	SUM[0x000:0x7FF] + CFGW & 0x3F7F + SUM_ID	0x51EE	0x03A3
	3/4	SUM[0x000:0x3FF] + CFGW & 0x3F7F + SUM_ID	0x40DE	0xF293
	ALL	CFGW & 0x3F7F + SUM_ID	0x2FCE	0xFB9C
PIC16C74	OFF	SUM[0x000:0xFFF] + CFGW & 0x003F + 0x3F80	0x2FBF	0xFB8D
	1/2	SUM[0x000:0x7FF] + SUM_XNOR7[0x800:FFF] + CFGW & 0x003F + 0x3F80	0x2FAF	0x1569
	3/4	SUM[0x000:0x3FF] + SUM_XNOR7[0x400:FFF] + CFGW & 0x003F + 0x3F80	0x2F9F	0x1559
	ALL	SUM_XNOR7[0x000:0xFFF] + CFGW & 0x003F + 0x3F80	0x2F8F	0x2F35
PIC16C74A	OFF	SUM[0x000:0xFFF] + CFGW & 0x3F7F	0x2F7F	0xFB4D
	1/2	SUM[0x000:0x7FF] + CFGW & 0x3F7F + SUM_ID	0x51EE	0x03A3
	3/4	SUM[0x000:0x3FF] + CFGW & 0x3F7F + SUM_ID	0x40DE	0xF293
	ALL	CFGW & 0x3F7F + SUM_ID	0x2FCE	0xFB9C
PIC16C76	OFF	SUM[0x000:0x1FFF] + CFGW & 0x3F7F	0x1F7F	0xEB4D
	1/2	SUM[0x000:0xFFF] + CFGW & 0x3F7F + SUM_ID	0x39EE	0xEBA3
	3/4	SUM[0x000:0x7FF] + CFGW & 0x3F7F + SUM_ID	0x2CDE	0xDE93
	ALL	CFGW & 0x3F7F + SUM_ID	0x1FCE	0xEB9C
PIC16C77	OFF	SUM[0x000:0x1FFF] + CFGW & 0x3F7F	0x1F7F	0xEB4D
	1/2	SUM[0x000:0xFFF] + CFGW & 0x3F7F + SUM_ID	0x39EE	0xEBA3
	3/4	SUM[0x000:0x7FF] + CFGW & 0x3F7F + SUM_ID	0x2CDE	0xDE93
	ALL	CFGW & 0x3F7F + SUM_ID	0x1FCE	0xEB9C
PIC16C923	OFF	SUM[0x000:0xFFF] + CFGW & 0x3F3F	0x2F3F	0xFB0D
	1/2	SUM[0x000:0x7FF] + CFGW & 0x3F3F + SUM_ID	0x516E	0x0323
	3/4	SUM[0x000:0x3FF] + CFGW & 0x3F3F + SUM_ID	0x405E	0xF213
	ALL	CFGW & 0x3F3F + SUM_ID	0x2F4E	0xFB1C
PIC16C924	OFF	SUM[0x000:0xFFF] + CFGW & 0x3F3F	0x2F3F	0xFB0D
	1/2	SUM[0x000:0x7FF] + CFGW & 0x3F3F + SUM_ID	0x516E	0x0323
	3/4	SUM[0x000:0x3FF] + CFGW & 0x3F3F + SUM_ID	0x405E	0xF213
	ALL	CFGW & 0x3F3F + SUM_ID	0x2F4E	0xFB1C

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, location_a = 0x123 and location_b = 0x456, then SUM_XNOR7 [location_a : location_b] = 0x001F.

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

PROGRAM/VERIFY MODE ELECTRICAL CHARACTERISTICS

TABLE 5: AC/DC CHARACTERISTICS TIMING REQUIREMENTS FOR PROGRAM/VERIFY TEST MODE

Standard Operating Conditions Operating Temperature: +10°C ≤ TA ≤ +40°C, unless otherwise stated, (20°C recommended) Operating Voltage: 4.5V ≤ VDD ≤ 5.5V, unless otherwise stated.							
Parameter No.	**Sym.**	**Characteristic**	**Min.**	**Typ.**	**Max.**	**Units**	**Conditions**
General							
PD1	VDDP	Supply voltage during programming	4.75	5.0	5.25	V	
PD2	IDDP	Supply current (from VDD) during programming	–	–	20	mA	
PD3	VDDV	Supply voltage during verify	VDDmin	–	VDDmax	V	Note 1
PD4	VIHH1	Voltage on $\overline{MCLR}$/VPP during programming	12.75	–	13.25	V	Note 2
PD5	VIHH2	Voltage on $\overline{MCLR}$/VPP during verify	VDD + 4.0	–	13.5	–	
PD6	IPP	Programming supply current (from VPP)	–	–	50	mA	
PD9	VIH1	(RB6, RB7) input high level	0.8 VDD	–	–	V	Schmitt Trigger input
PD8	VIL1	(RB6, RB7) input low level	0.2 VDD	–	–	V	Schmitt Trigger input
Serial Program Verify							
P1	TR	$\overline{MCLR}$/VPP rise time (VSS to VHH) for test mode entry	–	–	8.0	µs	
P2	Tf	$\overline{MCLR}$ Fall time	–	–	8.0	µs	
P3	Tset1	Data in setup time before clock ↓	100	–	–	ns	
P4	Thld1	Data in hold time after clock ↓	100	–	–	ns	
P5	Tdly1	Data input not driven to next clock input (delay required between command/data or command/command)	1.0	–	–	µs	
P6	Tdly2	Delay between clock ↓ to clock ↑ of next command or data	1.0	–	–	µs	
P7	Tdly3	Clock ↑ to date out valid (during read data)	200	–	–	ns	
P8	Thld0	Hold time after $\overline{MCLR}$ ↑	2	–	–	µs	

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 5: LOAD DATA COMMAND (PROGRAM/VERIFY)

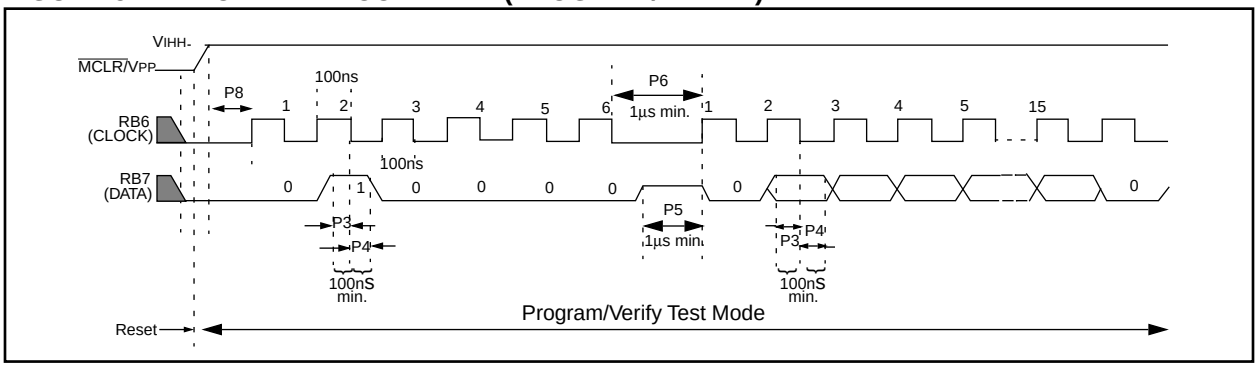


FIGURE 6: READ DATA COMMAND (PROGRAM/VERIFY)

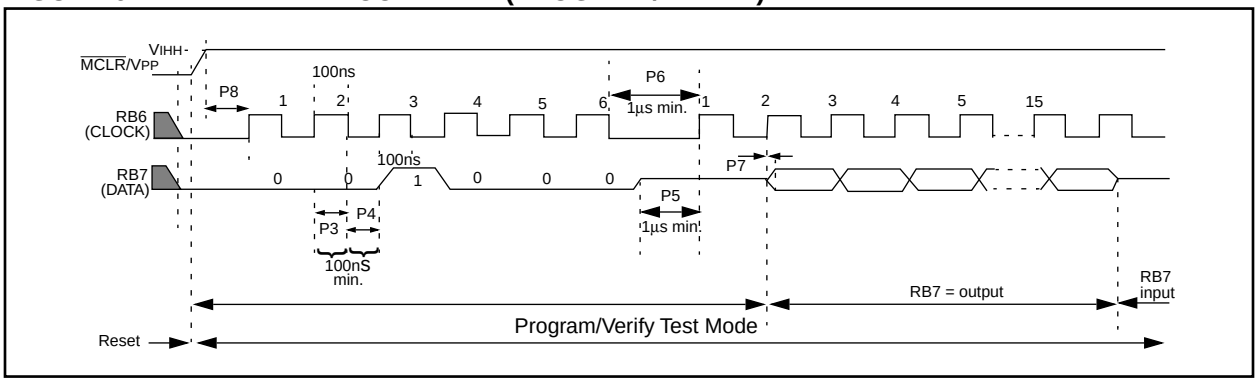
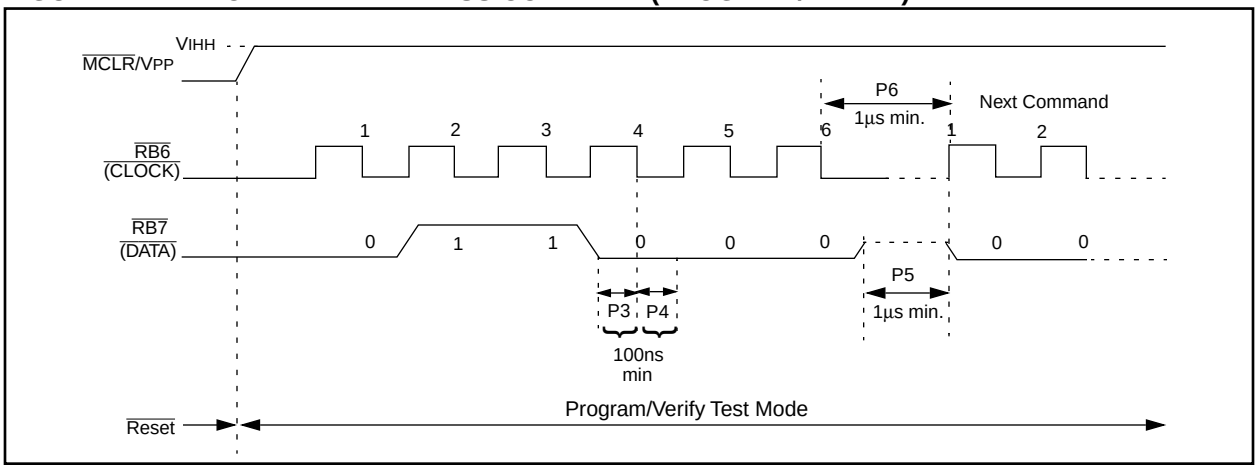


FIGURE 7: INCREMENT ADDRESS COMMAND (PROGRAM/VERIFY)



NOTES:

In-Circuit Serial Programming for PIC17CXXX OTP Microcontrollers

PROGRAMMING THE PIC17CXXX

PIC17CXXX microcontrollers (MCU) can be programmed in serial mode using an RS-232 or equivalent serial interface. The serial programming mode can program the PIC17CXXX while in the user's system. This allows for increased design flexibility and a faster time to market for the user's application. The PIC17CXXX uses the TABLWT and TLWT instructions to program locations in its program memory. It uses the TABLRD and TLRD instructions to verify the program locations in its program memory.

Hardware Requirements

The PIC17CXXX requires one fixed 13V supply. The supply should have a minimum resolution of 250 mV. Since the PIC17CXXX is actually executing code when doing In-Circuit Serial Programming (ICSP), a clock is required to run the device. The programming voltage (V_{PP}) must be present in order for the EPROM cells to get programmed.

Programming Margin

In order to ensure that the EPROM cell is fully programmed and can be read as a 0 or a 1, over the entire voltage range, it is recommended that the V_{DD} voltage during program/verify be kept at the highest operating voltage of the system.

TABLE 1: PARAMETER SPECIFICATIONS

Characteristic	Symbol	Min	Typical	Max	Units	Conditions
Supply voltage	V_{DD}	4.75	5.0	5.25	V	
Programming Voltage	V_{PP}	12.5	-	13.5	V	
Current into $\overline{MCLR}/V_{PP}$ pin	I_{PP}	-	25	50	mA	
Supply current during programming	I_{DDP}	-	-	30	mA	
Programming pulse width	T_{PROG}	10	100	1000	µs	Terminated by an external/internal interrupt or reset

Firmware Requirements

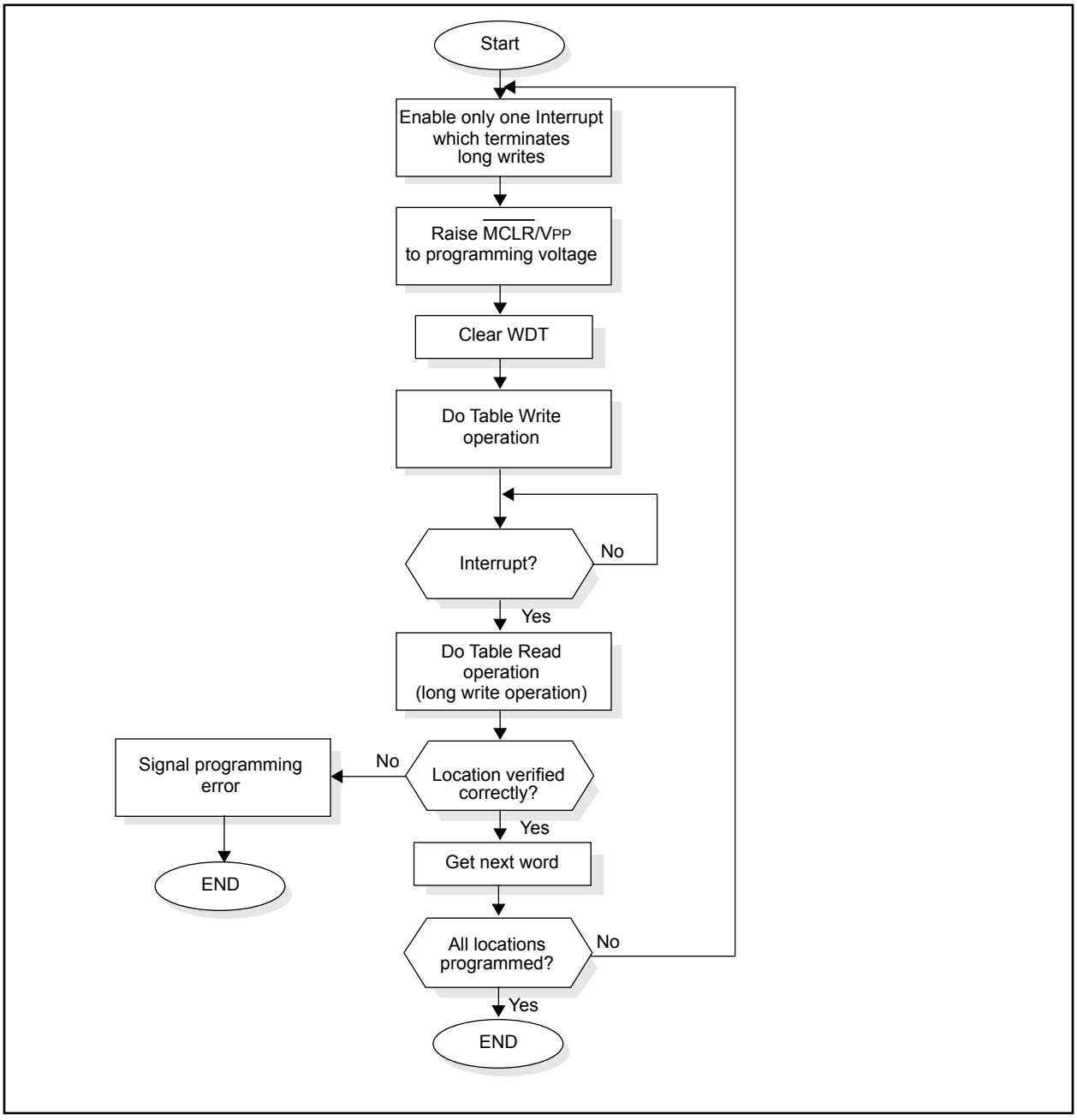
The PIC17CXXX executes some boot code during ICSP. This boot code retrieves serial data from the USART/SCI and programs memory locations using the TABLWT and TLWT instructions. Using a standard Hex format for the serial data, the address to be programmed can also be retrieved serially and loaded in the TBLPTR (Table pointer) in order to program the desired program memory location.

A table write to internal memory causes a long write operation. The long write operation is necessary for the internal program cells to get programmed (programming pulse width, TPROG). During a long write operation, instruction execution is halted. Instruction execution can be resumed either by an enabled interrupt or reset. In order to ensure that the EPROM cell has been well programmed, a minimum time is required before the long write is terminated. This time should exceed the programming pulse width or TPROG.

The flowchart in Figure 1 depicts the sequence of events to follow when programming an internal program memory location.

Once all desired locations are programmed, the VPP voltage is disabled, the boot code is terminated and the programmed code is executed.

FIGURE 1: ICSP FLOWCHART



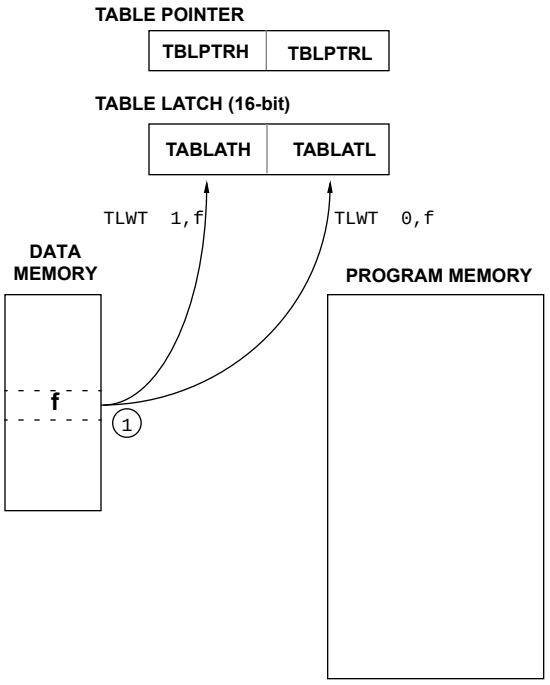
TABLWT Instruction

The PIC17CXXX has four instructions that allow the processor to move data from the data memory space to the program memory space, and vice versa. Since the program memory space is 16-bits wide and the data memory space is 8-bits wide, two operations are required to move 16-bit values to/from the data memory.

The `TLWT t,f` and `TABLWT t,i,f` instructions are used to write data from the data memory space to the program memory space. The `TLRD t,f` and `TABLRD t,i,f` instructions are used to write data from the program memory space to the data memory space.

Figure 2 through Figure 5 show the operation of these four instructions.

FIGURE 2: TLWT INSTRUCTION OPERATION



Note 1: 8-bit value, from register 'f', loaded into the high or low byte in TABLAT (16-bit).

Terminating Long writes

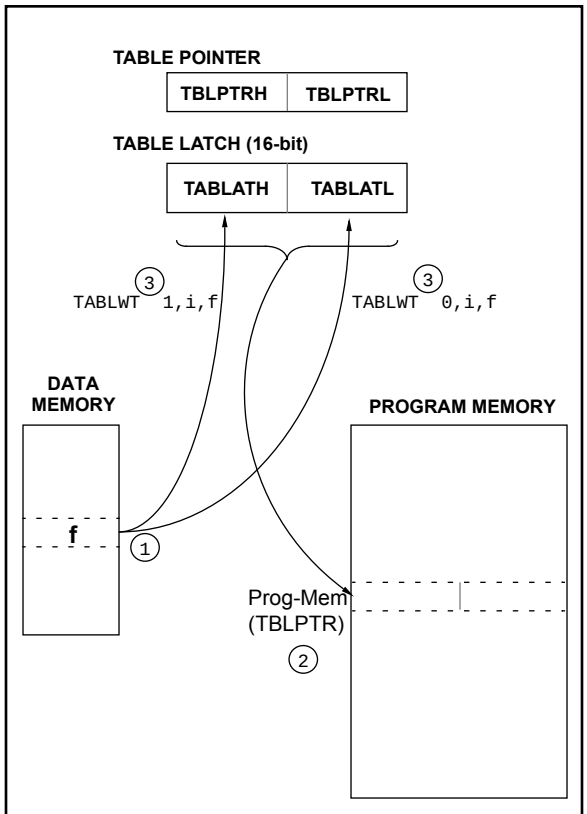
An interrupt source or reset are the only events that terminate a long write operation. Terminating the long write from an interrupt source requires that the interrupt enable and flag bits be set.

If the T0CKI, RA0/INT, or TMR0 interrupt source is used to terminate the long write; the interrupt flag, of the highest priority enabled interrupt, will first terminate the long write and then automatically be cleared.

If a peripheral interrupt source is used to terminate the long write, the interrupt enable and flag bits must be set. The interrupt flag will not be automatically cleared upon the vectoring to the interrupt vector address.

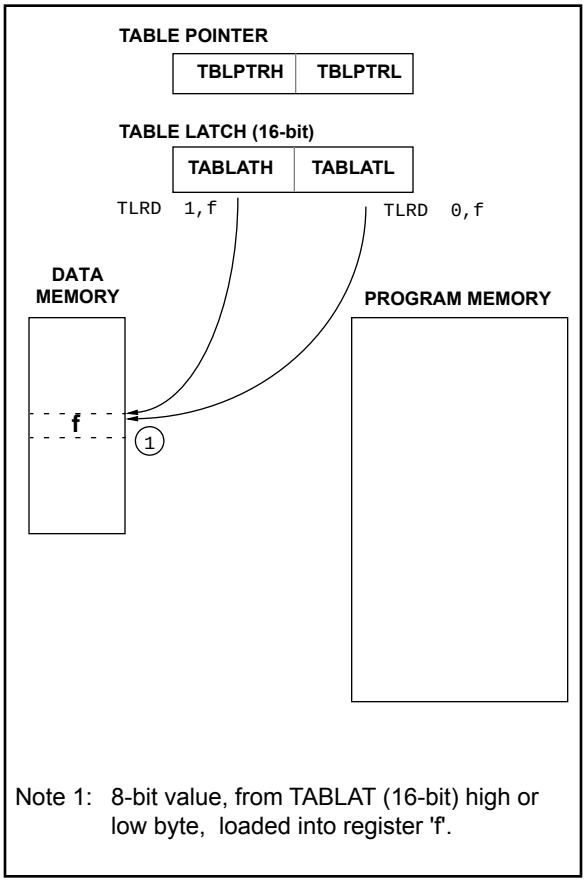
If the GLINTD bit is set prior to the long write, when the long write is terminated, the program will not vector to the interrupt address.

FIGURE 3: TABLWT INSTRUCTION OPERATION



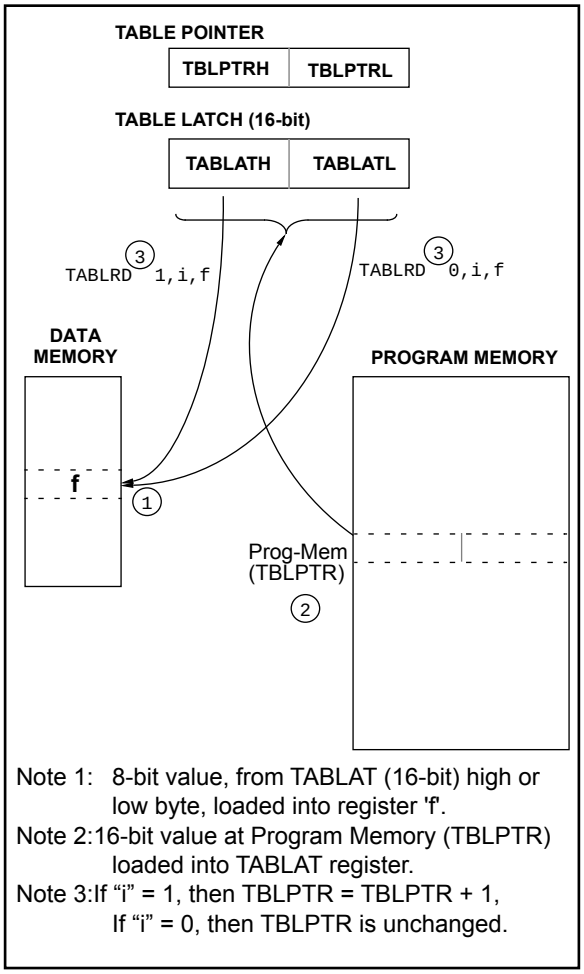
Note 1: 8-bit value, from register 'f', loaded into the high or low byte in TABLAT (16-bit).
Note 2:16-bit TABLAT value written to address Program Memory (TBLPTR).
Note 3:If "i" = 1, then TBLPTR = TBLPTR + 1, If "i" = 0, then TBLPTR is unchanged.

FIGURE 4: TLRD INSTRUCTION OPERATION



Note 1: 8-bit value, from TABLAT (16-bit) high or low byte, loaded into register 'f'.

FIGURE 5: TABLRD INSTRUCTION OPERATION



Note 1: 8-bit value, from TABLAT (16-bit) high or low byte, loaded into register 'f'.
Note 2:16-bit value at Program Memory (TBLPTR) loaded into TABLAT register.
Note 3:If “i” = 1, then TBLPTR = TBLPTR + 1, If “i” = 0, then TBLPTR is unchanged.

PIC16F8X

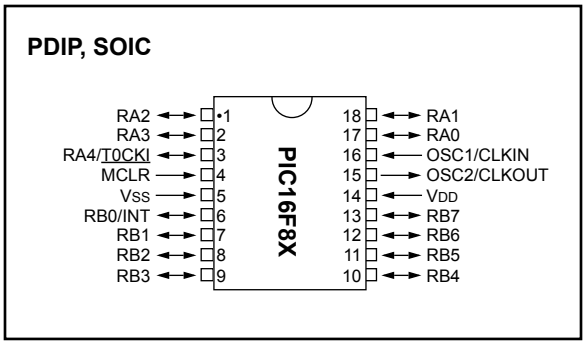
In-Circuit Serial Programming for PIC16F8X FLASH Microcontrollers

This document includes the programming specifications for the following devices:

- PIC16F83
- PIC16CR83
- PIC16F84
- PIC16CR84

Pin Diagram

PDIP, SOIC



PROGRAMMING THE PIC16F8X

The PIC16F8X is programmed using a serial method. The serial mode will allow the PIC16F8X to be programmed while in the users system. This allows for increased design flexibility. This programming specification applies to PIC16F8X devices in all packages.

Hardware Requirements

The PIC16F8X requires 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.

Programming Mode

The programming mode for the PIC16F8X allows programming of user program memory, data memory, special locations used for ID, and the configuration word.

PIN DESCRIPTIONS (DURING PROGRAMMING): PIC16F8X

Pin Name	During Programming		
	Pin Name	Pin Type	Pin Description
RB6	CLOCK	I	Clock input
RB7	DATA	I/O	Data input/output
$\overline{MCLR}$	VTEST MODE	P*	Program Mode Select
VDD	VDD	P	Power Supply
VSS	VSS	P	Ground

Legend: I = Input, O = Output, P = Power

*In the PIC16F8X, the programming high voltage is internally generated. To activate the programming mode, high voltage needs to be applied to $\overline{MCLR}$ input. Since the $\overline{MCLR}$ is used for a level source, this means that $\overline{MCLR}$ does not draw any significant current.

PROGRAM MODE ENTRY

User Program Memory Map

The user memory space extends from 0x0000 to 0x1FFF (8K), of which 1K (0x0000 - 0x03FF) 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 0x3FF, it will wrap around and address a location within the physically implemented memory. (See Figure 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 .

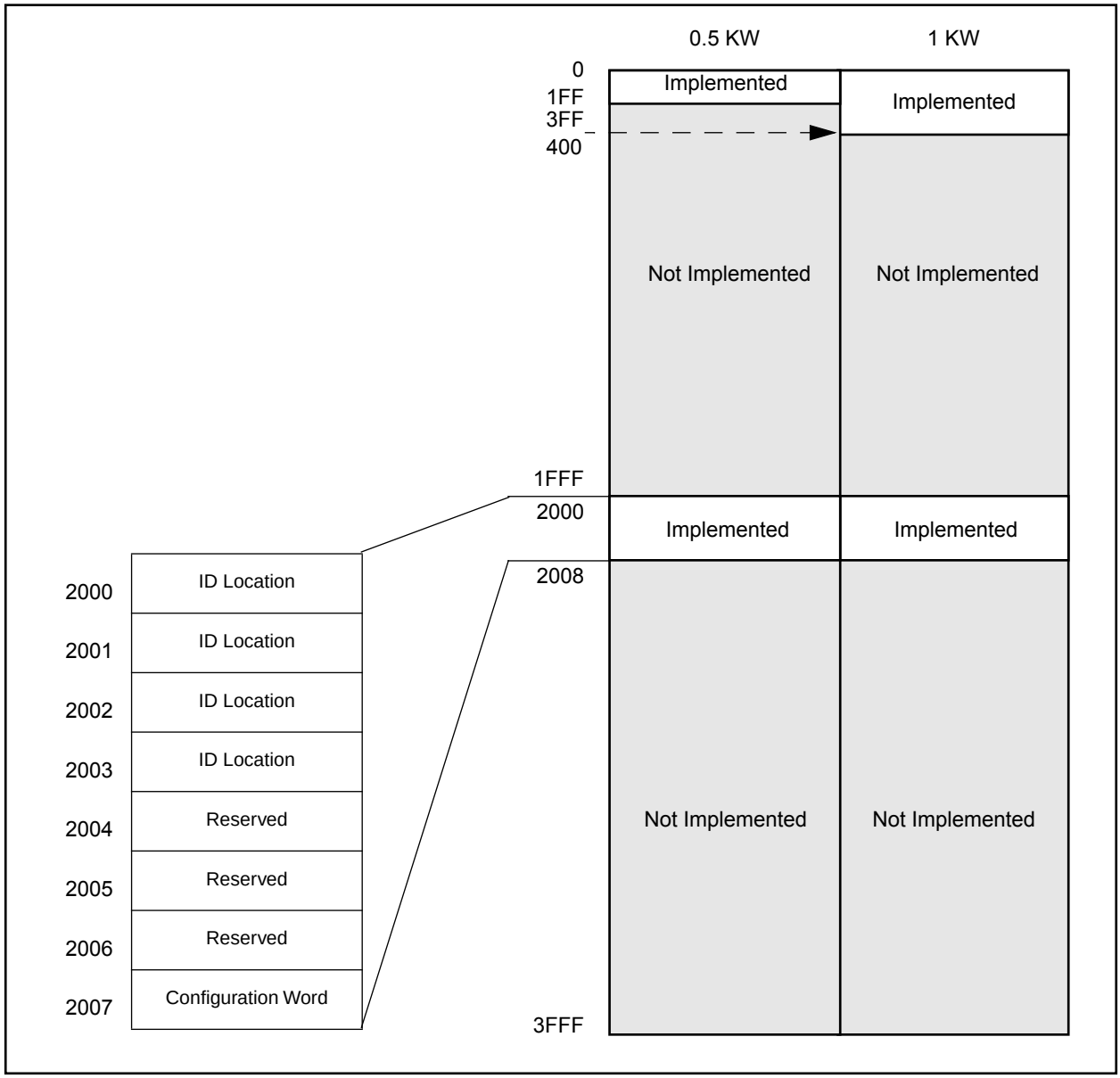
In the configuration memory space, 0x2000-0x200F are physically implemented. However, only locations 0x2000 through 0x2007 are available. Other locations are reserved. Locations beyond 0x200F will physically access user memory. (See Figure 1).

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 3.

FIGURE 1: PROGRAM MEMORY MAPPING



Program/Verify Mode

The program/verify mode is entered by holding pins RB6 and RB7 low while raising $\overline{MCLR}$ pin from V_{IL} to V_{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 and RB7 are Schmitt Trigger Inputs in this mode.

Note:	The OSC must not have 72 osc clocks while the device $\overline{MCLR}$ is between V_{IL} and V_{IHH}.

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

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 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 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 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 1 µs is required between a command and a data word (or another command).

The commands that are available are:

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 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 $\overline{MCLR}$ low (V_{IL}).

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.

TABLE 1: COMMAND MAPPING

Command	Mapping (MSB ... LSB)						Data
Load Configuration	0	0	0	0	0	0	0, data (14), 0
Load Data for Program Memory	0	0	0	0	1	0	0, data (14), 0
Read Data from Program Memory	0	0	0	1	0	0	0, data (14), 0
Increment Address	0	0	0	1	1	0	
Begin Programming	0	0	1	0	0	0	
Load Data for Data Memory	0	0	0	0	1	1	0, data (14), 0
Read Data from Data Memory	0	0	0	1	0	1	0, data (14), 0
Bulk Erase Program Memory	0	0	1	0	0	1	
Bulk Erase Data Memory	0	0	1	0	1	1	

FIGURE 2: PROGRAM FLOW CHART - PIC16F8X PROGRAM MEMORY

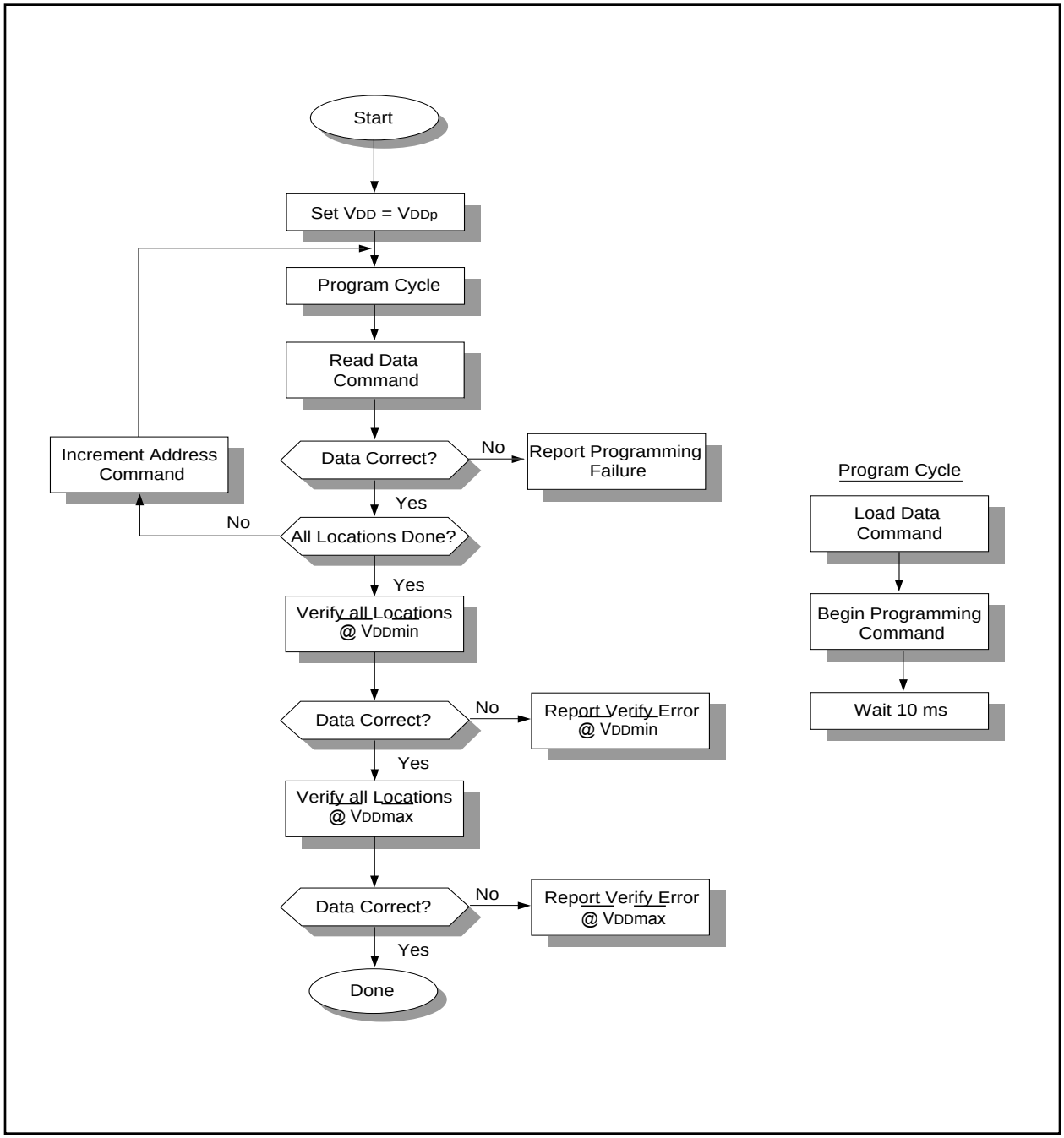
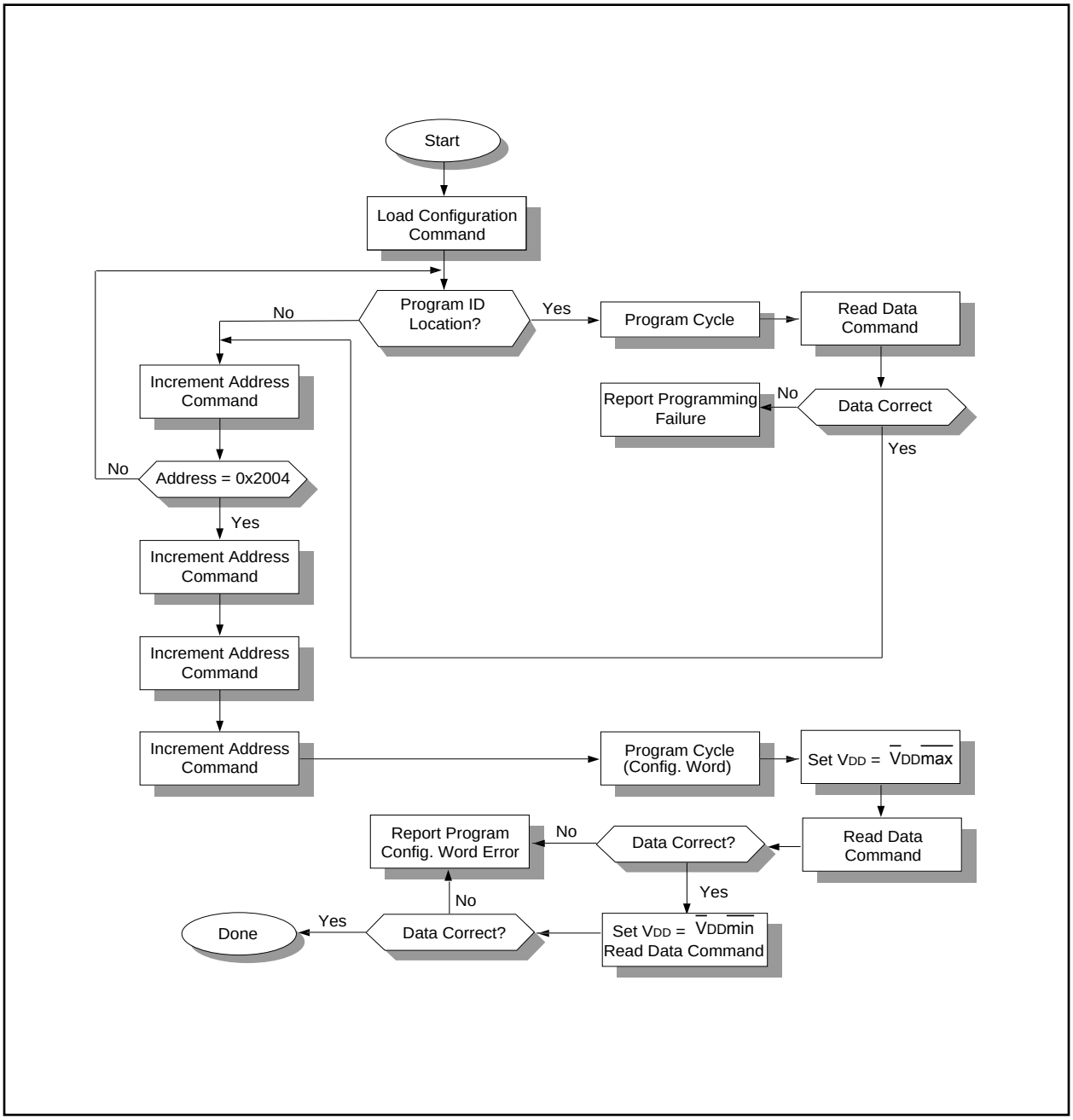


FIGURE 3: PROGRAM FLOW CHART - PIC16F8X CONFIGURATION MEMORY



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.

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.

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.

Increment Address

The PC is incremented when this command is received. A timing diagram of this command is shown in Figure 7.

Begin Programming

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 10ms for programming to complete. No "end programming" command is required.

Bulk Erase Program Memory

To perform a bulk erase of the program memory, the following sequence must be performed.

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 test program memory (0x2000 - 0x200F), then both the user memory and the test memory will be erased. The configuration word will not be erased, even if the address is pointing to location 0x2007.

Bulk Erase Data Memory

To perform a bulk erase of the data memory, the following sequence must be performed.

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.

Programming Algorithm Requires Variable V_{DD}

The PIC16F8X uses an intelligent algorithm. The algorithm calls for program verification at V_{DD}min. as well as V_{DD}max. Verification at V_{DD}min. guarantees good "erase margin". Verification at V_{DD}max guarantees good "program margin".

The actual programming must be done with V_{DD} in the V_{DDP} range (See Table 4).

V_{DDP} = V_{CC} range required during programming.

V_{DD}min. = minimum operating V_{DD} spec for the part.

V_{DD}max. = maximum operating V_{DD} spec for the part.

Programmers must verify the PIC16F8X at its specified V_{DD} max. and V_{DD}min levels. Since Microchip may introduce future versions of the PIC16F8X with a broader V_{DD} 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.

CONFIGURATION WORD

The PIC16F8X has five configuration bits. These bits can be set (reads '0') or left unchanged (reads '1') to select various device configurations.

FIGURE 4: CONFIGURATION WORD BIT MAP

Bit Number:	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PIC16F83/F84	CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	$\overline{\text{PWRTE}}$	WDTE	FOSC1	FOSC0
PIC16CR83/CR84	CP	CP	CP	CP	CP	CP	DP	CP	CP	CP	$\overline{\text{PWRTE}}$	WDTE	FOSC1	FOSC0

bit 4-13: **CP,** Code Protection Configuration Bit
1 = code protection off
0 = code protection on

bit 7: **PIC16CR83/CR84 only**
DP, Data Memory Code Protection Bit
1 = code protection off
0 = data memory is code protected

bit 3: **$\overline{\text{PWRTE}}$**, Power Up Timer Enable Configuration Bit
1 = Power up timer disabled
0 = Power up timer enabled

bit 3-2: **$\overline{\text{WDTE}}$**, WDT Enable Configuration Bits
1 = WDT enabled
0 = WDT disabled

bit 1-0 **FOSC<1:0>**, Oscillator Selection Configuration Bits
11: RC oscillator
10: HS oscillator
01: XT oscillator
00: LP oscillator

CODE PROTECTION

For PIC16F8X devices, once code protection is enabled, all program 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.

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:

a) Execute load configuration (with a '1' in bit 4, code protect).
b) Increment to configuration word location (0x2007)
c) Execute command (000001)
d) Execute command (000111)
e) Execute 'Begin Programming' (001000)
f) Wait 10ms
g) Execute command (000001)
h) Execute command (000111)

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 PIC16F8X, the EEPROM data memory should also be embedded in the hex file (see Section).

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

TABLE 2: CONFIGURATION WORD

PIC16F83

To code protect: `000000000XXX`

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0x2007)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
All memory	Read All 0's, Write Disabled	Read Unscrambled, Write Enabled
ID Locations [0x2000 : 0x2003]	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled

PIC16CR83

To code protect: `000000000XXX`

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0x2007)	Read Unscrambled	Read Unscrambled
All memory	Read All 0's for Program Memory, Read All 1's for Data Memory - Write Disabled	Read Unscrambled, Data Memory - Write Enabled
ID Locations [0x2000 : 0x2003]	Read Unscrambled	Read Unscrambled

PIC16CR84

To code protect: 000000000XXX

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0x2007)	Read Unscrambled	Read Unscrambled
All memory	Read All 0's for Program Memory, Read All 1's for Data Memory - Write Disabled	Read Unscrambled, Data Memory - Write Enabled
ID Locations [0x2000 : 0x2003]	Read Unscrambled	Read Unscrambled

PIC16F84

To code protect: 000000000XXX

Program Memory Segment	R/W in Protected Mode	R/W in Unprotected Mode
Configuration Word (0x2007)	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled
All memory	Read All 0's, Write Disabled	Read Unscrambled, Write Enabled
ID Locations [0x2000 : 0x2003]	Read Unscrambled, Write Enabled	Read Unscrambled, Write Enabled

Legend: X = Don't care

CHECKSUM COMPUTATION

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., 0x1FF for the PIC16F8X. 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 3.

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: CHECKSUM COMPUTATION

Device	Code Protect	Checksum*	Blank Value	0x25E6 at 0 and max address
PIC16F83	OFF ON	SUM[0x000:0x1FF] + CFGW & 0x3FFF CFGW & 0x3FFF + SUM_ID	0x3DFF 0x3E0E	0x09CD 0x09DC
PIC16CR83	OFF ON	SUM[0x000:0x1FF] + CFGW & 0x3FFF CFGW & 0x3FFF + SUM_ID	0x3DFF 0x3E0E	0x09CD 0x09DC
PIC16F84	OFF ON	SUM[0x000:0x3FF] + CFGW & 0x3FFF CFGW & 0x3FFF + SUM_ID	0x3BFF 0x3C0E	0x07CD 0x07DC
PIC16CR84	OFF ON	SUM[0x000:0x3FF] + CFGW & 0x3FFF CFGW & 0x3FFF + SUM_ID	0x3BFF 0x3C0E	0x07CD 0x07DC

Legend: CFGW = Configuration Word
SUM[a:b] = [Sum of locations a to b inclusive]
*Checksum = [Sum of all the individual expressions] **MODULO** [0xFFFF]
+ = Addition
& = Bitwise AND

PROGRAM/VERIFY MODE ELECTRICAL CHARACTERISTICS

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 0x2100. The format for data memory storage is one data byte per address location, lsb aligned.

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

Standard Operating Conditions							
Operating Temperature: +10°C ≤ TA ≤ +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/Comments**
	VDDP	Supply voltage during programming	4.5	5.0	5.5	V	
	VDDV	Supply voltage during verify	VDDMIN		VDDMAX	V	Note 1
	VIHH	High voltage on $\overline{MCLR}$ for test mode entry	12		14.0	V	Note 2
	IDDP	Supply current (from VDD) during program/verify			50	mA	
	IHH	Supply current from VIHH (on $\overline{MCLR}$)			200	mA	
	VIH1	(RB6, RB7) input high level	0.8 VDD			V	Schmitt Trigger input
	VIL1	(RB6, RB7) input low level $\overline{MCLR}$ (test mode selection)	0.2 VDD			V	Schmitt Trigger input
P1	TVHHR	$\overline{MCLR}$ rise time (VSS to VHH) for test mode entry			8.0	ms	
P2	TSET0	RB6, RB7 setup time (before pattern setup time)	100			ns	
P3	TSET1	Data in setup time before clock ↓	100			ns	
P4	THLD1	Data in hold time after clock ↓	100			ns	
P5	TDLY1	Data input not driven to next clock input (delay required between command/data or command/command)	1.0			ms	
P6	TDLY2	Delay between clock ↓ to clock ↑ of next command or data	1.0			ms	
P7	TDLY3	Clock to data out valid (during read data)	80			ns	
P8	THLD0	RB <7:6> hold time after $\overline{MCLR}$ ↑	100			ns	

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 5: LOAD DATA COMMAND (PROGRAM/VERIFY)

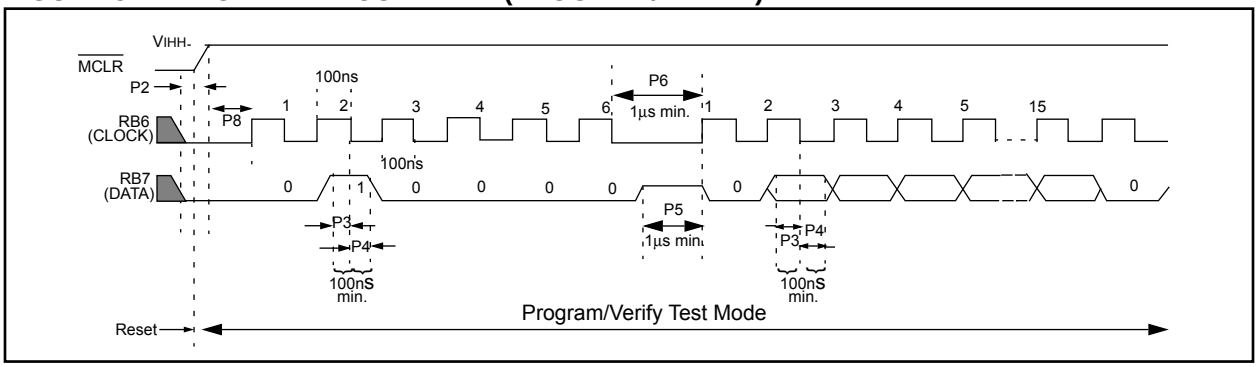


FIGURE 6: READ DATA COMMAND (PROGRAM/VERIFY)

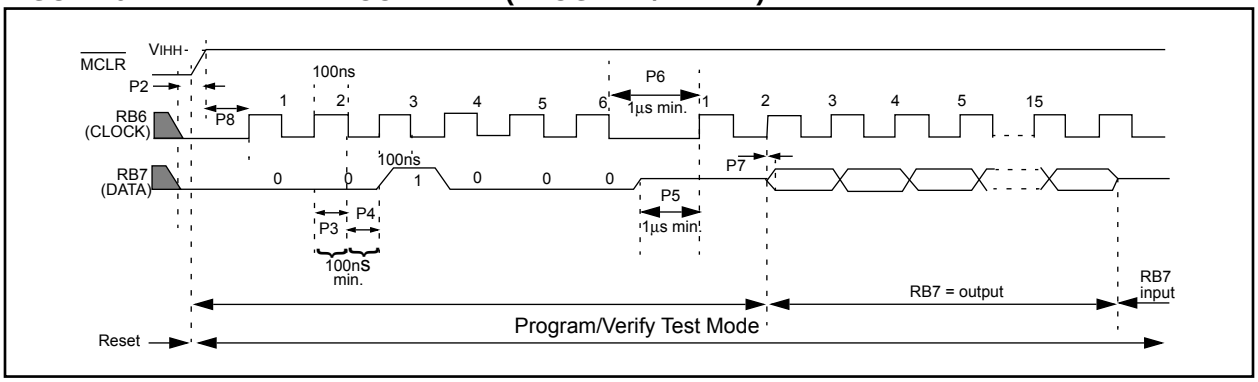
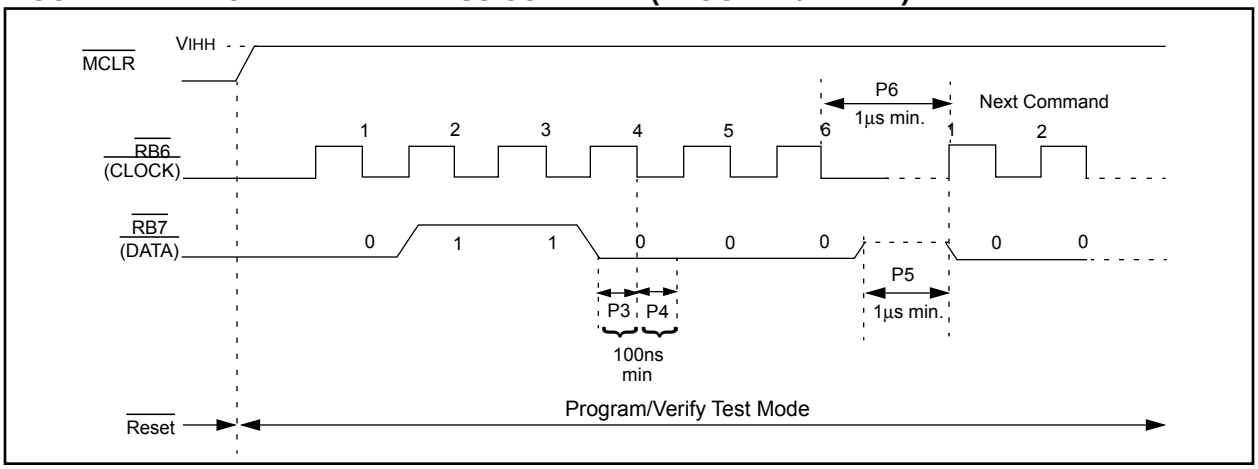


FIGURE 7: INCREMENT ADDRESS COMMAND (PROGRAM/VERIFY)



NOTES:

SECTION 4
IN-CIRCUIT SERIAL PROGRAMMING (ICSP™) APPLICATION NOTE

PICmicro is a trademark of Microchip Technology, Inc.

AN656

In-Circuit Serial Programming (ICSP™) of Calibration Parameters Using a PICmicro™ Microcontroller

Author: John Day
Microchip Technology Inc.

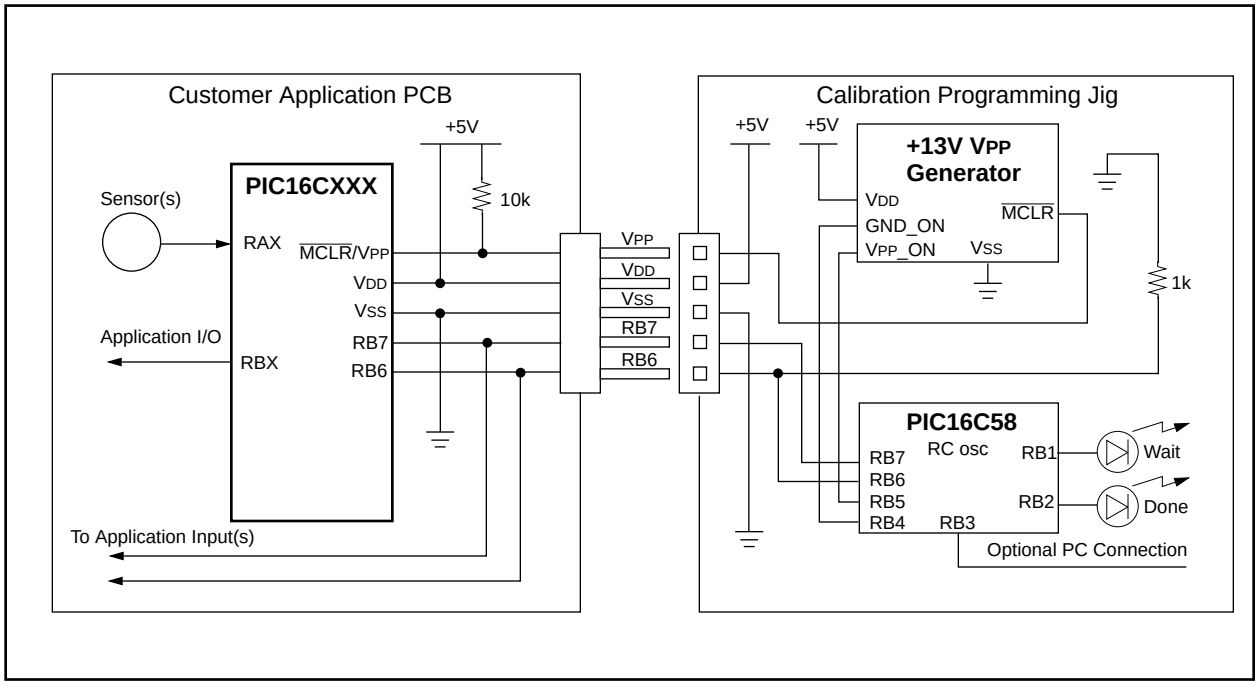
INTRODUCTION

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 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 re-usable 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:

- **$\overline{\text{MCLR}}$/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 it's 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 determing 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, 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 2.

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 3.

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 2:

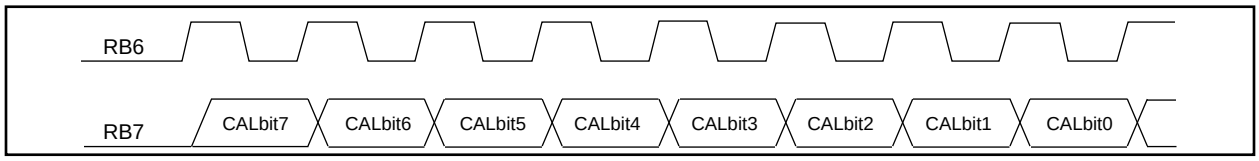
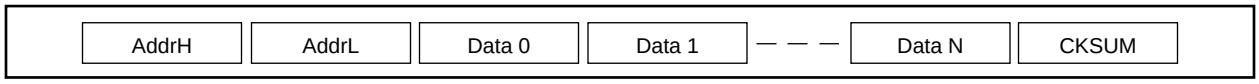


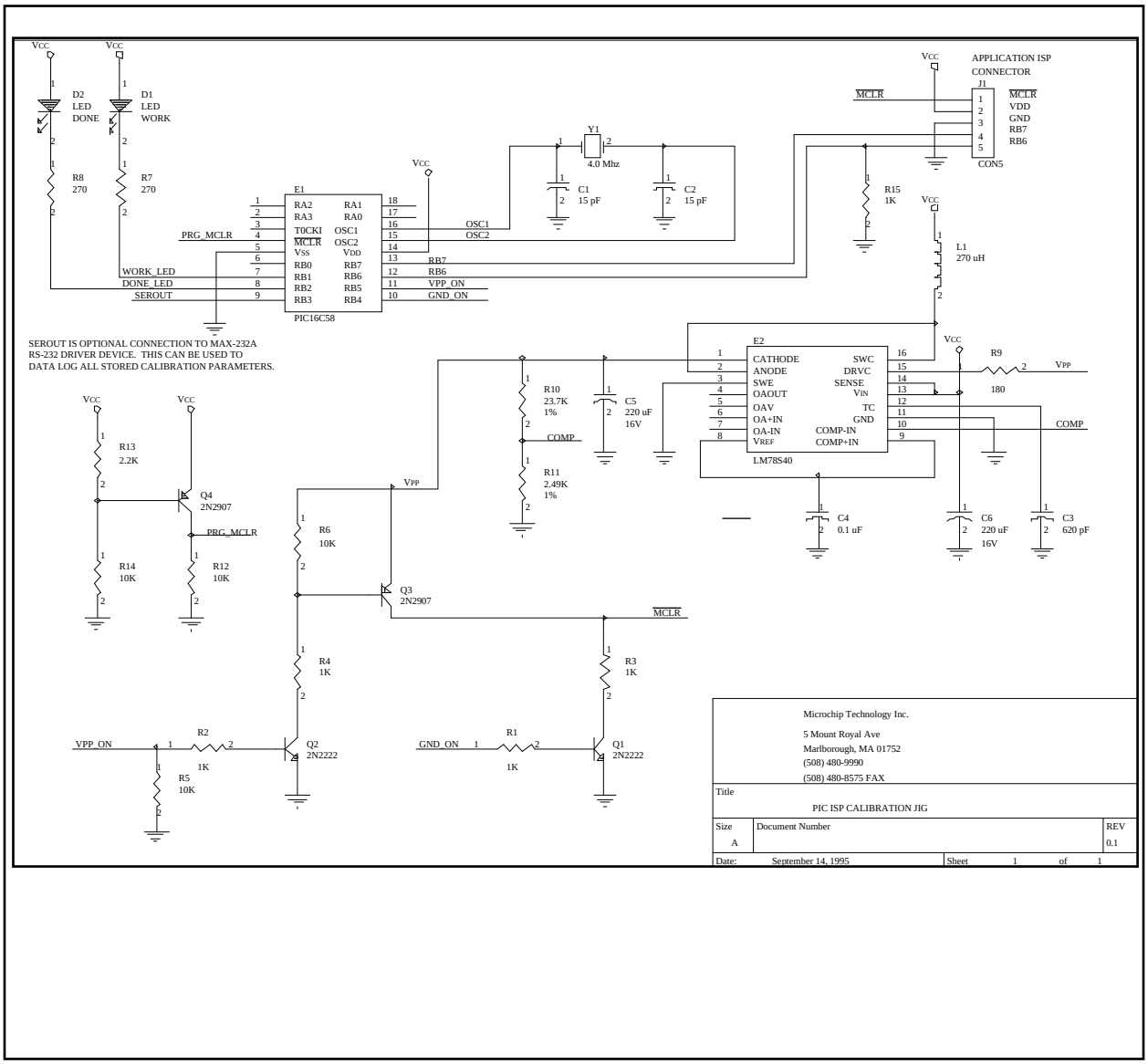
FIGURE 3:



LED Operation

When the programming jig is waiting for communication from the application PICmicro, 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 4: 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

Bill of Material			
Item	**Quantity**	**Reference**	**Part**
1	2	C1,C2	15 pF
2	1	C3	620 pF
3	1	C4	0.1 mF
4	2	C5,C6	220 mF
5	2	D1,D2	LED
6	1	E1	PIC16C58
7	1	E2	LM78S40
8	1	J1	CON5
9	1	L1	270 mH
10	2	Q1,Q2	2N2222
11	2	Q3,Q4	2N2907
12	5	R1,R2,R3,R4,R15	1k
13	4	R5,R6,R12,R14	10k
14	2	R7,R8	270
15	1	R9	180
16	1	R10	23.7k
17	1	R11	2.49k
18	1	R13	2.2k
19	1	Y1	4.0 MHz

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               *
                     00010 ; ********************************************
                     00011 ; * Include files:                           *
                     00012 ; *           P16C5X.ASM                     *
                     00013 ; ********************************************
                     00014 ; * Fuses:    OSC:  XT (4.0 Mhz xtal)        *
                     00015 ; *           WDT:  OFF                      *
                     00016 ; *            CP:  OFF                      *
                     00017
                     ;*******************************************************************************
                     00018 ; This program is intended to be used as a self programmer
                     00019 ; to store calibration constants into a lookup table
                     00020 ; within the main system processor.  A 4 Mhz crystal
                     00021 ; is needed and an optional 9600 baud seiral port will
                     00022 ; display the parameters to be programmed.
                     00023 ;
                     ;*******************************************************************************
                     00024 ; * Program Memory:                                                      *
                     00025 ; *    Words - communication with test jig                              *
                     00026 ; *    17 Words - calibration look-up table (16 bytes of data)          *
                     00027 ; *    13 Words - Test Code to generate Calibration Constants           *
                     00028 ; * RAM memory:                                                         *
                     00029 ; *    64 Bytes - Store up to 64 bytes of calibration constant          *
                     00030 ; *     9 Bytes - Store 9 bytes of temp variables (reused)              *
                     00031 ;
                     ;*******************************************************************************
                     00032
                     00033     list p=16C58A
                     00034     include <p16C5x.inc>
                     00001         LIST
                     00002 ; P16C5X.INC Standard Hdr File, Version 3.30 Microchip Technology, Inc.
                     00224         LIST
0FFF 0FF9            00035     __CONFIG _CP_OFF&_WDT_OFF&_XT_OSC
                     00036
                     00037 ; ***********************************
                     00038 ; * Port A (RA0-RA4) bit definitions *
                     00039 ; ***********************************
                     00040 ; No PORT A pins are used in this design
                     00041
                     00042 ; ***********************************
                     00043 ; * Port B (RB0-RB7) bit definitions *
                     00044 ; ***********************************
  00000006           00045 ISPCLOCK      EQU 6   ; Clock line for ISP and parameter comm
  00000007           00046 ISPDATA       EQU 7   ; Data line for ISP and parameter comm
  00000005           00047 VPPON         EQU 5   ; Apply +13V VPP voltage to MCLR (test mode)
  00000004           00048 GNDON         EQU 4   ; Apply +0V (gnd) voltage to MCLR (reset)
  00000003           00049 SEROUT        EQU 3   ; Optional RS-232 TX output (needs 12V driver)
  00000002           00050 DONELED       EQU 2   ; Turns on LED when done sucessfully program
  00000001           00051 WORKLED       EQU 1   ; On during programming, off when done
                     00052                       ; RB0 is not used in this design
                     00053
```

```
                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
                00071 ; ***
                00072 ; *** The following VARS are used to receive and store CAL params:
                00073 ; ***
00000007        00074 COUNT          EQU 07h  ; Counter var used to receive cal params
00000008        00075 TEMP1          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 0Eh  ; Pointer to next byte of CAL storage
0000000F        00081 BYTECOUNT      EQU 0Fh  ; Number of CAL bytes received
                00082
                00083 ; ************************************
                00084 ; * Various constants used in program *
                00085 ; ************************************
00000001        00086 DATISPOUT      EQU b'00000001'  ; tris settings for ISP data out
00000081        00087 DATISPIN       EQU b'10000001'  ; tris settings for ISP data in
00000006        00088 CMDISPCNT      EQU 6            ; Number of bits for ISP command
00000010        00089 STARTCALBYTE   EQU 10h          ; Address in RAM where CAL byte data stored
00000007        00090 VFYYES         EQU PA2          ; Flag bit enables verification (STATUS)
00000006        00091 CMDISPINCRADDR EQU b'00000110'  ; ISP Pattern to increment address
00000008        00092 CMDISPPGMSTART EQU b'00001000'  ; ISP Pattern to start programming
0000000E        00093 CMDISPPGMEND   EQU b'00001110'  ; ISP Pattern to end programming
00000002        00094 CMDISPLOAD     EQU b'00000010'  ; ISP Pattern to load data for program
00000004        00095 CMDISPREAD     EQU b'00000100'  ; ISP Pattern to read data for verify
00000034        00096 UPPER6BITS     EQU 034h         ; Upper 6 bits for retlw instruction
                00097
                00098 ; ************************************
                00099 ; * 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 dlylabels
                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 dlylabels
                00112     decfsz  COUNT,F        ; Decrement count until done
                00113     goto    dlylabels      ; Not done delaying - go back!
                00114     ENDM                   ; Done with Macro
                00115
                00116 ; ********************************************
                00117 ; * addrtofsr 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 ; ************************************
07FF            00146     ORG     7FFh
Message[306]: Crossing page boundary -- ensure page bits are set.
07FF 0A00       00147     goto    start
                00148
                00149 ; ************************************
                00150 ; * Start of CAL param read routine    *
                00151 ; ************************************
0000            00152     ORG     0h
0000            00153 start
0000 0C0A       00154     movlw   b'00001010'   ; Serial OFF, LEDS OFF, VPP OFF
0001 0026       00155     movwf   PORTB         ; Place "0" into port b latch register
0002 0CC1       00156     movlw   b'11000001'   ; RB7;:RB6, RB0 set to inputs
0003 0006       00157     tris    PORTB         ; Move to tris registers
0004 0040       00158     clrw                  ; Place 0 into W
0005 0065       00159     clrf    PORTA         ; Place all ZERO into latch
0006 0005       00160     tris    PORTA         ; Make all pins outputs to be safe..
0007 0586       00161     bsf     PORTB,GNDON   ; TEST ONLY-RESET PIC-NOT NEEDED IN REAL DESIGN!
0008            00162 clearram
0008 0C10       00163     movlw   010h          ; Place start of buffer into W
0009 0027       00164     movwf   COUNT         ; Use count for RAM pointer
000A            00165 loopclrram
                00166     addrtofsr COUNT       ; Set up FSR
000A 0C10           M     movlw   STARTCALBYTE  ; Place base address into W
000B 0087           M     subwf   COUNT,w       ; Offset by STARTCALBYTE
000C 0024           M     movwf   FSR           ; Place into FSR
000D 06A4           M     btfsc   FSR,5         ; Shift bits 4,5 to 5,6
000E 05C4           M     bsf     FSR,6
000F 04A4           M     bcf     FSR,5
0010 0684           M     btfsc   FSR,4
0011 05A4           M     bsf     FSR,5
0012 0584           M     bsf     FSR,4
0013 0060       00167     clrf    INDF           ; Clear buffer value
0014 02A7       00168     incf    COUNT,F        ; Move to next reg
0015 0C50       00169     movlw   050h           ; Move end of buffer addr to W
0016 0087       00170     subwf   COUNT,W        ; Check if at last MEM
0017 0743       00171     btfss   STATUS,Z       ; Skip when at end of counter
0018 0A0A       00172     goto    loopclrram     ; go back to next location
0019 0486       00173     bcf     PORTB,GNDON    ; TEST ONLY-LET IT GO-NOT NEEDED IN REAL DESIGN!
001A            00174 calget
001A 006A       00175     clrf    CSUMTOTAL      ; Clear checksum total byte
```

```
001B 0069      00176     clrf    DATAREG             ; Clear out data receive register
001C 0C10      00177     movlw   STARTCALBYTE        ; Place RAM start address of first cal byte
001D 002E      00178     movwf   ADDRPTR             ; Place this into ADDRPTR
001E           00179 waitclockpulse
001E 07C6      00180     btfss   PORTB,ISPCLOCK      ; Wait for CLOCK high pulse - skip when high
001F 0A1E      00181     goto    waitclockpulse      ; CLOCK is low - go back and wait!
0020           00182 loopcal
0020 0C08      00183     movlw   .8                  ; Place 8 into W (8 bits/byte)
0021 0027      00184     movwf   COUNT               ; set up counter register to count bits
0022           00185 loopsendcal
0022 006B      00186     clrf    TIMEHIGH            ; Clear timeout counter for high pulse
0023 006C      00187     clrf    TIMELOW             ; Clear timeout counter for low pulse
0024           00188 waitclkhi
0024 06C6      00189     btfsc   PORTB,ISPCLOCK      ; Wait for CLOCK high - skip if it is low
0025 0A29      00190     goto    waitclklo           ; Jump to wait for CLOCK low state
0026 02EB      00191     decfsz  TIMEHIGH,F          ; Decrement counter - skip if timeout
0027 0A24      00192     goto    waitclkhi           ; Jump back and wait for CLOCK high again
0028 0A47      00193     goto    timeout             ; Timed out waiting for high - check data!
0029           00194 waitclklo
0029 07C6      00195     btfss   PORTB,ISPCLOCK      ; Wait for CLOCK low - skip if it is high
002A 0A2E      00196     goto    clockok             ; Got a high to low pulse - jump to clockok
002B 02EC      00197     decfsz  TIMELOW,F           ; Decrement counter - skip if timeout
002C 0A29      00198     goto    waitclklo           ; Jump back and wait for CLOCK low again
002D 0A47      00199     goto    timeout             ; Timed out waiting for low - check data!
002E           00200 clockok
002E 0C08      00201     movlw   .8                  ; Place initial count value into W
002F 0087      00202     subwf   COUNT,W             ; Subtract from count, place into W
0030 0743      00203     btfss   STATUS,Z            ; Skip if we are at count 8 (first value)
0031 0A34      00204     goto    skipcsumadd         ; Skip checksum add if any other count value
0032 0209      00205     movf    DATAREG,W           ; Place last byte received into W
0033 01EA      00206     addwf   CSUMTOTAL,F         ; Add to checksum
0034           00207 skipcsumadd
0034 0503      00208     bsf     STATUS,C            ; Assume data bit is high
0035 07E6      00209     btfss   PORTB,ISPDATA       ; Skip if the data bit was high
0036 0403      00210     bcf     STATUS,C            ; Set data bit to low
0037 0369      00211     rlf     DATAREG,F           ; Rotate next bit into DATAREG
0038 02E7      00212     decfsz  COUNT,F             ; Skip after 8 bits
0039 0A22      00213     goto    loopsendcal         ; Jump back and send next bit
               00214     addrtofsr ADDRPTR           ; Convert pointer address to FSR
003A 0C10          M     movlw   STARTCALBYTE        ; Place base address into W
003B 008E          M     subwf   ADDRPTR,w           ; Offset by STARTCALBYTE
003C 0024          M     movwf   FSR                 ; Place into FSR
003D 06A4          M     btfsc   FSR,5               ; Shift bits 4,5 to 5,6
003E 05C4          M     bsf     FSR,6
003F 04A4          M     bcf     FSR,5
0040 0684          M     btfsc   FSR,4
0041 05A4          M     bsf     FSR,5
0042 0584          M     bsf     FSR,4
0043 0209      00215     movf    DATAREG,W           ; Place received byte into W
0044 0020      00216     movwf   INDF                ; Move recv'd byte into CAL buffer location
0045 02AE      00217     incf    ADDRPTR,F           ; Move to the next cal byte
0046 0A20      00218     goto    loopcal             ; Go back for next byte
0047           00219 timeout
0047 0C14      00220     movlw   STARTCALBYTE+4      ; check if we received (4) params
0048 008E      00221     subwf   ADDRPTR,W           ; Move current address pointer to W
0049 0703      00222     btfss   STATUS,C            ; Skip if we have at least (4)
004A 0A93      00223     goto    sendnoise           ; not enough params - print and RESET!
004B 0200      00224     movf    INDF,W              ; Move received checksum into W
004C 00AA      00225     subwf   CSUMTOTAL,F    ; Subtract received Checksum from calc'd checksum
004D 0743      00226     btfss   STATUS,Z            ; Skip if CSUM OK
004E 0A9F      00227     goto    sendcsumbad         ; Checksum bad - print and RESET!
004F           00228 csumok
004F 0426      00229     bcf     PORTB,WORKLED       ; Turn on WORK LED
0050 0C10      00230     movlw   STARTCALBYTE        ; Place start pointer into W
0051 008E      00231     subwf   ADDRPTR,W           ; Subtract from current address
0052 002F      00232     movwf   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     addrtofsr ADDRPTR          ; Set up FSR
0056 0C10          M     movlw   STARTCALBYTE       ; Place base address into W
0057 008E          M     subwf   ADDRPTR,w          ; Offset by STARTCALBYTE
0058 0024          M     movwf   FSR                ; Place into FSR
0059 06A4          M     btfsc   FSR,5              ; Shift bits 4,5 to 5,6
005A 05C4          M     bsf     FSR,6
005B 04A4          M     bcf     FSR,5
005C 0684          M     btfsc   FSR,4
005D 05A4          M     bsf     FSR,5
005E 0584          M     bsf     FSR,4
005F 0380      00238     swapf   INDF,W             ; Place received char into W
0060 0E0F      00239     andlw   0Fh                ; 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 0E0F      00247     andlw  0Fh                 ; Strip off upper digits
0068 002D      00248     movwf   TEMP               ; Place into TEMP
0069 0C30      00249     movlw   '0'                ; Place ASCII '0' into W
006A 01CD      00250     addwf   TEMP,w             ; Add to TEMP, place into W
006B 09AE      00251     call    putchar            ; Send out char
006C 0A73      00252     goto    printlo            ; Jump to print next char
006D           00253 printhiletter
006D 0380      00254     swapf   INDF,W             ; Place received char into W
006E 0E0F      00255     andlw  0Fh                 ; 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     movf    INDF,W             ; Place received char into W
0074 0E0F      00262     andlw   0Fh                ; 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 printlonumber
007A 0200      00269     movf    INDF,W             ; Place received char into W
007B 0E0F      00270     andlw   0Fh                ; Strip off upper digits
007C 002D      00271     movwf   TEMP               ; Place into TEMP
007D 0C30      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    INDF,W             ; Place received char into W
0082 0E0F      00278     andlw   0Fh                ; 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    ADDRPTR,W          ; Go to next buffer value
008A 0E0F      00287     andlw   0Fh                ; 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    ADDRPTR,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
Message[306]: Crossing page boundary -- ensure page bits are set.
0092 0A6B      00295     goto    programpartisp      ; Go to program part through ISP
0093           00296 sendnoise
0093 0C4E      00297     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           00309 sendcsumbad
009F 0C43      00310     movlw   'C'                 ; Place 'C' into W
00A0 09AE      00311     call    putchar             ; Send char in W to terminal
00A1 0C53      00312     movlw   'S'                 ; Place 'S' into W
00A2 09AE      00313     call    putchar             ; Send char in W to terminal
00A3 0C55      00314     movlw   'U'                 ; Place 'U' into W
00A4 09AE      00315     call    putchar             ; Send char in W to terminal
00A5 0C4D      00316     movlw   'M'                 ; Place 'M' into W
00A6 09AE      00317     call    putchar             ; Send char in W to terminal
00A7 09A9      00318     call    printcrlf           ; Print CR and LF
00A8 0A1A      00319     goto    calget              ; RESET!
               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           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           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           00348 putloop
00B3 0329      00349     rrf     DATAREG,F           ; Place next bit into carry
00B4           00350 putloop1
00B4 0703      00351     btfss   STATUS,C            ; Skip if carry was set
00B5 0466      00352     bcf     PORTB,SEROUT        ; Clear RS-232 serial output bit
00B6 0603      00353     btfsc   STATUS,C            ; Skip if carry was clear
```

```
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      00356     decfsz  TEMP1,F             ; Decrement bit counter, skip when done!
00BE 0AB3      00357     goto    putloop             ; Jump back and send next bit
00BF 0566      00358     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      00360     retlw   0                   ; Done - RETURN
               00361
               00362 ; *********************************************************************
               00363 ; *  ISP routines from PICSTART-16C                                   *
               00364 ; *  Converted from PIC17C42 to PIC16C5X code by John Day             *
               00365 ; *  Originially written by Jim Pepping                              *
               00366 ; *********************************************************************
0200           00367     ORG 200                     ; ISP routines stored on page 1
               00368
               00369 ; *********************************************************************
               00370 ; * poweroffisp                                                      *
               00371 ; * Power off application PIC - turn off VPP and reset device after *
               00372 ; * programming pass is complete                                     *
               00373 ; *********************************************************************
0200           00374 poweroffisp
0200 04A6      00375     bcf     PORTB,VPPON         ; Turn off VPP 13 volts
0201 0586      00376     bsf     PORTB,GNDON         ; Apply 0 V to MCLR to reset PIC
0202 0CC1      00377     movlw   b'11000001'         ; RB6,7 set to inputs
0203 0006      00378     tris    PORTB               ; Move to tris registers
0204 0486      00379     bcf     PORTB,GNDON   ; Allow MCLR to go back to 5 volts, deassert reset
0205 0526      00380     bsf     PORTB,WORKLED       ; Turn off WORK LED
0206 0800      00381     retlw 0                     ; Done so return!
               00382
               00383 ; *********************************************************************
               00384 ; * testmodeisp                                                      *
               00385 ; * Apply VPP voltage to place application PIC into test mode.      *
               00386 ; * this enables ISP programming to proceed                         *
               00387 ; *     RAM used:       TEMP                                        *
               00388 ; *********************************************************************
0207           00389 testmodeisp
0207 0C08      00390     movlw   b'00001000'         ; Serial OFF, LEDS OFF, VPP OFF
0208 0026      00391     movwf   PORTB               ; Place "0" into port b latch register
0209 04A6      00392     bcf     PORTB,VPPON         ; Turn off VPP just in case!
020A 0586      00393     bsf     PORTB,GNDON         ; Apply 0 volts to MCLR
020B 0C01      00394     movlw   b'00000001'         ; RB6,7 set to outputs
020C 0006      00395     tris    PORTB               ; Move to tris registers
020D 0206      00396     movf    PORTB,W             ; Place PORT B state into W
020E 002D      00397     movwf   TEMP                ; Move state to TEMP
020F 048D      00398     bcf     TEMP,4              ; Turn off MCLR GND
0210 05AD      00399     bsf     TEMP,5              ; Turn on VPP voltage
0211 020D      00400     movf    TEMP,W              ; Place TEMP into W
0212 0026      00401     movwf   PORTB               ; Turn OFF GND and ON VPP
```

```
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 05C6      00420     bsf     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 04E6      00424     bcf     PORTB,ISPDATA     ; Clear DATA bit to start (assume 0)
0220 0329      00425     rrf     HIBYTE,F          ; Rotate HIBYTE output
0221 032A      00426     rrf     LOBYTE,F          ; Rotate LOBYTE output
0222 0603      00427     btfsc   STATUS,C          ; Skip if data bit is zero
0223 05E6      00428     bsf     PORTB,ISPDATA     ; Set DATA line to send a one
0224 05C6      00429     bsf     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     bsf     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 (6 MSBs only) and LOBYTE.                           *
               00443 ; *     RAM used:       TEMP, W, HIBYTE (output), LOBYTE (output)   *
               00444 ; *******************************************************************
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     bsf     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     bsf     PORTB,ISPCLOCK    ; Set CLOCK bit
0238 0000      00459     nop                       ; Wait one cycle
0239 0403      00460     bcf     STATUS,C          ; Clear carry bit, assume 0 read
023A 06E6      00461     btfsc   PORTB,ISPDATA     ; Check the data, skip if it was zero
023B 0503      00462     bsf     STATUS,C          ; Set carry bit if data was one
023C 0329      00463     rrf     HIBYTE,F          ; Move recevied bit into HIBYTE
023D 032A      00464     rrf     LOBYTE,F          ; Update LOBYTE
023E 04C6      00465     bcf     PORTB,ISPCLOCK    ; Clear CLOCK line
023F 0000      00466     nop                       ; Wait one cycle
0240 0000      00467     nop                       ; Wait one cycle
```

```
0241 02ED      00468     decfsz  TEMP,F              ; Decrement bit counter, skip when zero
0242 0A37      00469     goto    P16cispinloop       ; 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           00492 commandisp
0252 002A      00493     movwf   LOBYTE              ; Place command into LOBYTE
0253 0C06      00494     movlw   CMDISPCNT           ; Place number of command bits into W
0254 002D      00495     movwf   TEMP                ; Use TEMP as command bit counter
0255 04E6      00496     bcf     PORTB,ISPDATA       ; Clear DATA line
0256 04C6      00497     bcf     PORTB,ISPCLOCK      ; Clear CLOCK line
0257 0C01      00498     movlw   DATISPOUT           ; Place tris value for data output into W
0258 0006      00499     tris    PORTB               ; Set tris to data output
0259           00500 P16cispcmmdoutloop
0259 0403      00501     bcf     STATUS,C            ; Clear carry bit to start clean
025A 04E6      00502     bcf     PORTB,ISPDATA       ; Clear the DATA line to start
025B 032A      00503     rrf     LOBYTE,F            ; Update carry with next CMD bit to send
025C 0603      00504     btfsc   STATUS,C            ; Skip if bit is supposed to be 0
025D 05E6      00505     bsf     PORTB,ISPDATA       ; Command bit was a one - set DATA to one
025E 05C6      00506     bsf     PORTB,ISPCLOCK      ; Set CLOCK line to clock the data
025F 0000      00507     nop                         ; Wait one cycle
0260 04C6      00508     bcf     PORTB,ISPCLOCK      ; Clear CLOCK line to clock data
0261 02ED      00509     decfsz  TEMP,F              ; Decement bit counter TEMP, skip when done
0262 0A59      00510     goto    P16cispcmmdoutloop  ; Jump back and send next cmd bit
0263 0000      00511     nop                         ; Wait one cycle
0264 04E6      00512     bcf     PORTB,ISPDATA       ; Clear DATA line
0265 04C6      00513     bcf     PORTB,ISPCLOCK      ; Clear CLOCK line
0266 0C81      00514     movlw   DATISPIN            ; Place tris value for data input into W
0267 0006      00515     tris    PORTB               ; set as input to avoid any contention
0268 0000      00516     nop                         ; Wait one cycle
0269 0000      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           00527 programpartisp
026B 0907      00528     call    testmodeisp         ; Place PIC into test/program mode
026C 0064      00529     clrf    FSR                 ; Point to bank 0
026D 0210      00530     movf    STARTCALBYTE,W ; Upper order address of data to be stored into W
026E 0027      00531     movwf   HIADDR              ; place into counter
026F 0211      00532     movf    STARTCALBYTE+1,W ; Lower order address byte of data to be stored
0270 0028      00533     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           00536 programsetptr
0273 0C06      00537     movlw   CMDISPINCRADDR      ; Increment address command load into W
0274 0952      00538     call    commandisp          ; Send command to PIC
0275 02E8      00539     decfsz  LOADDR,F            ; Decrement lower address
0276 0A73      00540     goto    programsetptr       ; Go back again
0277 02E7      00541     decfsz  HIADDR,F            ; Decrement high address
0278 0A73      00542     goto    programsetptr       ; Go back again
0279 0C03      00543     movlw   .3                  ; Place start pointer into W, offset address
027A 008B      00544     subwf   TIMEHIGH,W          ; Restore byte count into W
027B 002F      00545     movwf   BYTECOUNT           ; Place into byte counter
027C 0C12      00546     movlw   STARTCALBYTE+2      ; Place start of REAL DATA address into W
027D 002E      00547     movwf   ADDRPTR             ; Update pointer
027E           00548 programisploop
027E 0C34      00549     movlw   UPPER6BITS          ; retlw instruction opcode placed into W
027F 0027      00550     movwf   HIDATA              ; Set up upper bits of program word
               00551     addrtofsr ADDRPTR           ; Set up FSR to point to next value
0280 0C10          M     movlw   STARTCALBYTE        ; Place base address into W
0281 008E          M     subwf   ADDRPTR,w           ; Offset by STARTCALBYTE
0282 0024          M     movwf   FSR                 ; Place into FSR
0283 06A4          M     btfsc   FSR,5               ; Shift bits 4,5 to 5,6
0284 05C4          M     bsf     FSR,6
0285 04A4          M     bcf     FSR,5
0286 0684          M     btfsc   FSR,4
0287 05A4          M     bsf     FSR,5
0288 0584          M     bsf     FSR,4
0289 0200      00552     movf    INDF,W              ; Place next cal param into W
028A 0028      00553     movwf   LODATA              ; Move it out to LODATA
028B 0208      00554     movf    LODATA,W            ; Place LODATA into LOBYTE
028C 002A      00555     movwf   LOBYTE              ;
028D 0207      00556     movf    HIDATA,W            ; Place HIDATA into HIBYTE
028E 0029      00557     movwf   HIBYTE              ;
028F 006B      00558     clrf    PULSECNT            ; Clear pulse counter
0290           00559 pgmispcntloop
0290 05E3      00560     bsf     STATUS,VFYYES       ; Set verify flag
0291 09B1      00561     call    pgmvfyisp           ; Program and verify this byte
0292 02AB      00562     incf    PULSECNT,F          ; Increment pulse counter
0293 0C19      00563     movlw   .25                 ; Place 25 count into W
0294 008B      00564     subwf   PULSECNT,w          ; Subtract pulse count from 25
0295 0643      00565     btfsc   STATUS,Z            ; Skip if NOT 25 pulse counts
0296 0AA9      00566     goto    pgmispfail          ; Jump to program failed - only try 25 times
0297 0209      00567     movf    HIBYTE,w            ; Subtract programmed and read data
0298 0087      00568     subwf   HIDATA,w
0299 0743      00569     btfss   STATUS,Z            ; Skip if programmed is OK
029A 0A90      00570     goto    pgmispcntloop       ; Miscompare - program it again!
029B 020A      00571     movf    LOBYTE,w            ; Subtract programmed and read data
029C 0088      00572     subwf   LODATA,w
029D 0743      00573     btfss   STATUS,Z            ; Skip if programmed is OK
029E 0A90      00574     goto    pgmispcntloop       ; Miscompare - program it again!
029F 0040      00575     clrw                        ; Clear W reg
02A0 01CB      00576     addwf   PULSECNT,W          ; now do 3 times overprogramming pulses
02A1 01CB      00577     addwf   PULSECNT,W
02A2 01CB      00578     addwf   PULSECNT,W
02A3 002B      00579     movwf   PULSECNT            ; Add 3X pulsecount to pulsecount
02A4           00580 pgmisp3X
02A4 04E3      00581     bcf     STATUS,VFYYES       ; Clear verify flag
02A5 09B1      00582     call    pgmvfyisp           ; Program this byte
02A6 02EB      00583     decfsz  PULSECNT,F          ; Decrement pulse counter, skip when done
02A7 0AA4      00584     goto    pgmisp3X            ; Loop back and program again!
02A8 0AAA      00585     goto    prgnextbyte         ; Done - jump to program next byte!
02A9           00586 pgmispfail
02A9 0446      00587     bcf     PORTB,DONELED       ; Failure - clear green LED!
02AA           00588 prgnextbyte
02AA 0C06      00589     movlw   CMDISPINCRADDR      ; Increiment address command load into W
02AB 0952      00590     call    commandisp          ; Send command to PIC
```

```
02AC 02AE      00591     incf    ADDRPTR,F           ; Increment pointer to next address
02AD 02EF      00592     decfsz  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 Veryify 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   CMDISPLOAD          ; 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 002A      00615     movwf   LOBYTE              ; Move it to LOBYTE reg
02B8 0207      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           00625 loopprgm
02C0 02ED      00626     decfsz  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   CMDISPREAD          ; 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!
               00638     END
```

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXX---------- ---------------- ---------------- ----------------
0200 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0240 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0280 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
02C0 : XXXXXXXXXX----- ---------------- ---------------- ----------------
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


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ; Filename: ISPTEST.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:     PIC16CXX                       *
                     00009 ; * Compiled using MPASM V1.40               *
                     00010 ; ********************************************
                     00011 ; * Include files:                           *
                     00012 ; *           P16CXX.ASM                     *
                     00013 ; ********************************************
                     00014 ; * Fuses:    OSC:  XT (4.0 Mhz xtal)        *
                     00015 ; *           WDT:  OFF                      *
                     00016 ; *            CP:  OFF                      *
                     00017 ; *         PWRTE:  OFF                      *
                     00018 ; ********************************************************************************
                     00019 ; * This program is intended to be used as a code example to               *
                     00020 ; * show how to comunicate with a manufacturing test jig that            *
                     00021 ; * allows this PIC16CXX device to self program.   The RB6 and RB7        *
                     00022 ; * lines of this PIC16CXX device are used to clock the data from        *
                     00023 ; * this device to the test jig (running ISPPRGM.ASM).  Once the         *
                     00024 ; * PIC16C58 running ISPPRGM in the test jig receives the data,          *
                     00025 ; * it places this device in test mode and programs these parameters.    *
                     00026 ; * The code with comments “TEST -“ is used to create some fakecalibration *
                     00027 ; * parameters that are first written to addresses STARTCALBYTE through   *
                     00028 ; * ENDCALBYTE and later used to call the self-programming algorithm.    *
                     00029 ; * Replace this code with your parameter calculation procedure,         *
                     00030 ; * placing each parameter into the STARTCALBYTE to ENDCALBYTE           *
                     00031 ; * file register addresses (16 are used in this example).  The address  *
                     00032 ; * “lookuptable” is used by the main code later on for the final lookup *
                     00033 ; * table of calibration constants.  16 words are reserved for this lookup *
                     00034 ; * table.                                                              *
                     00035 ; ********************************************************************************
                     00036 ; * Program Memory:                                                    *
                     00037 ; *             49 Words - communication with test jig                 *
                     00038 ; *             17 Words - calibration look-up table (16 bytes of data)  *
                     00039 ; *             13 Words - Test Code to generate Calibration Constants  *
                     00040 ; * RAM Memory:                                                        *
                     00041 ; *             16 Bytes -Temporary- Store 16 bytes of calibration constant*
                     00042 ; *              4 Bytes -Temporary- Store 4 bytes of temp variables    *
                     00043 ; ********************************************************************************
                     00044
Warning[217]: Hex file format specified on command line.
                     00045     list p=16C71,f=inhx8m
                     00046     include <p16C71.inc>
                     00001         LIST
                     00002 ; P16C71.INC  Standard Header File, Version 1.00 Microchip Technology, Inc.
                     00142         LIST
2007 3FF1            00047     __CONFIG _CP_OFF&_WDT_OFF&_XT_OSC&_PWRTE_OFF
                     00048
                     00049 ; ***********************************
                     00050 ; * Port A (RA0-RA4) bit definitions *
                     00051 ; ***********************************
                     00052 ; Port A is not used in this test program
                     00053
                     00054 ; ***********************************
                     00055 ; * Port B (RB0-RB7) bit definitions *
```

```
                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 ; ***********************************
  0000000C      00064 CSUMTOTAL   EQU 0Ch ; Address for checksum var
  0000000D      00065 COUNT       EQU 0Dh ; Address for COUNT var
  0000000E      00066 DATAREG     EQU 0Eh ; Address for Data output register var
  0000000F      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
  00000010      00075 STARTCALBYTE    EQU 10h     ; Address pointer for start CAL byte
  0000002F      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
  00000020      00080 CALTABLELENGTH EQU  ENDCALBYTE - STARTCALBYTE + 1
                00081
0000            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 ; ******************************************************************
0000            00092 testcode
0000 3010       00093     movlw   STARTCALBYTE    ; TEST -
0001 0084       00094     movwf   FSR             ; TEST - Init FSR with start of RAM addres
0002            00095 looptestram
0002 0804       00096     movf    FSR,W           ; TEST - Place address into W
0003 0080       00097     movwf   INDF            ; TEST - Place address into RAM data byte
0004 0A84       00098     incf    FSR,F           ; TEST - Move to next address
0005 0804       00099     movf    FSR,W           ; TEST - Place current address into W
0006 3C30       00100     sublw   ENDCALBYTE+1    ; TEST - Subtract from end of RAM
0007 1D03       00101     btfss   STATUS,Z        ; TEST - Skip if at END of ram
0008 2802       00102     goto    looptestram     ; TEST - Jump back and init next RAM byte
0009 0103       00103     clrw                    ; TEST - Clear W
000A 200F       00104     call    lookuptable     ; TEST - Get first CAL value from lookup table
000B 3CFF       00105     sublw   0FFh            ; TEST - Check if lookup CAL table is blank
000C 1903       00106     btfsc   STATUS,Z        ; TEST - Skip if table is NOT blank
000D 2830       00107     goto    calsend         ; TEST - Table blank - send out cal parameters
000E            00108 mainloop
000E 280E       00109     goto    mainloop        ; TEST - Jump back to self since CAL is done
                00110
                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 pogrammed by the test jig    *
                00116 ; * running ISPPRGM later.                                        *
                00117 ; *     Input Variable:  W stores index for table lookup         *
                00118 ; *     Output Variable: W returns with the calibration constant  *
                00119 ; * NOTE: Blank table when programmed reads “FF” for all locations *
                00120 ; ******************************************************************
000F            00121 lookuptable
```

```
000F 0782      00122     addwf   PCL,F           ; Place the calibration constant table here!
               00123
002F           00124     ORG     lookuptable + CALTABLELENGTH
002F 34FF      00125     retlw   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           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
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
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 checksum total byte
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           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 0804      00158     movf    FSR,W           ; Place byte address into W
0046 3C30      00159     sublw   ENDCALBYTE+1    ; Set Z bit if we are at the end of CAL data
0047 1D03      00160     btfss   STATUS,Z        ; Skip if we are done
0048 2842      00161     goto    loopcal         ; Go back for next byte
0049 080C      00162     movf    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           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           00173 sendcalbyte
004D 008E      00174     movwf   DATAREG         ; Place send byte into data register
004E 078C      00175     addwf   CSUMTOTAL,F     ; Update checksum total
004F 3008      00176     movlw   .8              ; Place 8 into W
0050 008D      00177     movwf   COUNT           ; set up counter register
0051           00178 loopsendcal
0051 1706      00179     bsf     PORTB,CLOCK     ; Set clock line high
0052 205C      00180     call    delaysend       ; Wait for test jig to synch up
0053 0D8E      00181     rlf     DATAREG,F       ; Rotate to next bit
0054 1786      00182     bsf     PORTB,DATA      ; Assume data bit is high
0055 1C03      00183     btfss   STATUS,C        ; Skip if the data bit was high
0056 1386      00184     bcf     PORTB,DATA      ; Set data bit to low
0057 1306      00185     bcf     PORTB,CLOCK     ; Clear clock bit to clock data out
0058 205C      00186     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
```

NOTES:

WORLDWIDE SALES & SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602-786-7200 Fax: 602-786-7277
Technical Support: 602 786-7627
Web: www.microchip.com

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
Microchip Technology Inc.
14651 Dallas Parkway, Suite 816
Dallas, TX 75240-8809
Tel: 972-991-7177 Fax: 972-991-8588

Dayton
Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 714-263-1888 Fax: 714-263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 416
Hauppauge, NY 11788
Tel: 516-273-5305 Fax: 516-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong
Microchip Asia Pacific
RM 3801B, Tower Two
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

India
Microchip Technology India
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Shanghai
Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hongiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700
Fax: 86 21-6275-5060

Singapore
Microchip Technology Taiwan
Singapore Branch
200 Middle Road
#10-03 Prime Centre
Singapore 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan, R.O.C
Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886 2-717-7175 Fax: 886-2-545-0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
Unit 6, The Courtyard
Meadow Bank, Furlong Road
Bourne End, Buckinghamshire SL8 5AJ
Tel: 44-1628-851077 Fax: 44-1628-850259

France
Arizona Microchip Technology SARL
Zone Industrielle de la Bonde
2 Rue du Buisson aux Fraises
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 Müchen, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleone
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-39-6899939 Fax: 39-39-6899883

JAPAN

Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shin Yokohama
Kohoku-Ku, Yokohama
Kanagawa 222 Japan
Tel: 81-4-5471- 6166 Fax: 81-4-5471-6122

06/16/97