AN2345

Four-Channel PIC16-Based Power Sequencer Using MIC45116 Power Modules

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

INTRODUCTION

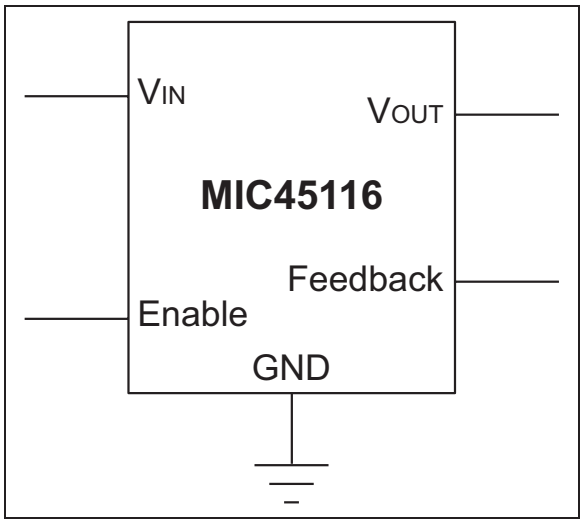
Power sequencers are commonly used in system-level board designs where multiple power supplies are enabled in a sequential manner. Typically, systems using a power sequencer have different components on them, which require different power supply voltages and power levels. The sequence of enabling the 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 Microchip MIC45116 Power Module (PM). These PMs have multiple pins; however, the main pins are: Input/Output Voltage, Ground, Enable and Feedback. Users can select any number of PMs for their system. The software was written in a modular format to accommodate up to ten PMs. PMs can be added/removed to meet specific needs. An enhanced core PIC16F18344 device has been selected as the MCU for this application. If more program memory or data memory is needed, then PIC16F18344 can easily be replaced with a pin-compatible PIC16F18345, which has double the program and data memory of the PIC16F18344.

HARDWARE DESIGN CONSIDERATIONS

Microchip MIC45116 Power Modules (PM)

PMs are power supply blocks, sold with specific voltage, current and power capabilities. The PMs used in our design are the MIC45116 series made by Microchip (Figure 1). Each PM has multiple pins; however, only five are important for this design: Input Voltage (VIN), Output Voltage (VOUT), Ground (GND), Enable input and Feedback. The Enable signal is active-high, and when enabled, a voltage will appear at the VOUT pin of the PM. This voltage is directly related to the RTOP and RBOTTOM resistors connected to the Feedback pin of the PM. The user can select the RTOP and RBOTTOM resistors by using the formula in Figure 2. In addition, the Feedback pin allows for a ±X mV margining of the output voltage (where X mV is defined by the user). A DC voltage at the Feedback pin is needed and it is supplied by the PIC® microcontroller. In this design, the DC voltage is provided by a PWM-controlled DAC output through an RC-based filter circuit (Figure 3). If no margining is required in the design, then only the external resistors are used at the Feedback pin. There is no need for a DAC voltage at the Feedback pin, and the PWM with its associated RC circuit and software in the PIC16 MCU is eliminated. Refer to the *"MIC45116 20V/6A DC/DC Power Module"* data sheet (DS20005571) for more details on RTOP and RBOTTOM resistor values. The typical RBOTTOM resistor values are calculated with a fixed RTOP value of 10k, as shown in Table 1.

FIGURE 1: MIC45116 POWER MODULE BLOCK DIAGRAM

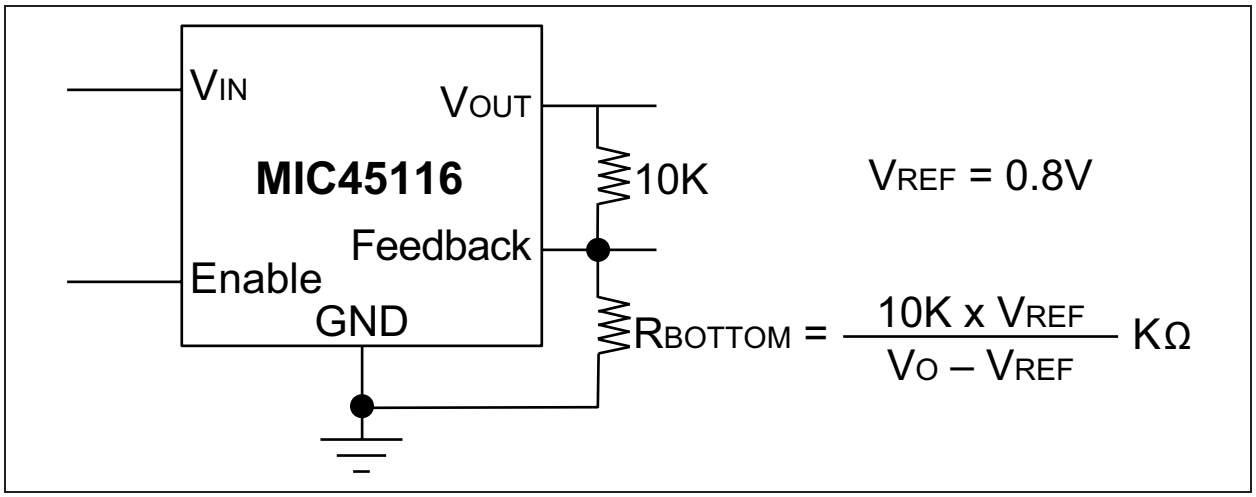


The MIC45116 typical RBOTTOM values for voltages of 1.8V, 2.5V, 3.3V and 5.0V are listed in Table 1 below:

TABLE 1: RBOTTOM VALUES FOR TYPICAL VOUT VOLTAGES

VOUT	VIN	RTOP	RBOTTOM
1.8V	5-0V	10K	8.06K
2.5V	5-20V	10K	4.75K
3.3V	5-20V	10K	3.24K
5.0V	7-20V	10K	1.91K

FIGURE 2: RBOTTOM FORMULA FOR VOUT



Power-up Sequencing

A PIC16F18344 device, 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:

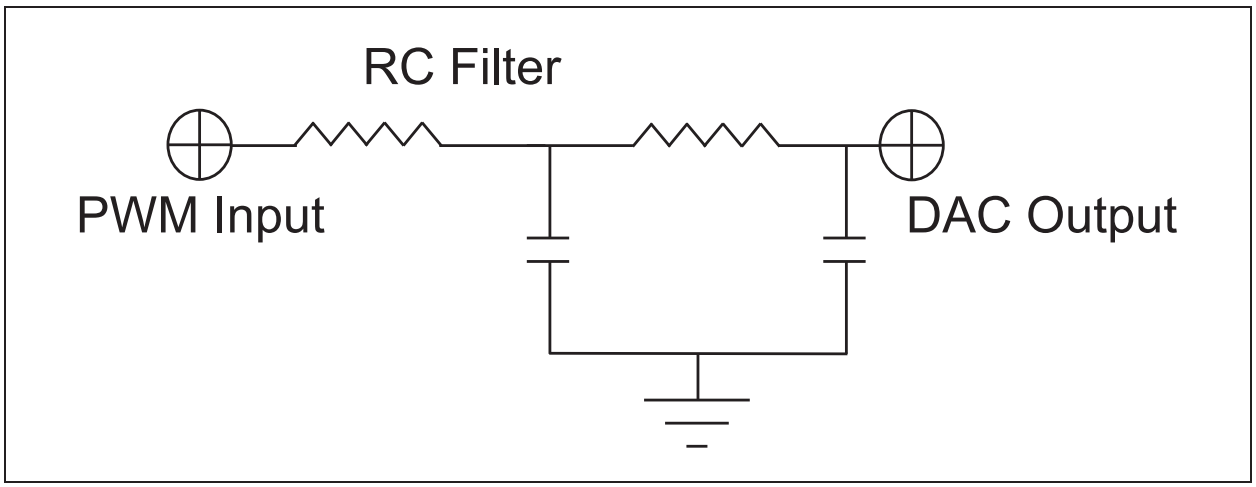
1. A serial command using the I^2C interface.
2. Pressing the push button switch, S1.

Each PM is sequenced on at a set time interval, from zero to 16,393 ms (16.4s), with 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 ms. 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 devices. The on/off sequence can also be started/stopped using the serial I^2C Graphical User Interface (GUI).

Feedback

When the PM is turned on, the Feedback voltage is automatically generated based on the preassigned resistor values. An additional Feedback voltage is created by using a PWM output from the PIC16 microcontroller. The PWM output is then sent through an RC filter. The output of the filter creates a DAC voltage, which is sent to the Feedback pin of the PM (Figure 3). The output of the PM is monitored to within ±1 Least Significant bit (LSb) of the set voltage, using a 10-bit A/D Converter (ADC) on the PIC16 microcontroller. Each PM voltage is averaged over 16 readings to give a 14-bit value. Only the Most Significant 8 bits of this value are used to reference the VOUT voltage value of each PM. The reference voltage of the ADC is VDD or 5.0V. For example, if the PM output voltage is 2.5V, then the accuracy of the measurement would be (2.5/5.0)/256 = 2 mV. The output voltage is monitored for a failure (i.e., the output voltage falls below or above the min/max output limits for each VOUT). If a failure occurs, the PMs are automatically shut down as per the power-down sequence.

FIGURE 3: PWM INPUT TO DAC OUTPUT THROUGH FILTER



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:

1. Serial command from the I^2C bus.
2. Pressing the push button switch, S1.
3. Any Fault condition on the PMs or the input voltage.

Each PM is sequenced off at a set time interval, from zero to 16,393 ms (16.4s), with 1 ms accuracy. For example, PM4 can be shut down 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 bits). This value is independent from the on-time value. This value is compared to a 16-bit counter value incremented every ms. If the two are equal, then the corresponding PM is turned off. The off times are user-selectable and are programmed 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 specified by the user. Typically, a user may specify two or three retries. After all the retries return a failure, the system is shut down and a Fault condition is signaled. Using the I^2C GUI, the user can figure out which PM 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 GUI, and then retry a successful power-up sequence.

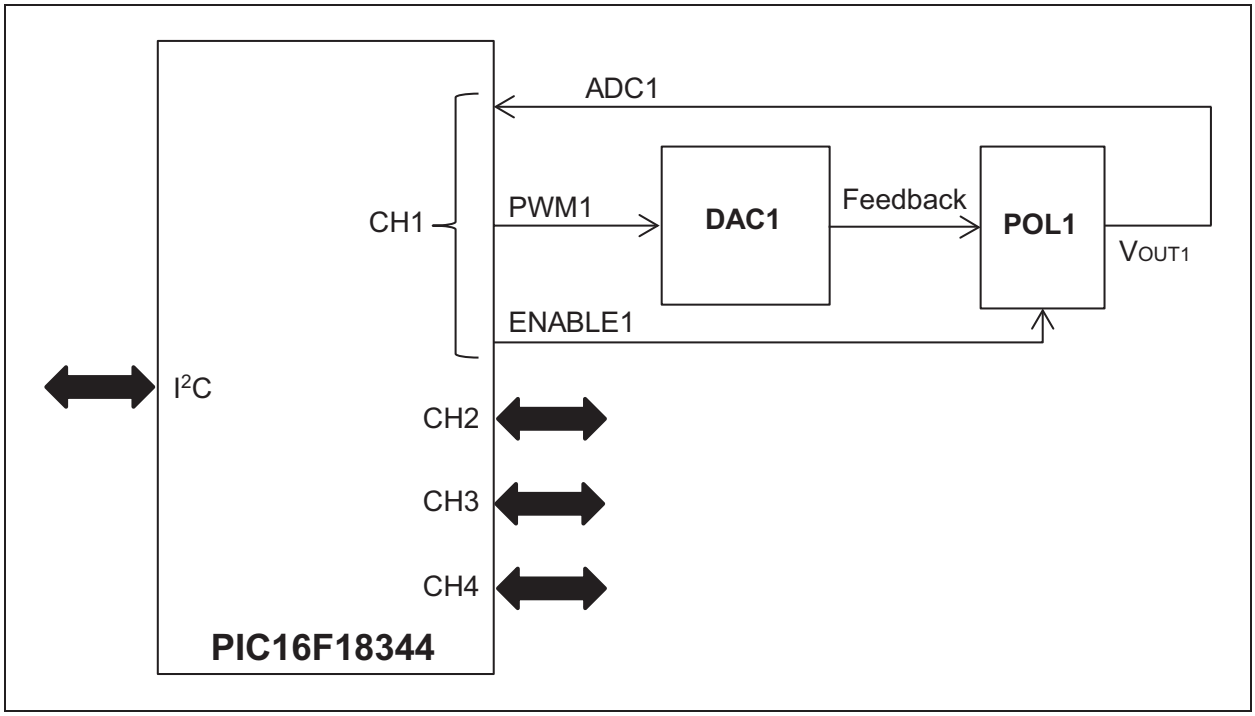
MCU Requirements

The MCU requirements fall into four categories:

1. I/O pins to enable/disable the PMs.
2. ADC input to sample the PM V_{OUT} voltage and the input voltage.
3. PWM output to generate the DAC output for the Feedback voltage.
4. Communications using I^2C.

Since four channels of PMs are used in this design, at least four I/O lines for the enable/disable function are required. Also needed are: four ADC channels, plus one for the input voltage, four PWM outputs, two lines for I^2C, $\overline{MCLR}$, SW1 (KEY input), V_{DD}, V_{SS} and programming pins. In total, 20 pins are needed. Thus, a PIC16F18344 MCU was selected for the design. Figure 4 shows the system block diagram.

FIGURE 4: SYSTEM BLOCK DIAGRAM



The MCU is powered by 5.0V and 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/Os 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 Feedback voltage requirements can also be adjusted. If the user wants to control the PMs using the external resistor only, then the DAC voltage is not required. The Feedback voltage is preset using the external resistor values, and the PWM requirement as well, as the filter for the DAC is eliminated. The software to drive the PWM and DAC will also be eliminated.

Voltage Limits

Each PM has its own normal over/undervoltage limit. PMs also have margin over/undervoltage limits for the Feedback voltage supplied through the DAC.

NORMAL OVER/UNDERVOLTAGE LIMITS

As specified, these limits are the absolute limits for each PM. The output voltage of each PM is monitored, and if at any time, the output voltage goes under or above these limits, a Fault is signaled and the power is sequenced off. A retry is initiated after a power Fault occurs. After a certain number (specified by the user) of automatic retries, the system is shut down until the user clears the system with a serial command, using the I^2C 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/UNDERVOLTAGE LIMITS

Each PM has a user-defined margin over/undervoltage 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 using the GUI. The DAC value can be changed within the bounds of the over/undervoltage margin limits.

Note:	The trim voltage is not monitored by the PIC16 microcontroller. Only the PWM duty cycle value is compared with the limit set by the user. This value is defined as an 8-bit value.

I^2C Communication Protocol and Commands

An I^2C Slave interface is implemented on the MCU for the serial communications with an external I^2C GUI. The command structure for the I^2C interface is specified in **Appendix A: "Serial Command Definitions and Structures"**. On the demo board, an MCP2221 I^2C to mini-USB interface chip is provided. This interface can be implemented by the user in their own design or a different I^2C interface can be implemented.

I^2C 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. The PMBus™ protocol is not supported at this time.

FIRMWARE IMPLEMENTATION

Peripheral Code Generated Using the Microchip Code Configurator (MCC)

All the firmware for the I/Os, Timer1, ADC, PWM, Flash memory and I²C peripherals has been created and initialized using the free MPLAB® Code Configurator (MCC) software. The system setup also uses the MCC software. The entire firmware was developed using the MPLAB® 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 PIC16FXXXX device, the recreation 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 IDE project. As an example, **Appendix B: "Using MPLAB® Code Configurator (MCC) to Add Timer1 Function to an Application"** shows a step-by-step process using the MCC to generate code for Timer1.

I²C Slave Interrupt Firmware

The I²C 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 I²C used in the power sequencer program. This method is quick and very easy to use without any prior knowledge, or setup details on the I²C 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 initialization 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 an over/undervoltage Fault

Note: Figure E-1 in Appendix E shows the 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 values and 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 the 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 Modules 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 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 of the Modules 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 over/undervoltage 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 over and undervoltage 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 States

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^2C 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 – Initializes the 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 PIC16F18344. 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), these system parameters are also saved:

- V_{IN} Undervoltage Limit
- V_{IN} 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 the Flash PM.

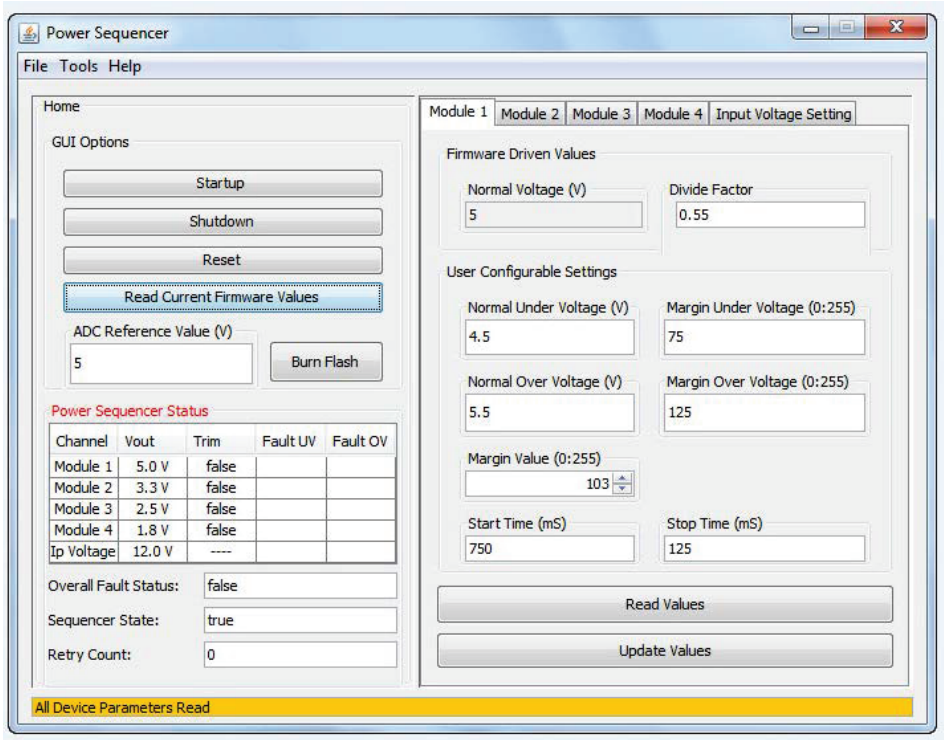
Power Sequencer GUI

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 Graphical 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. 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^2C 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.

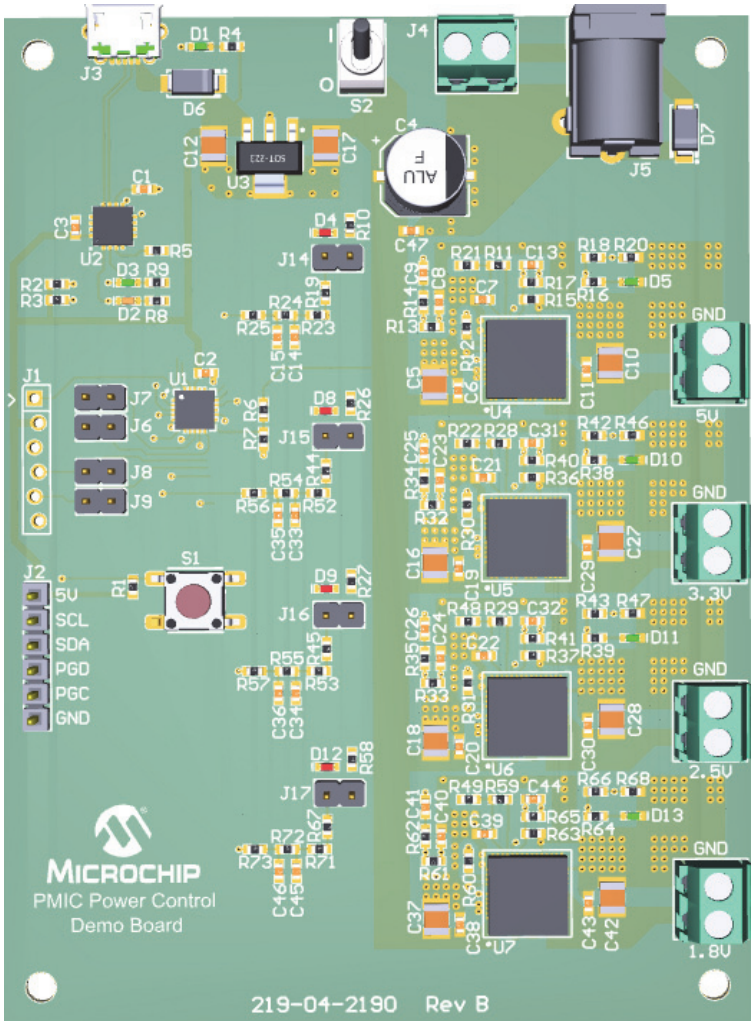
In each module tab, the DAC value can be incremented or decremented using the up/down arrow located at one side of the DAC value box. The value increments or decrements, and if the module is on, then the output voltage will be read and updated. To see the voltage change, more than one increment or decrement 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 over/undervoltage 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 PIC16FXXXX 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

A.1 I²C Serial Command Format

A.1.1 NOTATIONS

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

A.2 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.

A.3 Read Commands

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

A.3.1 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 I²C serial instructions mentioned in this section), a Program Flash operation will permanently burn the values onto the Flash memory of the PIC16F18344 device. A subsequent Reset or Power-on Reset will retrieve values from the Flash memory. This operation is recommended after the user updates all the module parameters.

A.3.2 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.

A.3.3 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.

A.3.4 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.

A.3.5 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.

A.3.6 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.

A.3.7 COMMAND 0x38: READ DAC MARGIN VOLTAGE

Writing a 0x38 command, followed by the module number, will set up 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.

A.3.8 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.

A.3.9 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.

A.3.10 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.

A.3.11 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.

A.3.12 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.

A.3.13 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.

A.3.14 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.

A.3.15 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.

A.3.16 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.

A.3.17 COMMAND 0xB0: READ MARGIN OVERVOLTAGE LIMIT

Writing a 0xB0, 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.

A.3.18 COMMAND 0xC0: READ POWER SEQUENCER STATUS

Writing a 0xC0, 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 shown in the following Table A-1.

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

A.3.19 COMMAND 0xC8: CLEAR STATUS ERROR FLAGS AS WELL AS RETRY NUMBER

Writing a 0xC8, 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.

A.3.20 COMMAND 0xD0: READ MODULE ANALOG VOLTAGE

Writing a 0xD0, 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.

A.3.21 COMMAND 0xE0: POWER-UP

Writing a 0xE0, 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 over/undervoltage Fault that had occurred.

A.3.22 COMMAND 0xF0: POWER-DOWN

Writing a 0xF0, 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 Normal UV Limit
0x90	Ch #	—	—	ValueL	ValueH	Read Normal 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 Register
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							

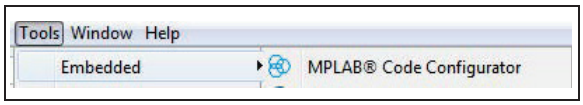
Legend: Modx = Module x
MRG En = Margin Enable
OV = Overvoltage
UV = Undervoltage

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 IDE, from the main menu, select Tools → Microchip Embedded → MPLAB Code Configurator, as shown in Figure B-1.

FIGURE B-1: SELECT MCC

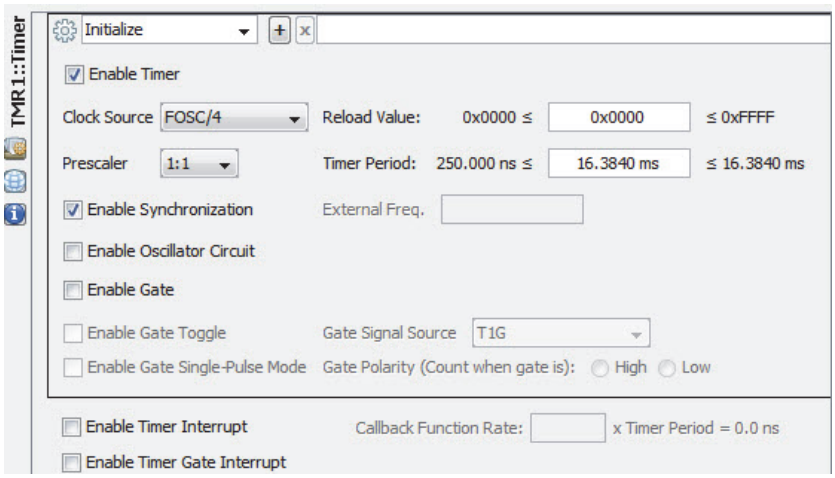


2. After MCC opens, from the Device Resources menu, select Timer1 to bring it into the Project Resources window, as illustrated in Figure B-2.

FIGURE B-2: OPENING TIMER1 IN PROJECT RESOURCES WINDOW

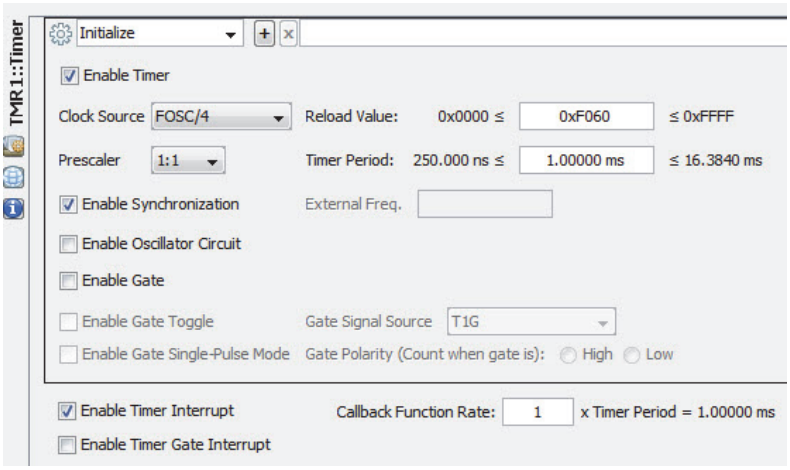
3. Click on TMR1 to edit and enter values to configure Timer1, as shown in Figure B-3.

FIGURE B-3: TMR1 CONFIGURATION



4. 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



5. 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^2C.

APPENDIX C: MODIFYING MCC CREATED I²C SLAVE INTERRUPT FILE TO IMPLEMENT APPLICATION SERIAL INTERFACE

When MCC is used to generate an I²C 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 I²C interface code.

The I²C 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 I²C 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 I²C 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 I²C interface will work as required in the application.

APPENDIX D: ADDING 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: REDEFINE ALL MODULES

```
#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 the signal I/O pin assigned by the user during MCC initialization.
2. A PWM output to drive the trim line on the module. This is assigned by the user during MCC initialization.
3. An 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 recreate 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

```
M3MarginUVL  EQU 10   ;(0.2/5.0)*256

;Default voltage/time defs for M1 = 5V module
M1NormalOVL  EQU 155  ;(5.5V*0.55/5.0V)*256; 0.55 =
                        Res. Divide Factor
M1NormalUVL  EQU 127  ;(4.5V*0.55/5.0V)*256; 0.55 =
                        Res. Divide Factor
PWM1DC       EQU 100  ;
M1OnTime     EQU 250
M1OffTime    EQU 2000
M1MarginOVL  EQU 51   ;(1.2V/5.0V)* 256
M1MarginUVL  EQU 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
M4NormalOVL  EQU 118   ;(3.8V/5.0V)*256;
M4NormalUVL  EQU 67    ;(2.8V/5.0V)*256;
PWM4DC       EQU 52    ;
M4OnTime     EQU 2000
M4OffTime    EQU 500
M4MarginOVL  EQU 51    ;(1.2V/5.0V)* 256
M4MarginUVL  EQU 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 V04Nominal

```
// 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 5th element, as shown in Example D-5 below.

EXAMPLE D-5: ADD V04Nominal 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 the APP_M4_Initialize() function needs to be created. The user can copy an existing prototype definition and then change the designator.

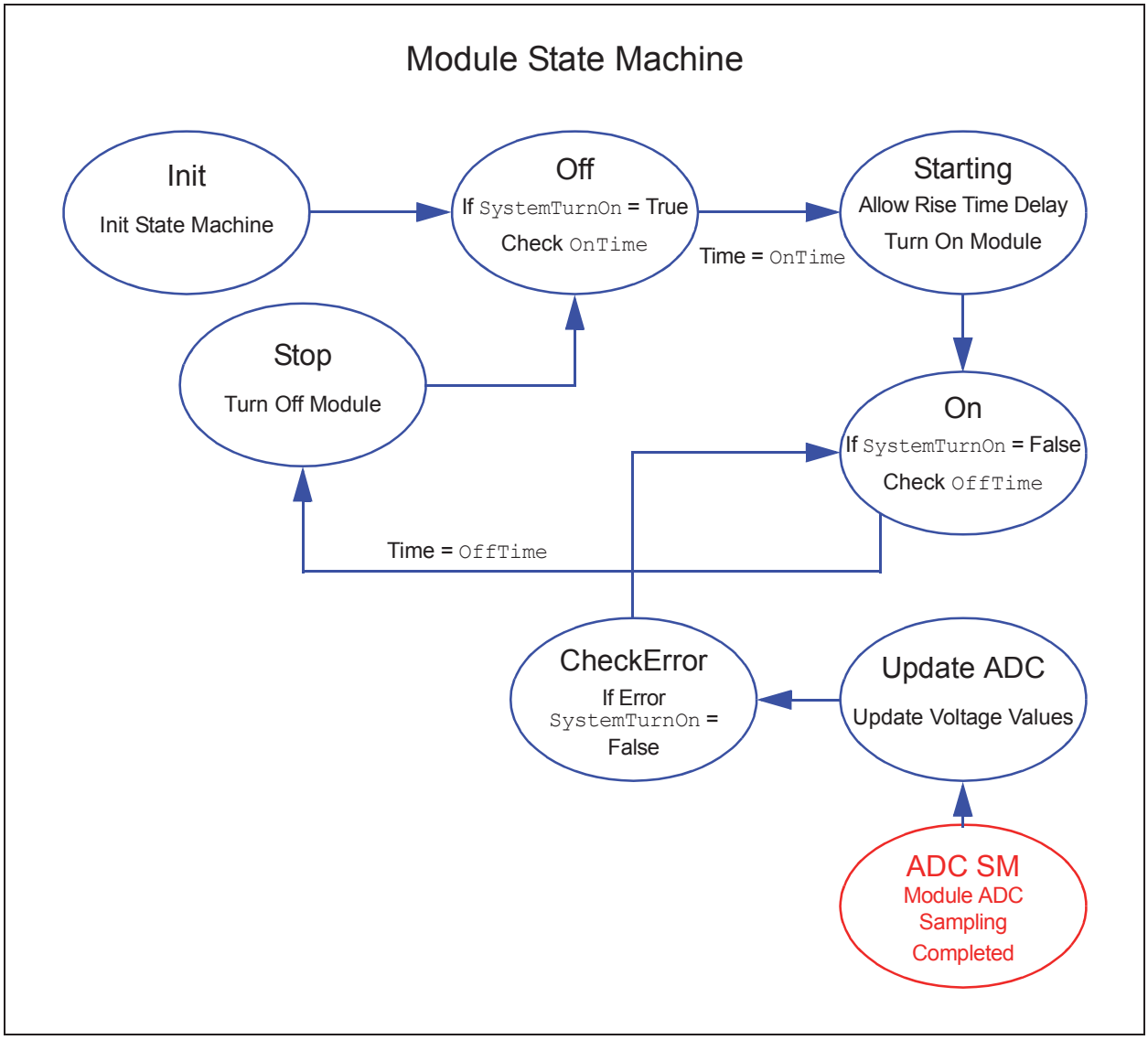
Once all the needed changes in the app.h file have been made, the user can switch to the app.c file to make the changes for the APP_M4_Initialize(), SetUVFault(), SetOVFault(), TurnOnMod() and TurnOffMod() functions. It is very easy to just copy and paste an existing module, and then make the required 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 the 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 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: DIAGRAM FOR THE MODULE STATE MACHINE

FIGURE E-1: MODULE STATE MACHINE DIAGRAM



The states of the module state machine are listed below:

- **Init** – Initializes the state machine and moves to the OFF state.
- **Off** – Check for `SystemTurnOn` flag. If true, then check for `OnTime`. When Time = `OnTime`, turn on module and move to the Starting state.
- **Starting** – A fixed delay to take care of the voltage rise time is allowed. Move to the ON state.
- **On** – Check for the `SystemTurnOn` flag. If false, then check for `OffTime`. When Time = `OffTime`, move to the Stop state.
- **Stop** – Turn off module and move to the 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 the CheckError state.
- **CheckError** – An over/undervoltage check is done. If an error is detected, then the `SystemTurnOn` flag is set to false. Move to the ON state.

APPENDIX F: POWER SEQUENCER SCHEMATICS

FIGURE F-1: POWER SEQUENCER SCHEMATICS REV. C PAGE 1 OF 2

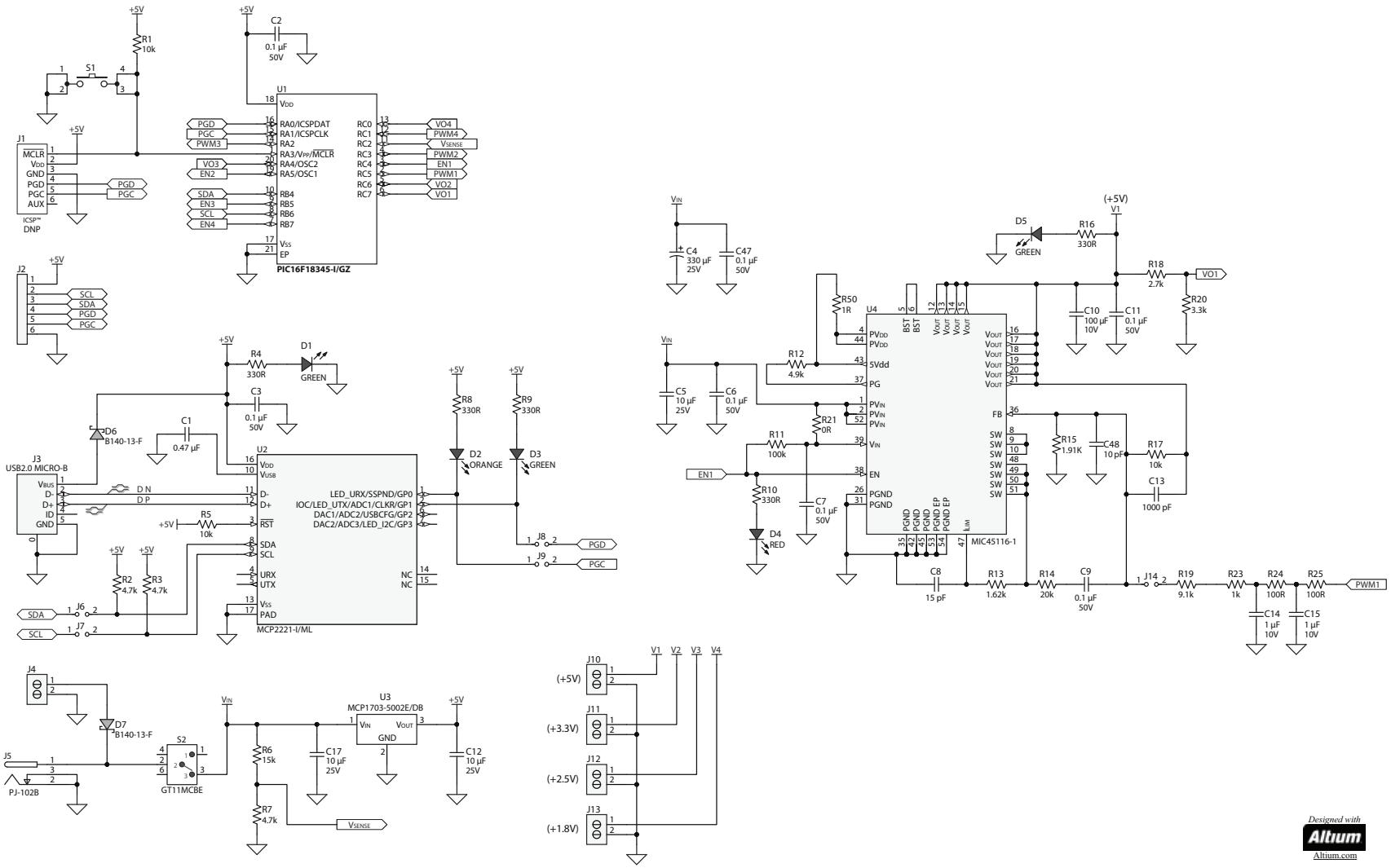
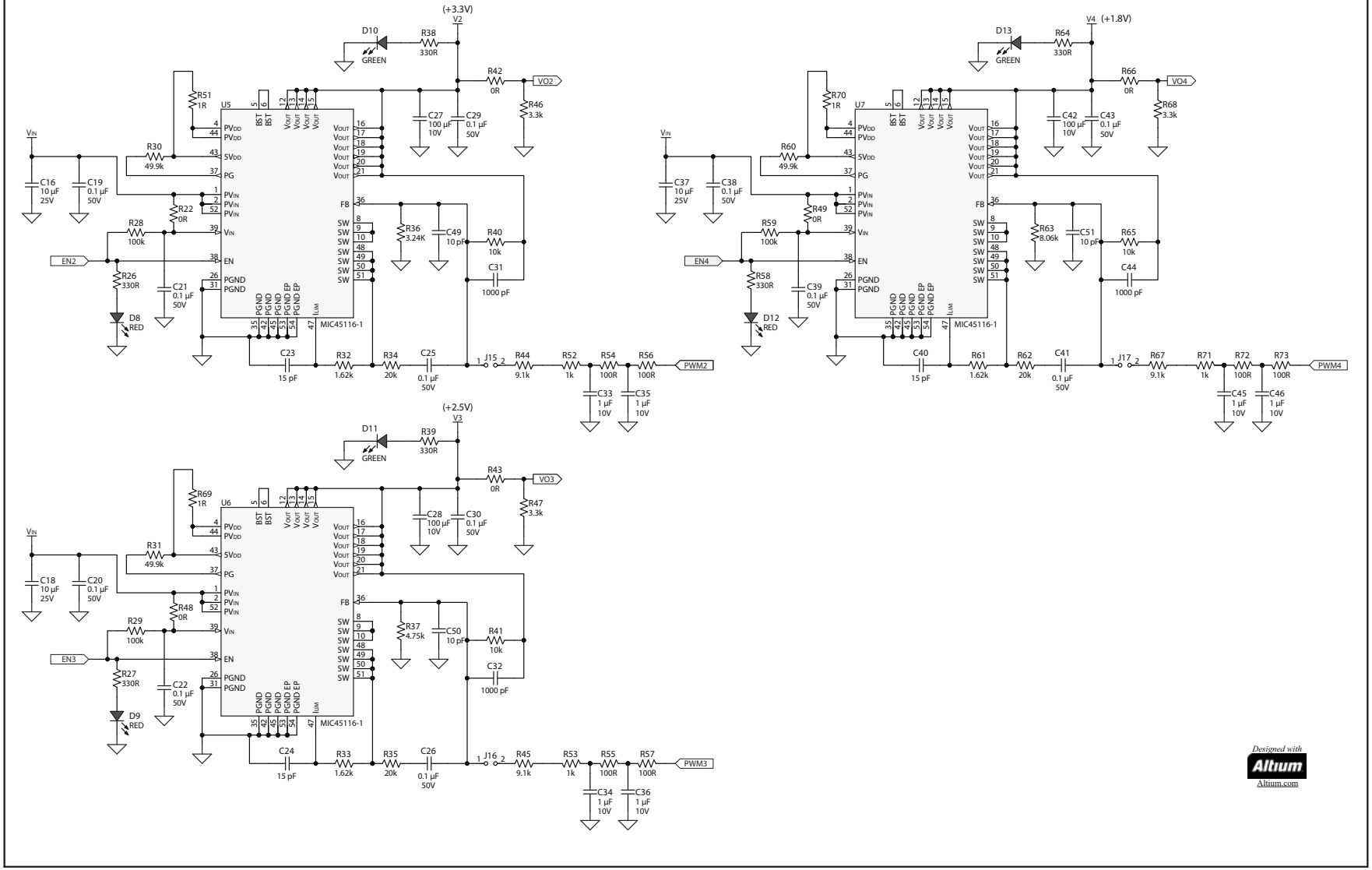


FIGURE F-2: POWER SEQUENCER SCHEMATICS REV. C PAGE 2 OF 2



NOTES:

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.

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, 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.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949

Trademarks

ISBN: 978-1-5224-1388-2

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

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

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

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

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

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

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

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

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

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

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

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

China - Shanghai
Tel: 86-21-3326-8000
Fax: 86-21-3326-8021

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

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

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

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

ASIA/PACIFIC

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

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

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

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

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

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

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

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

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

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

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

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

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

EUROPE

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

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

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

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

France - Saint Cloud
Tel: 33-1-30-60-70-00

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: 47-7289-7561

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