INTRODUCTION

This document describes the programming techniques that will maximize the software portability when the user is migrating from one revision to another or between two different families of PIC32 microcontrollers. It also offers recommendations to increase the probability of running old software on future PIC32 MCUs. The changes required to port a software application depends on the functional and architectural similarity of the two microcontroller families.

The following cases are provided to describe these techniques:

- Case 1: Writing to SFR
- Case 2: Reading from SFR
- Case 3: Creating a Delay Logic
- Case 4: Using Compiler Supplied Supporting Files

CASE 1: WRITING TO SFR

While modifying the register values, write the recommended values into reserved bit positions in Special Function Registers (SFRs), as specified in the datasheet.

The reserved bits are frequently used to define new capabilities or improvements to existing capabilities.

In most cases, new capabilities are enabled by writing '1' in the appropriate bits. Writing a value of '0' in reserved bit positions ensures that new capabilities will not be enabled accidentally and will not interfere with previously configured capabilities.

For example, to configure the Output Compare module for the PWM mode, load the desired values in the Output Compare Configuration bits.

\[
\text{OCxCON} = 0x8026;
\]

This will write the desired configuration values in the implemented bits and zeros in the reserved bits.

CASE 2: READING FROM SFR

While reading from an SFR, ignore the values of reserved bits.

For example, to check whether there is any new data in the RX buffer of UART, read register bit UxSTA<0> (RXDA) and mask all other bits except the desired bit 0.

\[
\text{unsigned int dataReady} = (\text{UxSTA} & 0x01);
\]

This method ensures that all the bits except bit 0 are ignored and the result represents the value of bit 0 only.

Alternative way of achieving this is by using the register bit structures. For example, call UxSTAbits.RXDA to read the bit. The compiler will execute the necessary bitwise AND operation.
CASE 3: CREATING A DELAY LOGIC

The exact timing of reads and writes to SFRs or memory locations may change when the future revisions or products contain bus matrix changes in one or more of the following forms:

- if fixes are made to the existing bus master peripherals,
- if improvements are made to the existing bus master peripherals,
- if new peripherals are added, e.g., CAN, Ethernet, DMA, USB, ICD, etc.

As a result, applications requiring hard timing delays should use the core timer or any of the available peripheral timers.

A specific delay can be created through various practices. A common practice, which is not recommended, and the two recommended practices are described in this document.

EXAMPLE 1: COMMON PRACTICE (NOT RECOMMENDED)

```c
while(1)
{
    for (i = 0; i < 10000; i++);  // Achieve X ms of delay @ Y MHz
    mPORTAToggleBits(BIT_2);    // Toggle the pin
}
```

Common Practice (Not Recommended)

The common practice to achieve a specific delay is shown in Example 1. This is not recommended since the delay varies when the bus matrix changes.

Recommended Practices

The following practices are recommended for achieving a specific delay:

- Configure the core timer or a peripheral timer to generate the required delay (see Example 2) or
- Enable the interrupt and perform the necessary action in the interrupt handler (see Example 3)

EXAMPLE 2: USING CORE TIMER

```c
#define CORE_TIMER_PERIOD (0xFFEF)

OpenCoreTimer(CORE_TIMER_PERIOD);  // Open the core timer with desired period

while(1)
{
    while(mCTGetIntFlag() == 0);       // Wait for the coretimer compare
    mCTClearIntFlag();                // Clear the interrupt flag
    mPORTAToggleBits(BIT_2);         // Toggle the pin
    UpdateCoreTimer(CORE_TIMER_PERIOD); // Update the period
}
```

EXAMPLE 3: USING CORE TIMER WITH INTERRUPT ENABLED

```c
#define CORE_TIMER_PERIOD (0xFFEF)

OpenCoreTimer(CORE_TIMER_PERIOD);  // Open the core timer with desired period

mConfigIntCoreTimer((CT_INT_ON | CT_INT_PRIOR_2 | CT_INT_SUB_PRIOR_0));

// Interrupt Handler
void __ISR(_CORE_TIMER_VECTOR, ipl2) CoreTimerHandler(void)
{
    mCTClearIntFlag();       // Clear the interrupt flag
    mPORTAToggleBits(BIT_2); // Toggle the pin
    UpdateCoreTimer(CORE_TIMER_PERIOD); // Update the period
}
**CASE 4: USING COMPILER SUPPLIED SUPPORTING FILES**

Always use the processor header files, peripheral libraries and other support files that are supplied with the compiler.

The processor header files abstract the underlying register address values and bit positions. As a result, changes to source code are not required to compile on different compiler versions or for different PIC32 families.

The peripheral libraries abstract the exact programming sequences and any variations that exist in different revisions or families.

In Example 4, the processor header file `p32xxxx.h` and peripheral library header file `plib.h` have been included to achieve compatibility.

**EXAMPLE 4: USING PROCESSOR AND LIBRARY HEADER FILES**

```c
#include <p32xxxx.h>
#include <plib.h>

int main(void) {
    mPORTDSetPinsDigitalOut(BIT_2); // configure the port register
    mPORTDClearBits(BIT_2); // Initialize the port pin to low

    while(1) {
        mPORTDToggleBits(BIT_2); // Toggle the pin
        // user code
    }

    return (0);
}
```

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

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

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

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

Trademarks
The Microchip name and logo, the Microchip logo, Accuron, dsPIC, Keeloq, Keeloq logo, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, rFIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.
FilterLab, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.
Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, PicKit, PICDEM, PICDEM.net, PicTail, PICmicro logo, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rFAB, Select Mode, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.
SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.
All other trademarks mentioned herein are property of their respective companies.
© 2008, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.
Printed on recycled paper.

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