



# AN2039

## Four-Channel PIC16F1XXX Based Power Sequencer

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

### INTRODUCTION

Power sequencers are commonly used in system-level board designs where multiple power supplies are enabled in a sequential manner. Typically, systems using a power sequencer have different components on them which require different power supply voltages and power levels. The sequence of enabling the different voltages would insure that there is no conflict between components being powered up and all units are powered up correctly. When shutting down the system, there may also be a sequence. The power-up and power-down sequence is programmable and is time based. This application note defines a four-channel power sequencer. The four voltages defined are 5.0V, 3.3V, 2.5V and 1.8V. Each of these voltages is provided through a power module (PM) unit (BEL power modules: VRAE-10E1A0 Series). These PM units typically have five pins: Input/Output Power, Ground, Enable and Trim (Figure 1). Users can select any number of PM units in their system. The software has been written in a modular format to support up to ten PMs. These can be added or removed to meet specific needs. An enhanced core PIC16F1509 device has been selected as the MCU for this application.

The PIC16F1509 peripherals used in this application note are:

- GPIO
- Timer1
- ADC
- I<sup>2</sup>C
- PWM(4)

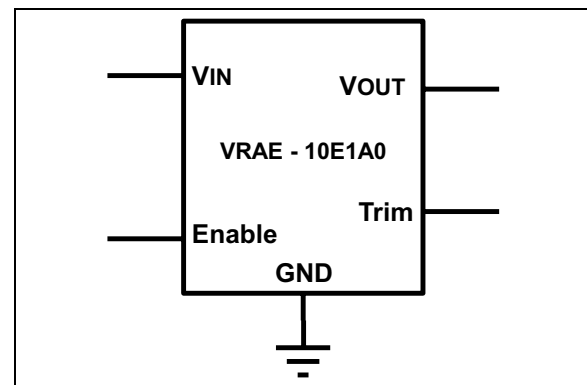
### HARDWARE DESIGN CONSIDERATIONS

#### Power Module (PM) Units

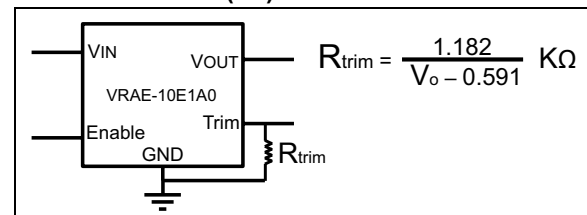
The PM units are typically off-the-shelf power supply blocks, sold with specific current and power capabilities. The PMs used in our design are: VRAE-10E1A0 Series made by BEL Power Products (Figure 1). Each PM has five pins: Input Voltage (VIN), Output Voltage (VOUT), Ground, Enable Input and Output Trim. The Enable signal is active-high and when

enabled, the PM output voltage appears at the Output Voltage pin. The Trim pin with the associated Rtrim resistor (Figure 2) allows the output voltage to be set as desired. The voltage at the Trim pin is typically around 0.591V. A DC voltage at the Trim pin also allows the system to provide some voltage load regulation of VOUT. In this design, the DC voltage is provided by a PWM-driven RC filter circuit which provides a DAC output to the Trim pin (Figure 3). Alternately, this DC voltage can be provided only by using fixed resistors from the Trim pin to GND (Figure 2) with no PWM trim DAC voltage. If this alternate option is selected, the PWM, with its associated hardware/firmware in the PIC16 microcontroller, would be eliminated. Refer to the BEL data sheet for more details on the trim-down resistor values.

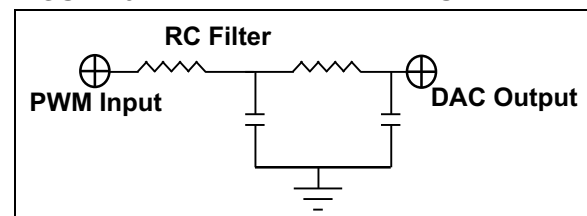
**FIGURE 1: BEL POWER MODULE**



**FIGURE 2: Rtrim FORMULA FOR Vout (Vo)**



**FIGURE 3: RC FILTER CIRCUIT**



## Power-Up Sequencing

A PIC16F1509 operating at 5.0V and 4 MIPS (using the internal RC clock) is used to control the power-up sequence. The power-up sequence is initiated by:

- A Serial command using the I<sup>2</sup>C interface
- Pressing the push button switch, S1

Each PM is sequenced ON at a set time interval from 1 to 16,393 ms (16.4 seconds) with a 1 ms accuracy. For example, PM1 can be started at 10 ms from the Start command, followed by PM2 at 25 ms, PM4 at 200 ms and finally, PM3 at 1000 ms. Each PM has a corresponding ON time value which is a 14-bit unsigned integer value in firmware. This value is compared to a timer value incremented every millisecond. If a match between the timer value and the ON time value of the PM occurs, then the corresponding PM is turned ON. The ON/OFF timing can be selected by the user and is saved in Flash on the PIC16 device. The ON/OFF sequence can be started/stopped using the serial I<sup>2</sup>C GUI.

## Trimming

When a PM is turned ON, the corresponding PWM output is enabled and V<sub>OUT</sub> is monitored via the PIC16 ADC. The PWM duty cycle corresponds to the 8-bit DAC value for the PM. This DAC value can be changed by the user in the GUI or in the firmware. The trim voltage is created by using a combination of R<sub>trim</sub> and the PWM output from the PIC16 microcontroller. This PWM output is sent to a RC filter to create a DAC voltage, which in combination with the R<sub>trim</sub> resistor, is applied to the trim pin of the PM (Figure 3). The output of the PM is monitored using a 10-bit ADC converter on the PIC16 microcontroller. Each PM voltage is averaged over 16 readings to give a 14-bit value. Only the Most Significant eight bits of this value are used to reference the V<sub>OUT</sub> voltage value of each PM. The reference voltage of the ADC is V<sub>DD</sub> or 5.0V. For example, if the PM output voltage is 2.5V, then the accuracy of the measurement would be  $(2.5V/5.0V)/256 = 2.0 \text{ mV}$ . All output voltages are constantly monitored to verify they are within the specified over/undervoltage limits specified by the customer. If the PM voltage goes above or under the under/overvoltage limits, then a failure is signaled and the system is automatically shut down.

## Power-Down Sequencing

The PIC16 MCU also controls the programmable power-down sequencing of the four power supplies. The power-down sequence is initiated on:

- Serial command from the I<sup>2</sup>C
- Any Fault condition on the PMs or input voltage
- Pressing the push button switch, S1

Each PM is sequenced OFF at a set time interval from 1 to 16,393 ms (16.4 seconds) with a 1 ms accuracy. For example, PM4 can be shut at 20 ms from the Stop command, followed by PM2 at 25 ms, PM3 at 200 ms and finally, PM1 at 1000 ms. Each PM has a corresponding OFF time value, which is an unsigned integer (14-bit value). This value is independent from the ON time value. This value is compared to a 16-bit counter value incremented every millisecond. If the two are equal, then the corresponding PM is turned OFF. The OFF time values are user selectable and are saved in Flash. In case of a Fault condition power-down, a new power-up sequence will be automatically initiated depending on the number of retries selected by the user. Typically, a user may specify two or three retries. After all retries return a failure, the system is shut down and the Fault condition is signaled. Using the I<sup>2</sup>C GUI interface, the user can figure out which PM condition or input voltage caused the failure. The user must take the appropriate corrective action to remove the failure condition and reset the system using the I<sup>2</sup>C Serial command or the Graphical User Interface (GUI), and then retry the power-up sequence.

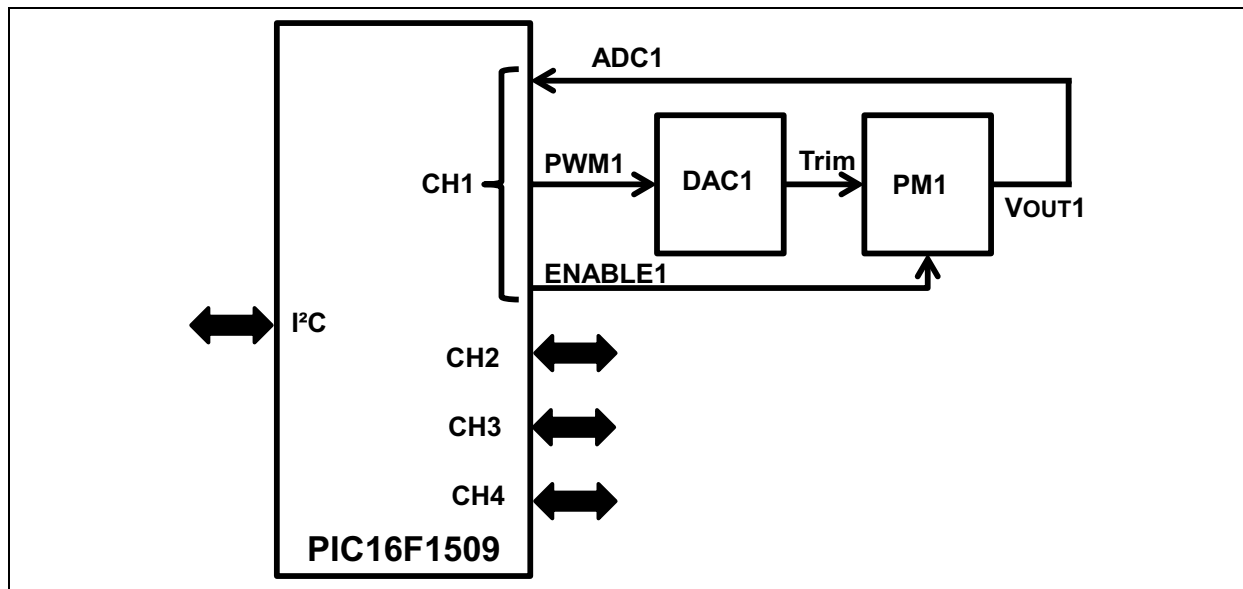
## MCU Requirements

The MCU requirements fall in four categories:

- I/O pins to enable/disable the PMs
- ADC input to sample the PM V<sub>OUT</sub> voltage
- PWM output to generate the DAC output for the trim voltage
- Communication using I<sup>2</sup>C

Since four channels of PMs are used in this design, at least four I/O lines for the enable/disable function is required. Also required is four ADC channels, four PWM outputs and finally, two lines for I<sup>2</sup>C. An additional ADC channel is used to sample the input voltage, the MCLR, the V<sub>DD</sub>, the V<sub>SS</sub> and the programming pins; a total of 20 pins. A PIC16F1509 was selected for the design. Figure 4 is the system block diagram.

FIGURE 4: SYSTEM BLOCK DIAGRAM



The MCU is powered by 5.0V via a 5V regulator. The internal 16 MHz RC clock is used to run the CPU at 4 MIPS. The hardware/firmware can be modified to accommodate up to ten PMs. If more PMs are required, the number of I/O will increase and a larger PIC16F1XXX device would have to be selected. If less PMs are required, then a smaller PIC16F1XXX device can be selected for the application. The trim voltage requirements can also be adjusted. If the user wants to trim the PMs using only the external resistor (Figure 2), then the DAC voltage and the associated PWM (Figure 3) is not required. The software to drive the PWM and DAC will also be eliminated.

### Voltage Limits

Each PM has its own normal over/undervoltage limit. The PMs also have margin under/over limit for the trim voltage.

### Normal Over/Under Limits

As specified, these limits are the absolute limits for each PM. The output voltage of each PM is monitored and, in case the output voltage goes under or above these limits at any time, a fault is signaled and the power is sequenced OFF. A retry is initiated after a power fault occurs. After a certain number (defined by the user) of automatic retries, the system is shut down until the user clears the system with a Serial command using the I<sup>2</sup>C GUI and fixes the hardware/software error. For example, for a 5V PM, the user may select 4.5V as the undervoltage limit and 5.5V as the overvoltage limit.

### Margin Over/Under Limits

Each PM has a user-defined margin over/under limit. These limits are 8-bit values which are maintained in the firmware and correspond to the PWM duty cycle value of the respective PM. The user can modify these values via the GUI. The DAC value can be changed within the bounds of the over/under margin limits.

**Note:** The trim voltage is not monitored by the PIC16 ADC. Only the trim value (i.e., the PWM duty cycle value), is compared with the limit set by the user. This value is defined as a 8-bit value.

### I<sup>2</sup>C Communication Protocol and Commands

An I<sup>2</sup>C slave interface is implemented on the MCU for serial communication with an external I<sup>2</sup>C GUI. The command structure for the I<sup>2</sup>C interface is specified in [Appendix A: "Serial Command Definitions and Structures"](#). On the demo board, a MCP2221 I<sup>2</sup>C-to-mini-USB interface chip is provided. This interface can be implemented by the users in their own hardware or if required, a different I<sup>2</sup>C interface can be implemented by the user.

### I<sup>2</sup>C Addressing

Only one device is connected to the interface so a fixed slave address of 0x14 is selected for the sequencer. This address can be changed in firmware if required. PMBus™ is not supported at this time.

## FIRMWARE IMPLEMENTATION

### Peripherals Code Generated Using the Microchip Code Configurator (MCC)

All the firmware for I/O, Timer1, ADC, PWM, Flash memory and I<sup>2</sup>C peripherals has been created and initialized using the free MPLAB<sup>®</sup> Code Configurator (MCC) software. The system setup also uses the MCC software. The entire firmware was developed using the MPLAB<sup>®</sup> X IDE. The use of the MCC software is highly recommended since the firmware created is tested and well documented. The initialization and the user routines for each peripheral are modular and, in case the user wants to use another larger or smaller PIC16F1XXX device, the re-creation of peripheral routines is very simple and takes very little time. Using the MCC, the code for all the peripherals will be automatically generated and placed in the MPLAB X project. As an example, [Appendix B: “Using MPLAB<sup>®</sup> Code Configurator \(MCC\) to Add Timer1 Function to an Application”](#) shows a step-by-step process using the MCC to generate code for Timer1.

### I<sup>2</sup>C Slave Interrupt Firmware

The I<sup>2</sup>C Slave Interrupt Firmware has also been created using MCC. The setup using MCC is explained in detail in [Appendix C: “Modifying MCC created I2C Slave Interrupt file to implement Application Serial Interface”](#), along with the modifications required. The MCC firmware has a built-in example of a serial EEPROM. This firmware is modified to incorporate the slave I<sup>2</sup>C used in the power sequencer program. This method is quick and very easy to use without any prior knowledge or setup details on the I<sup>2</sup>C protocol or function.

### User Application

All code has been written in state machine format. The code for the PM modules is in the file `app.c`. The code for the ADC, Flash, Key and System is in the respective files `appAdc.c`, `appFlash.c`, `appKey.c` and `appSys.c`. The definitions and the variables used are defined in `app.h`. The entire application code is state machine based, where each task is defined as a series of states. For example, there is a state machine for the ADC, for each PM module, for Flash memory and for the overall system. The `*main.c` program is very simple, as shown in [Example 1](#).

#### EXAMPLE 1: CODE SNIPPET

```
void main(void) {

    //All state machines are initialized at the start
    SYSTEM_Initialize();
    APP_FLASH_Initialize();
    APP_M0_Initialize();
    APP_M1_Initialize();
    APP_M2_Initialize();
    APP_M3_Initialize();
    APP_M4_Initialize();
    APP_KEY_Initialize();
    APP_ADC_Initialize();
    APP_SYS_Initialize();
    //In an infinite loop each state machine is run one after another
    while (1)
    {
        APP_M0_Tasks();
        for (MI=1; MI < AllModules; MI++)
            APP_MX_Tasks();
        APP_KEY_Tasks();
        APP_ADC_Tasks();
        APP_SYS_Tasks();
        APP_FLASH_Tasks();
    }
}
```

Each initialize routine initializes the module, the system or the peripheral. On the other hand, each task is a state machine which runs the tasks for the application PM modules 0 to 4, the ADC, KEY, Flash memory and the overall system.

**Note:** Note that modules 1 to 4 correspond to the PMs 1 to 4. Module 0 is designated to run a state machine which monitors the input voltage.

## Parameter Structure for Each Module

Each power module is assigned a set of parameters which are defined in `app.h`.

The parameters are:

- State – it defines the state in which each module is in at a given moment in time

The eight states are:

- Init – the initial state of the module
- On – the module is in the ON state
- Off – the module is in the OFF state
- Start – the module is turned ON
- Starting – the module rise time is taken into account when turning ON
- Stop – the module is turned OFF
- CheckADC – updates the ADC value for the corresponding PM module
- CheckError – checks the output voltage for a under/overvoltage fault

**Note:** Appendix E shows the State Machine Diagram for the module state machine.

- NormalUVL, NormalOVL, MarginUVL, MarginOVL – OverVoltageLimit and UnderVoltageLimit for normal and margin voltages. These variables are used and maintained for error checking. They are 8-bit unsigned value and they can be modified by the user
- ADC – unsigned 16-bit value to maintain 16 counts of 10-bit ADC values
- PWMValue – 10-bit PWM duty cycle value used in the specific PWM with the module
- ADCValue – 8-bit average value of ADC over 16 counts of 10-bit values. This value is compared with the limit voltage values and checked for errors
- anChannel – ADC channel number assigned to module
- OnTime and OffTime – 14-bit value in ms for the start and stop time for each module
- TurnOn – bit value to indicate if the module is turned ON

All parameters are defined in a type-defined structure and an array of this structure is defined as: `appmData[5]`. The `appmData[1]` is for module 1 and `appmData[4]` is for module 4. The input voltage, VIN is assigned parameters in `appmData[0]` or the first element of the array. Note that only some of the parameters may be relevant to VIN. [Example 2](#) shows a PWM parameter for module 3.

### EXAMPLE 2: PWM DUTY CYCLE PARAMETER

`appmData[3].PWMValue` – corresponds to the 10-bit duty cycle value of the PWM used for module 3.

Apart from creating this structure, each module has an initialize routine. The task routines are common for each PM module 1 to 4, so a common `APP_MX_Tasks()` routine is used along with a module index (MI) parameter corresponding to the respective PM module. VIN is assigned its own routine `APP_M0_Tasks()` ;.

By creating this structure and the associated functions for each module, it becomes very easy for the user to increase or decrease the number of modules used in the application. If the number of required modules is three, then the number of elements can be decreased by one (from four to three). One initialize and one task function can also be deleted. However, if the number of modules needs to be increased by one to five, then the number of elements in the array needs to be increased as well (from four to five). A copy/paste of one additional initialize routine will be needed and the appropriate parameters adjusted to reflect the new module 5. [Appendix D: “Adding an additional Module to the Existing Code”](#) is an example of how to add one additional module 4 to the application.

## ADC Routine and Voltage Measurement

The ADC routine essentially runs through and samples the voltage of each module 0 to 4. Module 0, as mentioned earlier, corresponds to the input voltage, which always gets monitored for a failure. A failure of the input voltage causes a shutdown. No retries will be attempted. The 10-bit ADC routine samples each voltage 16 times and then uses the average 8-bit value to check for an error with the corresponding under/overvoltage limits. In the hardware used, the voltage reference is 5.0V or VDD of the system. A 5.0V voltage reference will work fine when sampling and converting 1.8V, 2.5V and 3.3V. However, for the 5.0V module and the input voltage, a resistor divider is needed to bring the full-voltage range within the 5.0V reference voltage. The resistor divider factor for the 5.0V module is 0.55 and the input voltage divide factor is 0.239. Users will have to use this value during the calculation of the under and overvoltage limit values and define them appropriately in the header files. This is especially required if the user decides to use values other than those used in this application note.

## ADC State Machine States

- Init – initializes the ADC converter
- Sample – in this state, the ADC channel is assigned a 1 ms sampling time before the conversion is started
- Convert – the Convert command is given and the ADC conversion is started

- Done – a check to see if the conversion is complete; if yes, then the converted value is added and if all 16 samples have been taken, the appropriate module state is assigned for a voltage update and voltage check
- Next – in this state, the next module is assigned to the ADC and the whole process is repeated

The ADC state machine is always running. A designated module voltage is only checked if that module is turned ON or is in the ON state. Otherwise, only the input voltage is checked. Once a module is turned ON it is always checked.

## KEY State Machine

The KEY state machine has the following states:

- Init – initializes the state machine
- High – when the key is at the default high state, a check for a key press (0V) is done
- Low – when the key is in the pressed state, a check for a release (5V) is done
- Debounce – in this state a delay of 20 ms is executed then a check for a high or low state is done

The KEY state machine insures the proper functioning of the key-press switch, S1. If the system is OFF, a key press will turn the system ON. If the system is ON, a key press will turn the system OFF.

## SYS State Machine

The SYS state machine manages the overall functioning of the system. In particular, it manages failure modes during start-up. If a start-up failure occurs, then automatic restarts are executed as defined by the user in the retry parameter. The APP\_SYS\_Tasks() routine makes sure that all the retry attempts are executed. If all retry attempts are exhausted, the system will shut down and a Fault condition will result. The user must then reset the system using a Serial command on the I<sup>2</sup>C interface or a button on the GUI. Any failure mechanism must be fixed to prevent another failure during start-up. The system will not restart until the Reset command is executed or a Power-down Reset is done.

The SYS states are:

Init – initialize state for SYS

ON – state when the system is ON

OFF – state when the system is OFF

Starting – state when the system is starting up

Stopping – state when the system is shutting down

Fault – state when a system fault has occurred

## Flash State Machine

The default parameters are saved in the Flash memory of the PIC16F1509. The Flash PM storage location starts at 0x1f80 and the parameters are saved sequentially for each module. The default parameters for each module are:

- Start time
- Stop time
- Normal Undervoltage Limit
- Normal Overvoltage Limit
- Margin Undervoltage Limit
- Margin Overvoltage Limit
- PWM Duty Cycle

Apart from these 28 parameters (4 x 7), the system parameters are also saved:

- VIN Undervoltage Limit
- VIN Overvoltage Limit

## Number of Retries

The user can use the GUI or the Serial command to change the value of these parameters. These values have to be “burnt” onto the Flash memory by the user, using the **Burn Flash** button on the GUI or the Serial command: 0x10, 0xAB. Once they are burnt, they become the new default values. The file `flash.as` defines all the default values and locations in Flash PM.

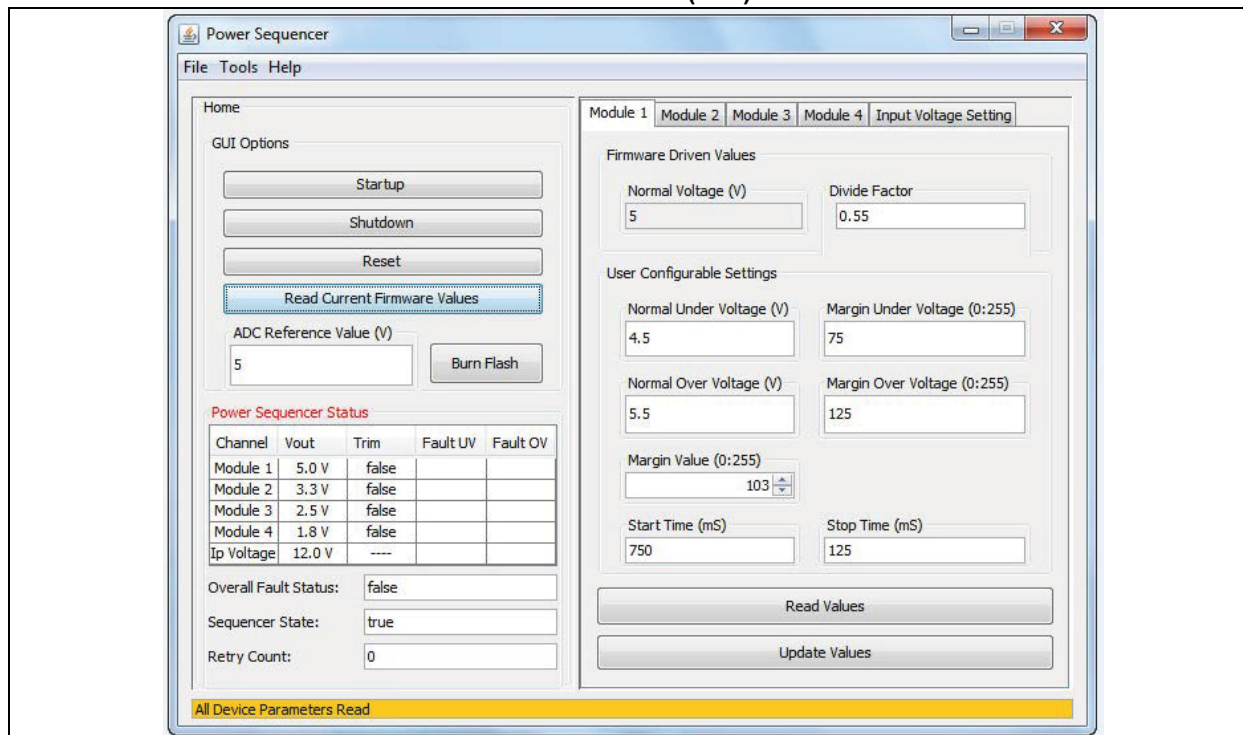
## Power Sequencer GUI

The power sequencer GUI has been designed for the user to:

1. Enter relevant data for the power sequencer application
2. Monitor relevant data from the power sequencer application
3. Control the power sequencer application

The Graphic User Interface (GUI) is shown in [Figure 5](#).

FIGURE 5: THE GRAPHICAL USER INTERFACE (GUI)



The Main window has the system options on the left and the module options as tabs on the right.

In the system options, the user can Start, Stop, Reset and Read the current firmware values. The Status window allows the user to define the VOUT corresponding to the module index. These values can be modified by the user and will be saved when the GUI is closed. The user can also enter the ADC reference value, which for this application note is set at 5.0V. Finally, the user can burn the updated module settings onto the Flash program memory by clicking on the **Burn Flash** button.

Under each module tab, the user can also set or read existing values for each module. Module 1 is the 5V module and the user can set the normal/margin, over/under limits for this module, along with start and stop times in milliseconds for this module. Also, the voltage divide factor can also be edited and entered in this tab. The **Read Values** button reads the existing values in RAM, and the **Update Values** button writes new values to the RAM. If the user wants to make these values permanent, then they would have to be programmed into the Flash using the **Burn Flash** button. By clicking on the appropriate tab, the user can modify/read/update values for all modules.

When a Fault occurs, the GUI is not automatically updated, since the I<sup>2</sup>C implementation on the PIC16 is in Slave mode. The user has to click the **Read Current Firmware Values** button to get a status update and identify which module failed.

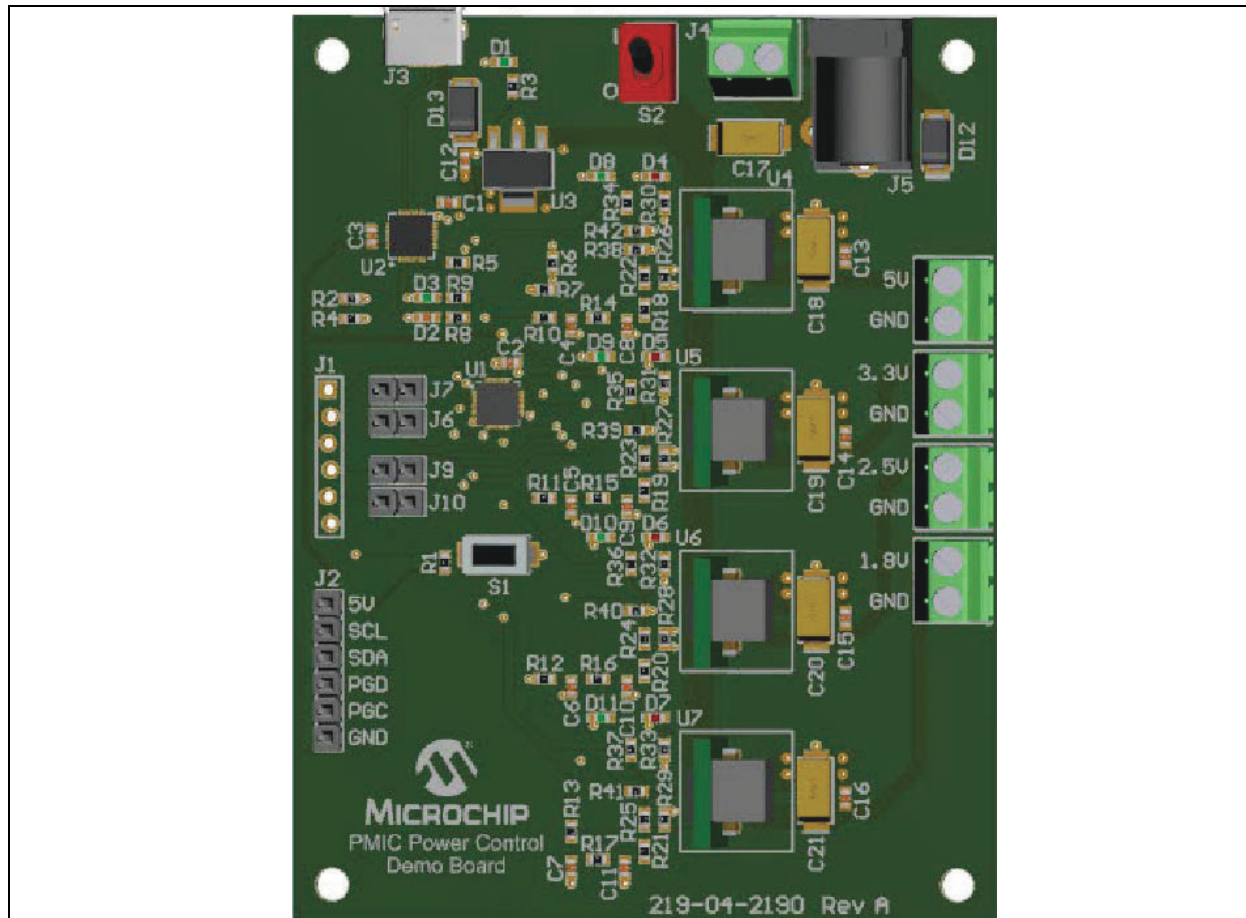
In each module tab, the DAC value can be incremented or decremented using the up/down arrow located at one side of the DAC value box. The value increments or decrements and if the module is ON, then the output voltage will be read and updated. To see the voltage change, more than one increment or decrement may have to be performed. This feature allows the user to increase or decrease the output voltage during a system test when voltages reach their limit. This is called voltage limit testing and allows the customer to test a complete system when one or more of the output voltages reaches their under/overvoltage limits.

The voltage values are displayed as actual voltage values (3.3V or 2.5V). The DAC value and the margin limits are displayed as 8-bit values from 0 to 255.

## CONCLUSION

This application note is designed for easy use by the customer when implementing a power sequencer design application using a PIC16F1XXX device. The user can easily modify this application note to control four voltage modules in their own design application. Additionally, customers can add more power modules to their application or remove modules for a smaller application. The hardware and firmware have been created in a modular format to accomplish these goals easily. The completed board is depicted in [Figure 6](#) below. [Appendix F: “Power Sequencer Schematics”](#) shows the schematics for the board.

**FIGURE 6: POWER CONTROL DEMO BOARD**



## APPENDIX A: SERIAL COMMAND DEFINITIONS AND STRUCTURES

### I<sup>2</sup>C Serial Command Format

#### NOTATIONS

- S = Start
- P = Stop
- A = Acknowledge
- N = N
- W = Read/Write bit

#### Written Commands

S<Slave Address> W A <Command> A <Parameter> A <ValueL> A <ValueH> A P

**Note:** Some commands will only contain a parameter and no ValueL or ValueH parameter will be sent. Typically, these are Write commands which precede a Read command.

#### Read Commands

S <Slave Address> R A <ValueL> A <ValueH> N P

#### COMMAND 0X10: PROGRAM FLASH

Writing a 0x10 followed by a 0xAB will start a Program Flash operation. Once the user has updated all the parameters for all the modules (using the GUI or I<sup>2</sup>C serial instructions mentioned in this section), a Program Flash operation will permanently burn the values onto the Flash memory of the PIC16F1509 device. A subsequent Reset or Power-up Reset will retrieve values from the Flash memory. This operation is recommended after the user updates all the module parameters.

#### COMMAND 0X20: SET MODULE START TIME VALUE

Writing a 0x20 command followed by the module number, will set a new start time for the specified module. The two data bytes which follow this command and parameter will contain the desired start time setting in milliseconds. Since the start time is only 14 bits, the value will be limited from 1 to 16393 ms.

#### COMMAND 0X24: READ MODULE START TIME VALUE

Writing a 0x24 command followed by the module number will read the start time for the specified module. A restart is then sent with a read cycle with the device slave address, to read the two data bytes. The two data bytes which follow this command and parameter will

contain the desired start time setting in ms. Since the start time is only 14 bits, the value will be limited from 1 to 16393 ms.

#### COMMAND 0X28: SET MODULE STOP TIME VALUE

Writing a 0x24 command followed by the module number, will set a new stop time for the specified module. The two data bytes which follow this command and parameter will contain the desired stop time setting in milliseconds. Since the stop time is only 14 bits, the value will be limited from 1 to 16393 ms.

#### COMMAND 0X2C: READ MODULE STOP TIME VALUE

Writing a 0x2C command followed by the module number, will read the stop time for the specified module. A restart is then sent with a read cycle with the device slave address to read the two data bytes. The two data bytes which follow this command and parameter will contain the desired stop time setting in milliseconds. Since the stop time is only 14 bits, the value will be limited from 1 to 16393 ms.

#### COMMAND 0X30: SET DAC MARGIN OUTPUT VOLTAGE

Writing a 0x30 command followed by the module number as the parameter, will send a new margin setting for the parameter specified PWM/DAC channel. The two data bytes which follow this command and parameter will contain the desired PWM/DAC output setting. Since the PWM/DAC is only eight bits, the ValueH is zero.

#### COMMAND 0X38: READ DAC MARGIN VOLTAGE

Writing a 0x38 command followed by the module number will setup a read of the present DAC value. A restart is then sent with a read cycle with the device slave address, to read the two data bytes. A read will respond with the high and low value for the DAC, however, since the value is only 8-bit, the high value will be zero and the low byte will contain the DAC value.

#### COMMAND 0X34: READ SYSTEM RETRY COUNT

Writing a 0x34 command followed by 0x00 as the parameter, will read the preset retry count parameter specified for the system. The two data bytes which follow this command and parameter will contain the retry counts. Since this value will typically be less than ten, the ValueH is ignored.

## COMMAND 0X3C: SET SYSTEM RETRY COUNT

Writing a 0x3C command followed by 0x00 as the parameter, will set the preset retry count parameter specified for the system. The two data bytes which follow this command and parameter will contain the retry counts. Since this value will typically be less than 10, the ValueH is zero.

## COMMAND 0X40: SET NORMAL UNDERVOLTAGE LIMIT

Writing a 0x40 followed by the module number as the parameter, will send a new normal undervoltage limit for that channel. The two data bytes which follow this command and parameter will contain the desired normal undervoltage limit for that ADC channel and apply to that power supply. This is an 8-bit value, so ValueH is zero.

## COMMAND 0X50: SET NORMAL OVERVOLTAGE LIMIT

Writing a 0x50 followed by the module number as the parameter, will send a new normal overvoltage limit for that channel. The two data bytes which follow this command and parameter will contain the desired normal overvoltage limit for that ADC channel and apply to that power supply. This is an 8-bit value so, ValueH is zero.

## COMMAND 0X60: SET MARGIN UNDERVOLTAGE LIMIT

Writing a 0x60 followed by the module number as the parameter, will send a new margin undervoltage limit for that channel. The two data bytes which follow this command and parameter will contain the desired margin undervoltage limit for that ADC channel and apply to that power supply. This is an 8-bit value, therefore ValueH is zero.

## COMMAND 0X70: SET MARGIN OVERVOLTAGE LIMIT

Writing a 0x70 followed by the module number as the parameter, will send a new margin overvoltage limit for that channel. The two data bytes which follow this command and parameter will contain the desired margin overvoltage limit for that ADC channel and apply to that power supply. This is an 8-bit value so, ValueH is zero.

## COMMAND 0X80: READ NORMAL UNDERVOLTAGE LIMIT

Writing a 0x80 followed by the module number as the parameter, will initiate a read for the normal undervoltage limit for that ADC channel. A restart is then sent with a read cycle with the device slave address, to read the two data bytes. The two data bytes which are read will contain the current normal

undervoltage limit for that module. Since the value is 8-bit only, the low byte will contain the value and the high byte will be zero. The command will allow clock stretching from the Master for several milliseconds.

## COMMAND 0X90: READ NORMAL OVERVOLTAGE LIMIT

Writing a 0x90 followed by the module number as the parameter, will initiate a read for the normal overvoltage limit for that ADC Channel. A restart is then sent with a read cycle with the device slave address, to read the two data bytes. The two data bytes which are read will contain the current normal overvoltage limit for that ADC channel. Since the value is 8-bit only, the low byte will contain the value and the high byte will be zero. The command will allow clock stretching from the Master for several milliseconds.

## COMMAND 0XA0: READ MARGIN UNDERVOLTAGE LIMIT

Writing a 0xA0 followed by the module number as the parameter, will initiate a read for the margin undervoltage limit for that ADC channel. A restart is then sent with a read cycle with the device slave address, to read the two data bytes. The two data bytes which are read will contain the present Margin undervoltage limit for that ADC channel. Since the value is 8-bit only, the low byte will contain the value and the high byte will be zero. The command will allow clock stretching from the Master for several milliseconds.

## COMMAND 0XB0: READ MARGIN OVERVOLTAGE LIMIT

Writing a 0xB0 command followed by the module number as the parameter, will initiate a read for the margin overvoltage limit for that ADC channel. A restart is then sent with a read cycle with the device slave address, to read the two data bytes. The two data bytes which are read will contain the current margin overvoltage limit for that ADC channel. Since the value is 8-bit only, the low byte will contain the value and the high byte will be zero. The command will allow clock stretching from the Master for several milliseconds.

## COMMAND 0XC0: READ POWER SEQUENCER STATUS

Writing a 0xC0 command followed by the desired STATUS register as the parameter, will initiate a read of the desired STATUS register. A restart is then sent with a read cycle with the device slave address to read the two data bytes of the STATUS register. The STATUS registers are as shown in [Table A-1](#) below.

**TABLE A-1: STATUS REGISTERS**

Register no.	Value
0	Power supply undervoltage error bits, one for each PM
1	Power supply overvoltage error bits, one for each PM
2	Fault<15>, Seq.State<14>,Margin Enable<0:9>
3	Number of system sequence retries since the last power ADC channel

#### COMMAND 0XC8: CLEAR STATUS ERROR FLAGS AS WELL AS RETRY NUMBER

A 0xC8 command followed by 0xCE will send a Clear command to clear all the Status bits mentioned in the 0xC0 command execution above. STATUS registers 0 to 3 will all be cleared. The margin enable bit will not be cleared.

#### COMMAND 0XD0: READ MODULE ANALOG VOLTAGE

Writing a 0xD0 command followed by the module number as the parameter will initiate a read for the module voltage reading. A restart is then sent with a read cycle to the device slave address to read the two data bytes. The two data bytes which are read will contain the present voltage from that module voltage. Since the value is 8-bit only, the low byte will contain the value and the high byte will be zero. The command will allow clock stretching from the Master for several milliseconds.

#### COMMAND 0XE0: POWER-UP

Writing a 0xE0 command followed by 0xEF will start a power-up sequence. If the power sequencer is already ON, then this command will be ignored. This command would typically be used to restart the modules after a desequencing event which was triggered by an under/overvoltage fault that has occurred.

#### COMMAND 0XF0: POWER-DOWN

Writing a 0xF0 command followed by 0xDF will start a power-down sequence. This command is ignored if the controller is presently in the start-up or power-down state. If the controller is in the ON state, then this command will initiate a power-down sequence.

**TABLE A-2: COMMAND SUMMARY**

Command	Parameter	Write Data 1	Write Data 2	Read Data 1	Read Data 2	Description
0x10	0xAB					Program Flash
0x20	DAC#	ValueL	ValueH			Set Start Time
0x24	DAC#			ValueL	ValueH	Read Start Time
0x28	DAC#	ValueL	ValueH			Set Stop Time
0x2C	DAC#			ValueL	ValueH	Read Stop Time
0x30	DAC#	ValueL	ValueH			Set DAC output
0x38	DAC#			ValueL	ValueH	Read DAC Value
0x34	0x00			ValueL	ValueH	Read Retry Count
0x3C	0x00	ValueL	ValueH			Set Retry Count
0x40	Ch #	ValueL	ValueH			Set Normal UV limit
0x50	Ch #	ValueL	ValueH			Set Normal OV limit
0x60	Ch #	ValueL	ValueH			Set Margin UV limit
0x70	Ch #	ValueL	ValueH			Set Margin OV limit
0x80	Ch #			ValueL	ValueH	Read Nor. UV limit
0x90	Ch #			ValueL	ValueH	Read Nor. OV limit
0xA0	Ch #			ValueL	ValueH	Read Margin UV limit
0xB0	Ch #			ValueL	ValueH	Read Margin OV limit
0xC0	STATUS Reg #			Status L	Status H	Read Sequencer Status
0xC8	0xCE					Clear STATUS Reg.
0xD0	Ch #			ValueL	ValueH	Read Analog Voltage
0xE0	0xEF					Power-Up Command
0xF0	0xDF					Power-Down Command

**TABLE A-3: STATUS REGISTER BIT DEFINITIONS**

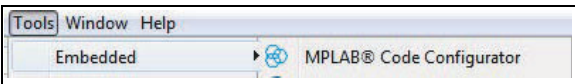
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Status 0 H	Vin UV	—	—	—	—	—	Mod10 UV	Mod9 UV
Status 0 L	Mod8 UV	Mod7 UV	Mod6 UV	Mod5 UV	Mod4 UV	Mod3 UV	Mod2 UV	Mod1 UV
Status 1 H	Vin OV	—	—	—	—	—	Mod10 OV	Mod9 OV
Status 1 L	Mod8 OV	Mod7 OV	Mod6 OV	Mod5 OV	Mod4 OV	Mod3 OV	Mod2 OV	Mod1 OV
Status 2 H	Fault On	Seq State	0	0	0	0	Mod10 Mrg En	Mod9 Mrg En
Status 2 L	Mod8 Mrg En	Mod7 Mrg En	Mod6 Mrg En	Mod5 Mrg En	Mod4 Mrg En	Mod3 Mrg En	Mod2 Mrg En	Mod1 Mrg En
Status 3 H	0	0	0	0	0	0	0	0
Status 3 L	System Retry Counter Value							
<b>Legend:</b> Modx = Module x MRG En = Margin Enable OV = Overvoltage UV = Undervoltage								

APPENDIX B: USING MPLAB® CODE CONFIGURATOR (MCC) TO ADD TIMER1 FUNCTION TO AN APPLICATION

To add a Timer1 function to an application using the MCC, follow these steps:

- 1. With the project opened in MPLAB® X, from the main menu select Tools → Microchip Embedded → MPLAB Code Configurator, as shown in [Figure B-1](#).

FIGURE B-1: SELECT MCC



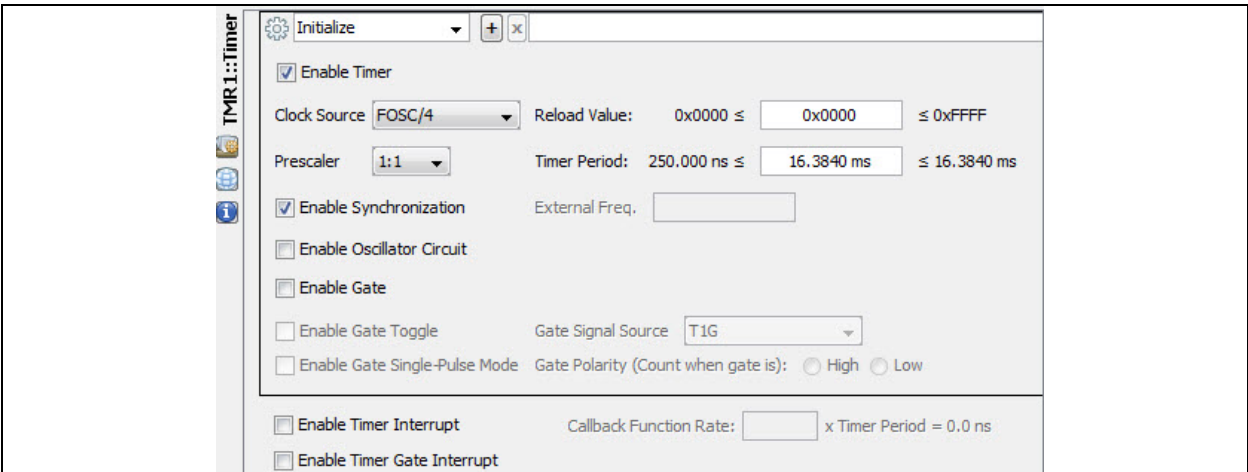
- 2. After MCC opens, from the Device Resources menu, select Timer1 to bring it into the Project Resources window, as illustrated in [Figure B-2](#).

FIGURE B-2: OPENING TIMER1 IN PROJECT RESOURCES WINDOW



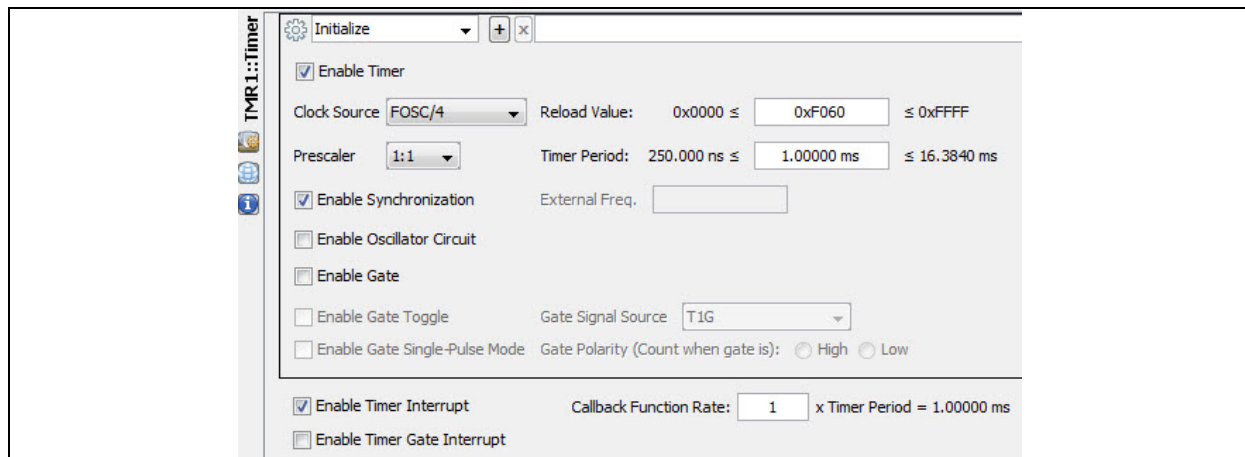
- 3. Click on TMR1 to edit and enter values to configure TMR1, as shown in [Figure B-3](#).

FIGURE B-3: TMR1 CONFIGURATION



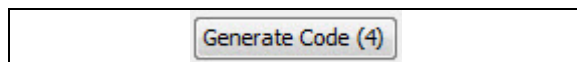
- To configure the TMR1 for a 1 mS interrupt using the 16 MHz internal clock, edit the “Time Period” to 1.0 mS, check the “Enable Timer Interrupt” box and edit the callback function rate to 1, as indicated in [Figure B-4](#) below.

**FIGURE B-4: TMR1 INTERRUPT CONFIGURATION**



- Click the **Generate Code** button, for MCC to create the code for TMR1.

**FIGURE B-5: GENERATE CODE BUTTON**



Using the MCC, this method can be used to initialize and create functions for the ADC, PWM, GPIO, Flash memory and I<sup>2</sup>C.

## APPENDIX C: MODIFYING MCC CREATED I<sup>2</sup>C SLAVE INTERRUPT FILE TO IMPLEMENT APPLICATION SERIAL INTERFACE

When MCC is used to generate an I<sup>2</sup>C Slave Interrupt file, the MCC generates an EEPROM example for the user in the `I2c.c` file. Since the EEPROM code is not used, this section will explain how to modify the existing code to accommodate the power sequencer I<sup>2</sup>C interface code.

The I<sup>2</sup>C command structure receives up to four bytes of information and transmits two bytes, so buffers need to be created and defined to hold that data:

- unsigned char RcvBuf[4];
- unsigned char TrmtBuf[2];

To access these buffers, appropriate pointers should also be defined as:

- unsigned char RcvPtr=0;
- unsigned char TrmtPtr=0;

The following example can be a possible scenario: presumably, only two commands are implemented: 0x20 (Set Start Time) and 0x24 (Read Start Time). All other commands follow the same structure as the two mentioned here and can be easily added to their respective state machine code.

The `ProcessRcvBuf()` function processes the received I<sup>2</sup>C data from the Master or from the GUI.

### EXAMPLE C-1: PARTIAL PROCESSRCVBUF () FUNCTION

```
void ProcessRcvBuf(void)
{
    unsigned int r;
    if (RcvPtr >= 4)
        RcvPtr = 0;
    switch (RcvBuf[0])
    {
        case 0x20: // Start time int value
            r = RcvBuf[3];
            r = r << 8;
            appmData[RcvBuf[1]].OnTime = r +
(int)RcvBuf[2];
            break;
        default:
            break;
    }
}
```

The `ProcessRcvBuf()` functions should be called in the I<sup>2</sup>C callback function after all serial data has been received. Depending on the command, two or four data bytes are received serially. The function `SetDataLength()` is called to determine the number of data bytes which will be received serially (either 2 or 4). Once the appropriate bytes are received, the `ProcessRcvBuf()` function is called and the serial data is processed. [Example C-2](#) shows where the `ProcessRcvBuf()` function is called.

### EXAMPLE C-2: PROCESSRCVBUF () FUNCTION CALL

```
case SLAVE_NORMAL_DATA:
default:
    // the master has written data to be processed
    RcvBuf[RcvPtr++] = I2C_slaveWriteData;
    if (RcvPtr == 1)
        SetDataLength();
    if (RcvPtr >= DataLength)
    {RcvPtr = 0;ProcessRcvBuf();}
    break;
```

The Transmit buffer is loaded as part of the `ProcessRcvBuf()` operation. Commands which need a transmit reply, load the transmit buffer appropriately. The actual transmit occurs in the callback function as shown in [Example C-3](#).

### EXAMPLE C-3: TRANSMIT CODE LOCATION

```
case I2C_SLAVE_READ_REQUEST:
    if (TrmtPtr == 0)
        LoadTrmtBuf();
    SSP1BUF = TrmtBuf[TrmtPtr++];
    if (TrmtPtr >= 2)
        TrmtPtr = 0;
    break;
```

With these modifications, the I<sup>2</sup>C interface will work as required in the application.

## APPENDIX D: ADDING AN ADDITIONAL MODULE TO THE EXISTING CODE

The code for the user has been made fairly modular so that the existing application can be expanded to include more modules. In this appendix, one extra module 4 is added to the existing 3 modules in the program.

Each module has two functions which are defined as below:

- `APP_M4_Initialize()` – Function which initializes the module 4 state machine and hardware/firmware
- `APP_MX_Task()` – Common function for all modules with the right Module Index (MI) when called.

In addition there is also a data structure which is defined as:

`appmData[4].XXXXX` – This data structure has a number of elements in it. Example: State, DACValue, PWMValue etc., which are the data elements for each module. They are resident in an array structure and the fourth element of that array will correspond to module 4.

In the file `app.h` the structure already exists, but one extra element needs to be added. A constant "AllModules" has been defined in `app.h` and this needs to be changed to 5, as shown in [Example D-1](#).

### EXAMPLE D-1: RE-DEFINE ALLMODULES

```
#define AllModules 5 // M1 to M4 + the Input voltage = 5.
```

By redefining the constant "AllModules", most parameters will appropriately be adjusted to accommodate the additional module 4 added to the application. Below are the parameters which need to be modified by the user.

**Note:** Module 0, or the zero element of this array, is assigned to the input voltage. The user does not need to copy/paste the entire structure, however, the structure needs to be increased by one element from 4 to 5, as shown above.

Module 4 has three hardware elements associated with it:

1. Enable signal I/O pin assigned by the user during MCC initialization.
2. PWM output to drive the Trim line on the module. This is assigned by the user during MCC initialization.
3. ADC channel to sample and convert the module output voltage. This ADC channel is defined by the user during MCC initialization.

These three hardware elements were assigned the following designations during the MCC initialization process:

1. EN4 for the Enable I/O pin
2. PWM4 for the trim voltage PWM
3. VO4 for the ADC channel number

These values need to be changed as the user copies and pastes an existing module in order to re-create an additional one.

Apart from the hardware elements, some firmware elements are also uniquely defined for each module, depending on the voltage of that new module. In this application note, module 4 is the 1.8V module and it has its own unique default values defined in the file `flash.as`. The user can copy an existing module's parameters in that file and then change the value with the new module parameter. The user can take module 1 values and copy and paste them just below module 3 values in the `flash.as` file.

In the `flash.as` file, copy and paste module 1 defines just below module 3 defines, as shown in [Example D-2](#).

### EXAMPLE D-2: COPY/PASTE MODULE 1 PARAMETERS

```
M3MarginUVLEQU 10 ; (0.2/5.0)*256

;Default voltage/time defs for M1 = 5V module
M1NormalOVLEQU 155 ; (5.5V*0.55/5.0V)*256; 0.55 =
Res. Divide Factor
M1NormalUVLEQU 127 ; (4.5V*0.55/5.0V)*256; 0.55 =
Res. Divide Factor
PWM1DC EQU 100 ;
M1OnTime EQU 250
M1OffTime EQU 2000
M1MarginOVLEQU 51 ; (1.2V/5.0V)*256
M1MarginUVLEQU 10 ; (0.2/5.0)*256
```

**Note:** Module 1 is the 5.0V module, so all the parameters associated with it are for the 5.0V voltage.

Then, the user can edit and modify the M1 parameter to M4 and the associated values for 1.8 volts to 5.0 volts, as per [Example D-3](#).

### EXAMPLE D-3: MODULE 1 TO MODULE 4 PARAMETER CHANGE

```
;voltage/time defs for M4 = 1.8V module
M4NormalOVLEQU 118 ; (3.8V/5.0V)*256;
M4NormalUVLEQU 67 ; (2.8V/5.0V)*256;
PWM4DC EQU 52 ;
M4OnTime EQU 2000
M4OffTime EQU 500
M4MarginOVLEQU 51 ; (1.2V/5.0V)*256
M4MarginUVLEQU 10 ; (0.2/5.0)*256
```

In the `app.h` file, the new module needs a nominal voltage output level defined as illustrated in [Example D-4](#) below.

## EXAMPLE D-4: DEFINE VO4NOMINAL

```
// Nominal voltage for M4 = 1.8 V module
#define VO4Nominal 93 // (1.8V/5.0V)*256
```

Also the VO4Nominal needs to be added to the array VONominal[AllModules] as the 5<sup>th</sup> element, as shown in [Example D-5](#) below.

## EXAMPLE D-5: ADD VO4NOMINAL TO ARRAY

```
unsigned char VONominal[AllModules] =
{0,VO1Nominal, VO2Nominal, VO3Nominal, VO4Nominal};
```

In addition, the M4 parameter defaults defined in flash.as need to be saved in the Buf[] array used during read/write operations on the Flash PM. Each element has its own unique Buffer Index (BI) and the user can copy an existing list for module 1 and edit it for module 4 as in [Example D-6](#) below.

## EXAMPLE D-6: DEFINE MODULE 4 BUFFER INDEX PARAMETERS

```
// Buf Index (BI) for M4 parameters
#define M4NormalOVLBI 24
#define M4NormalUVLBI 25
#define PWM4DCBI 26
#define M4OnTimeBI 27
#define M4OffTimeBI 28
#define M4MarginOVLBI 29
#define M4MarginUVLBI 30
```

**Note:** The Buf[] array is a 32 element buffer of unsigned 16-bit values.

In the app.h file, a prototype for APP\_M4\_Initialize() function needs to be created. The user can copy an existing prototype definition and then change the designator.

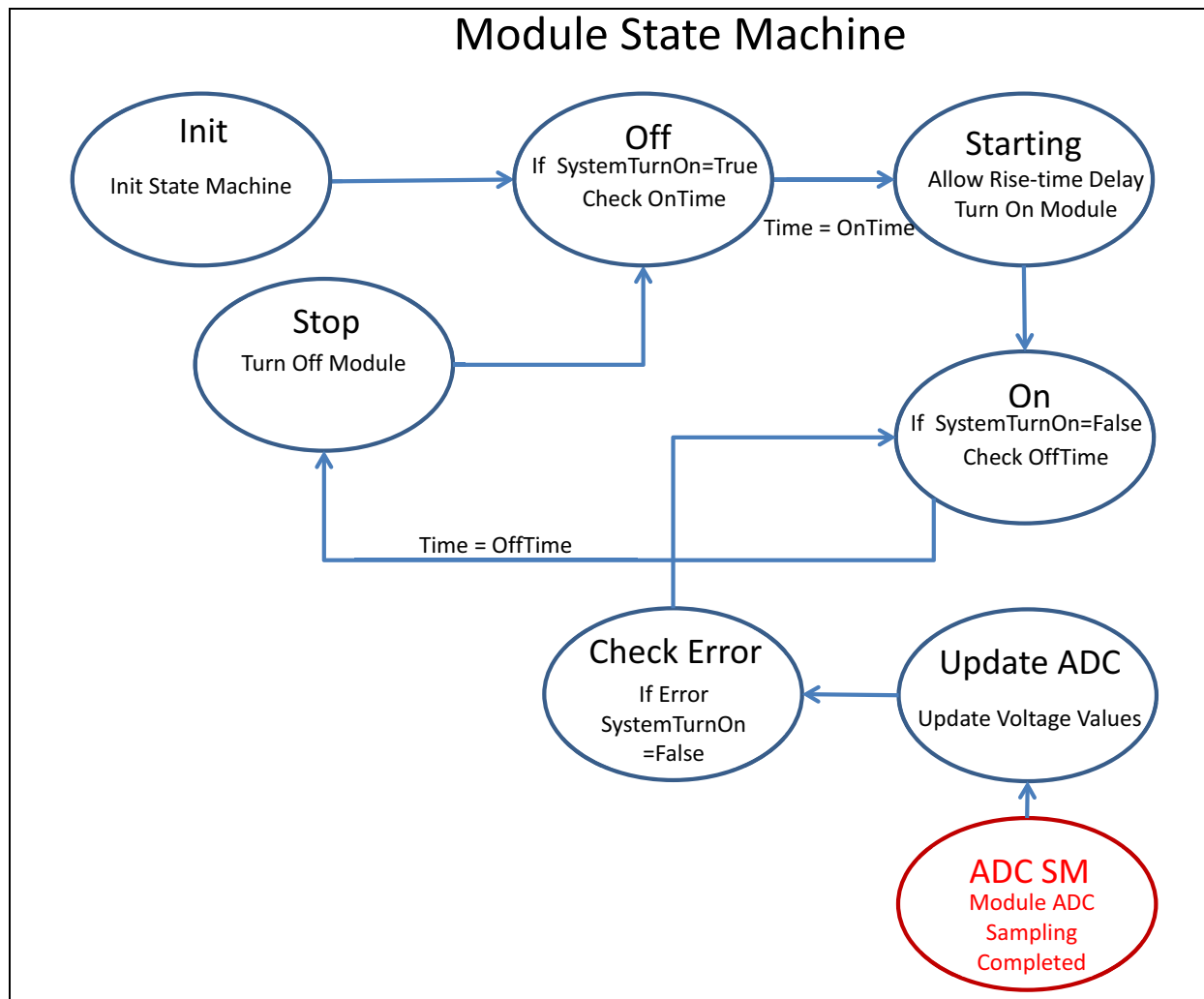
Once all the needed changes in the app.h file have been made, the user can switch to the app.c file to make the changes for the APP\_M4\_Initialize(), SetUVFault(), SetOVFault(), TurnOnMod() and TurnOffMod() functions. It is very easy to just copy and paste an existing module and then make the required module designator changes. Therefore, copy and paste the APP\_M1\_Initialize() function. Then change the 1 designators in the function to 4 to make it a APP\_M4\_Initialize() function. In the SetUVFault(), SetOVFault(), TurnOnMod() and TurnOffMod() functions, cut and paste an existing case statement for module 1, change the designator from 1 to 4 and create a new case statement for module 4.

Finally, in the file main.c, the user will have to make a function call for the APP\_M4\_Initialize() routine.

With all these changes complete and thoroughly checked, the user can build and program the part and verify proper operation. In case of an error, most likely a typo which has occurred during the copy and paste operation, it is highly recommended to check the new module edits and see that all the parameters are as desired.

## APPENDIX E: STATE MACHINE DIAGRAM FOR MODULE STATE MACHINE

FIGURE E-1: STATE MACHINE DIAGRAM



The states of module State Machine are listed below:

- **Init** – Initializes the State Machine and moves to OFF state
- **Off** – Check for `SystemTurnOn` flag. If true, then check for `OnTime`. When `Time = OnTime`, turn on module and move to Starting state
- **Starting** – A fixed delay to take care of the voltage rise time is allowed. Move to ON state
- **On** – Check for `SystemTurnOn` flag. If false, then check for `OffTime`. When `Time = OffTime`, move to Stop state.
- **Stop** – Turn off module and move to OFF state

- **CheckADC** – In the ADC State Machine, when the module voltage has been sampled, the module state is automatically moved to the CheckADC state. In this state the Voltage value is updated. Move to CheckError state.
- **CheckError** – A under/overvoltage check is done. If an error is detected, then the `SystemTurnOn` flag is set to false. Move to ON state.



---

**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 unless otherwise stated.

#### **Trademarks**

The Microchip name and logo, the Microchip logo, AnyRate, AVR, AVR logo, AVR Freaks, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, Helder, JukeBlox, KeeLoq, Klear, LANCheck, LINK MD, maXStylus, maXTouch, MediaLB, megaAVR, MOST, MOST logo, MPLAB, OptoLyzr, PIC, picoPower, PICSTART, PIC32 logo, Prochip Designer, QTouch, SAM-BA, SpyNIC, SST, SST Logo, SuperFlash, tinyAVR, UNI/O, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

ClockWorks, The Embedded Control Solutions Company, EtherSynch, Hyper Speed Control, HyperLight Load, IntelliMOS, mTouch, Precision Edge, and Quiet-Wire are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KlearNet, KlearNet logo, memBrain, Mindi, MiWi, motorBench, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, 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.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

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

© 2015-2019, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-4451-0

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

*Microchip received ISO/TS-16949:2009 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, KEELQ® 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.*

---

**NOTES:**

## Worldwide Sales and Service

### AMERICAS

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

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

**Austin, TX**  
Tel: 512-257-3370

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

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

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

**Detroit**  
Novi, MI  
Tel: 248-848-4000

**Houston, TX**  
Tel: 281-894-5983

**Indianapolis**  
Noblesville, IN  
Tel: 317-773-8323  
Fax: 317-773-5453  
Tel: 317-536-2380

**Los Angeles**  
Mission Viejo, CA  
Tel: 949-462-9523  
Fax: 949-462-9608  
Tel: 951-273-7800

**Raleigh, NC**  
Tel: 919-844-7510

**New York, NY**  
Tel: 631-435-6000

**San Jose, CA**  
Tel: 408-735-9110  
Tel: 408-436-4270

**Canada - Toronto**  
Tel: 905-695-1980  
Fax: 905-695-2078

### ASIA/PACIFIC

**Australia - Sydney**  
Tel: 61-2-9868-6733

**China - Beijing**  
Tel: 86-10-8569-7000

**China - Chengdu**  
Tel: 86-28-8665-5511

**China - Chongqing**  
Tel: 86-23-8980-9588

**China - Dongguan**  
Tel: 86-769-8702-9880

**China - Guangzhou**  
Tel: 86-20-8755-8029

**China - Hangzhou**  
Tel: 86-571-8792-8115

**China - Hong Kong SAR**  
Tel: 852-2943-5100

**China - Nanjing**  
Tel: 86-25-8473-2460

**China - Qingdao**  
Tel: 86-532-8502-7355

**China - Shanghai**  
Tel: 86-21-3326-8000

**China - Shenyang**  
Tel: 86-24-2334-2829

**China - Shenzhen**  
Tel: 86-755-8864-2200

**China - Suzhou**  
Tel: 86-186-6233-1526

**China - Wuhan**  
Tel: 86-27-5980-5300

**China - Xian**  
Tel: 86-29-8833-7252

**China - Xiamen**  
Tel: 86-592-2388138

**China - Zhuhai**  
Tel: 86-756-3210040

### ASIA/PACIFIC

**India - Bangalore**  
Tel: 91-80-3090-4444

**India - New Delhi**  
Tel: 91-11-4160-8631

**India - Pune**  
Tel: 91-20-4121-0141

**Japan - Osaka**  
Tel: 81-6-6152-7160

**Japan - Tokyo**  
Tel: 81-3-6880-3770

**Korea - Daegu**  
Tel: 82-53-744-4301

**Korea - Seoul**  
Tel: 82-2-554-7200

**Malaysia - Kuala Lumpur**  
Tel: 60-3-7651-7906

**Malaysia - Penang**  
Tel: 60-4-227-8870

**Philippines - Manila**  
Tel: 63-2-634-9065

**Singapore**  
Tel: 65-6334-8870

**Taiwan - Hsin Chu**  
Tel: 886-3-577-8366

**Taiwan - Kaohsiung**  
Tel: 886-7-213-7830

**Taiwan - Taipei**  
Tel: 886-2-2508-8600

**Thailand - Bangkok**  
Tel: 66-2-694-1351

**Vietnam - Ho Chi Minh**  
Tel: 84-28-5448-2100

### EUROPE

**Austria - Wels**  
Tel: 43-7242-2244-39  
Fax: 43-7242-2244-393

**Denmark - Copenhagen**  
Tel: 45-4450-2828  
Fax: 45-4485-2829

**Finland - Espoo**  
Tel: 358-9-4520-820

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

**Germany - Garching**  
Tel: 49-8931-9700

**Germany - Haan**  
Tel: 49-2129-3766400

**Germany - Heilbronn**  
Tel: 49-7131-67-3636

**Germany - Karlsruhe**  
Tel: 49-721-625370

**Germany - Munich**  
Tel: 49-89-627-144-0  
Fax: 49-89-627-144-44

**Germany - Rosenheim**  
Tel: 49-8031-354-560

**Israel - Ra'anana**  
Tel: 972-9-744-7705

**Italy - Milan**  
Tel: 39-0331-742611  
Fax: 39-0331-466781

**Italy - Padova**  
Tel: 39-049-7625286

**Netherlands - Drunen**  
Tel: 31-416-690399  
Fax: 31-416-690340

**Norway - Trondheim**  
Tel: 47-7288-4388

**Poland - Warsaw**  
Tel: 48-22-3325737

**Romania - Bucharest**  
Tel: 40-21-407-87-50

**Spain - Madrid**  
Tel: 34-91-708-08-90  
Fax: 34-91-708-08-91

**Sweden - Gothenberg**  
Tel: 46-31-704-60-40

**Sweden - Stockholm**  
Tel: 46-8-5090-4654

**UK - Wokingham**  
Tel: 44-118-921-5800  
Fax: 44-118-921-5820