面向嵌入式工程师的MPLAB® XC32用户指南

简介

本文档提供了5个适用于32位器件和MPLAB® XC32 C编译器的代码示例。读者需要掌握一些单片机和C编程语言的相关知识。

1. 点亮或熄灭LED
2. 使用延时函数使LED闪烁
3. 使用中断作为延时在LED上显示递增计数
4. 使用ADC在LED上显示电位器值（MPLAB Harmony）
5. 使用ADC在LED上显示电位器值（MCC）
6. 在LED上显示闪存值（MPLAB Harmony）
7. 在LED上显示闪存值（MCC）

A 在MPLAB X IDE中运行代码
B 获取软件和硬件

1. 点亮或熄灭LED

本示例将使用PIC32MX470F512L接插模块（Plug-In Module，PIM）交替点亮Explorer 16/32板上的LED。更多信息，请参见第B节“获取软件和硬件”。

```
// PIC32MX470F512L Configuration Bit Settings
// 'C' source line config statements  ◄——— 请参见第1.1节

// DEVCFG3
// USERID = No Setting
#pragma config FSRSSEL = PRIORITY_7 // Shadow Register Set Priority 7
#pragma config PMDL1WAY = ON // Peripheral Module - One Reconfig
#pragma config IOL1WAY = ON // Peripheral Pin Select - One Reconfig
#pragma config FUSBIDIO = ON // USB USID Selection - Port Function
#pragma config FVBUSONIO = ON // USB VBUS ON Selection - Port Function

// DEVCFG2
#pragma config FPLLIDIV = DIV_12 // PLL Input Divider - 12x
#pragma config FPLLMUL = MUL_24 // PLL Multiplier - 24x
#pragma config UPLLIDIV = DIV_12 // USB PLL Input Divider - 12x
#pragma config UPLLEN = OFF // USB PLL Disabled and Bypassed
#pragma config FPLLODIV = DIV_256 // Sys PLL Output Divide by 256

// DEVCFG1
#pragma config FNOSC = FRCDIV // Oscillator - Fast RC Osc w/Div-by-N
#pragma config FSOSCEN = ON // Secondary Oscillator Enabled
#pragma config IESO = OFF // Internal/External Switch Over Disabled
#pragma config POSCMOD = OFF // Primary Oscillator Disabled
#pragma config OSCIOFNC = OFF // CLKO on OSCO Pin Disabled
#pragma config FPBDIV = DIV_8 // Peripheral Clock Divisor: Sys_Clk/8
#pragma config FCKSM = CSDCMD // Clock Switch Disable, FSCM Disabled
#pragma config WDTPS = PS1048576 // WDT Postscaler 1:1048576
#pragma config WINDIS = OFF // Watchdog Timer is in Non-Window Mode
#pragma config FWDTEN = OFF // WDT Disabled (SWDTEN Control)
#pragma config FWDTWINSZ = WINSZ_25 // Watchdog Timer Window 25%

// DEVCFG0
#pragma config DEBUG = OFF // Background Debugger Disabled
#pragma config JTAGEN = OFF // JTAG Disabled
#pragma config ICESEL = ICS_PGx2 // ICE/ICD Comm Channel PGEC2/PGED2
#pragma config PWP = OFF // Program Flash Write Protect Disabled
#pragma config BWP = OFF // Boot Flash Write Protect Disabled
#pragma config CP = OFF // Code Protect Disabled

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>  ◄——— 请参见第1.2节

#define LEDS_ON_OFF 0x55  ◄——— 请参见第1.3节

int main(void) {

    // Port A access ◄——— 请参见第1.4节

    TRISA = 0x0000;       // set all port bits to be output
    LATA = LEDS_ON_OFF;   // write to port latch

    return 0;
}
```

1.1 配置位

Microchip器件具有配置寄存器，其中的位可用于使能和/或设置器件功能。

> **注：** 如果未正确设置配置位，器件将无法运行，或至少不按预期运行。

1.1.1 要设置的配置位

请特别注意以下几项：

- **振荡器选择**——该项必须与硬件的振荡器电路匹配。如果该选择有误，*器件时钟可能无法运行*。通常情况下，开发板使用高速晶振。示例中的相关代码如下：
  ```
  #pragma config FNOSC = PRI
  #pragma config POSCMOD = HS
  ```
- **看门狗定时器**——建议在必要时禁止该定时器。这样可防止*意外复位*。示例中的相关代码如下：
  ```
  #pragma config FWDTEN = OFF
  ```
- **代码保护**——在必要时关闭代码保护。这样可确保*器件存储器可完全访问*。示例中的相关代码如下：
  ```
  #pragma config CP = OFF
  ```

使用其他32位器件（而非本示例中使用的MCU）时，可能需要设置不同的配置位。请参见具体器件数据手册了解相应配置位的编号和功能。可使用部件编号在http://www.microchip.com上搜索相应的数据手册。

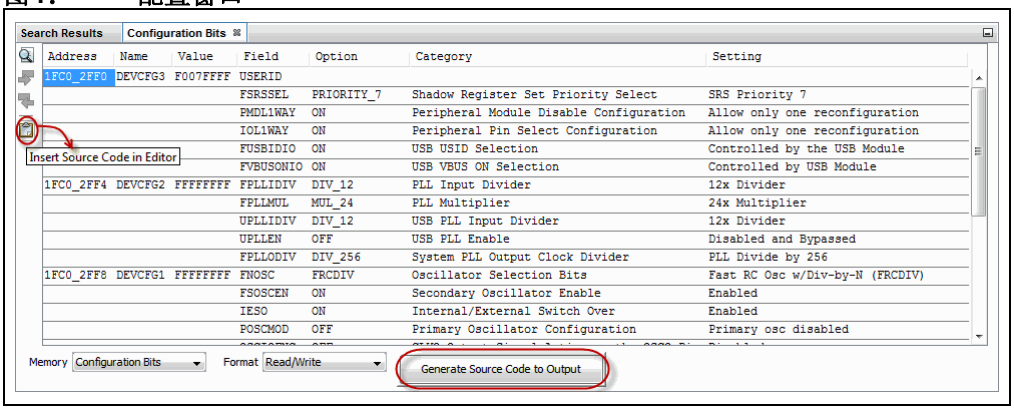
有关每款器件可用配置位的更多信息，请参见MPLAB XC32安装路径下的如下文件：

MPLAB XC32 Installation Directory/docs/PIC32ConfigSet.html

1.1.2 设置配置位的方法

在MPLAB X IDE中，可使用Configuration Bits（配置位）窗口查看和设置这些位。请选择*Window>PIC Memory Views>Configuration Bits*（窗口>PIC存储器视图>配置位）打开该窗口。

图1： 配置窗口



完成所需设置后，单击main()前您想要放置pragma伪指令的位置处的代码，并单击**Insert Source Code in Editor**（在编辑器中插入源代码）图标。或者，您也可单击**Generate Source Code to Output**（生成要输出的源代码），然后将pragma伪指令从Output（输出）窗口复制到您的代码中。

1.2 头文件<xc.h>

本头文件允许源文件中的代码访问编译器或器件特定的功能。可在MPLAB XC32安装目录的pic32mx/include子目录下找到本头文件和其他头文件。

编译器将根据您选择的器件设置相应的宏，以使xc.h指向正确的器件特定头文件。请勿将器件特定的头文件包含在您的代码中，否则将导致代码不可移植。

1.3 为LED值定义宏

如下一节所述，要写入LED的值已赋值给描述性宏（LEDS_ON_OFF），即LED D3、D5、D7和D9将点亮，LED D4、D6、D8和D10将熄灭。有关Explorer 16/32文档（包括电路板原理图）的链接，请参见**第B.5节“获取并设置Explorer 16/32板”**。

1.4 端口访问

器件数字I/O引脚可能与外设I/O引脚复用。为确保当前仅使用数字I/O，需禁止其他外设。为此，可使用代表外设寄存器及其位的预定义C变量。这些变量列于编译器目录的pic32mx/include/proc下的器件特定头文件中。关于哪些外设共用哪些引脚的信息，请参见具体器件的数据手册。

对于本节中的示例，端口A引脚与默认禁止的外设复用。另外，端口A没有模拟I/O，因此所有引脚默认为数字I/O。对于端口具有模拟I/O的器件，必须禁止模拟功能（例如，使用ADxPCFT寄存器）来确保数字I/O操作。

器件引脚连接至器件的数字I/O端口（PORT）或锁存器（LAT）寄存器。示例中使用LATA。为变量portValue赋值，以便随后赋值给锁存器。

```
LATA = portValue;   // write to port latch
```

此外，还有一个寄存器用于指定引脚的方向是输入还是输出，它称为TRIS寄存器。本节的示例中使用TRISA。将位设置为0可将引脚设为输出，将位设置为1可将引脚设为输入。对于本示例：

```
TRISA = 0x0000;   // set all port bits to be output
```

2. 使用延时函数使LED闪烁

本示例在上一示例代码的基础上进行修改。本代码不仅点亮LED，还将使LED交替闪烁。已添加的代码以红色显示。

```
// PIC32MX470F512L Configuration Bit Settings
// 'C' source line config statements

// DEVCFG3
// USERID = No Setting
#pragma config FSRSSEL = PRIORITY_7 // Shadow Register Set Priority 7
#pragma config PMDL1WAY = ON // Peripheral Module - One Reconfig
#pragma config IOL1WAY = ON // Peripheral Pin Select - One Reconfig
#pragma config FUSBIDIO = ON // USB USID Selection - Port Function
#pragma config FVBUSONIO = ON // USB VBUS ON Selection - Port Function

// DEVCFG2
#pragma config FPLLIDIV = DIV_12 // PLL Input Divider - 12x
#pragma config FPLLMUL = MUL_24 // PLL Multiplier - 24x
#pragma config UPLLIDIV = DIV_12 // USB PLL Input Divider - 12x
#pragma config UPLLEN = OFF // USB PLL Disabled and Bypassed
#pragma config FPLLODIV = DIV_256 // Sys PLL Output Divide by 256

// DEVCFG1
#pragma config FNOSC = FRCDIV // Oscillator - Fast RC Osc w/Div-by-N
#pragma config FSOSCEN = ON // Secondary Oscillator Enabled
#pragma config IESO = OFF // Internal/External Switch Over Disabled
#pragma config POSCMOD = OFF // Primary Oscillator Disabled
#pragma config OSCIOFNC = OFF // CLKO on OSCO Pin Disabled
#pragma config FPBDIV = DIV_8 // Peripheral Clock Divisor: Sys_Clk/8
#pragma config FCKSM = CSDCMD // Clock Switch Disable, FSCM Disabled
#pragma config WDTPS = PS1048576 // WDT Postscaler 1:1048576
#pragma config WINDIS = OFF // Watchdog Timer is in Non-Window Mode
#pragma config FWDTEN = OFF // WDT Disabled (SWDTEN Control)
#pragma config FWDTWINSZ = WINSZ_25 // Watchdog Timer Window 25%

// DEVCFG0
#pragma config DEBUG = OFF // Background Debugger Disabled
#pragma config JTAGEN = OFF // JTAG Disabled
#pragma config ICESEL = ICS_PGx2 // ICE/ICD Comm Channel PGEC2/PGED2
#pragma config PWP = OFF // Program Flash Write Protect Disabled
#pragma config BWP = OFF // Boot Flash Write Protect Disabled
#pragma config CP = OFF // Code Protect Disabled

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>

#define LEDS_ON_OFF 0x55
#define LEDS_OFF_ON 0xAA

void delay (void)
{
    int n = 50000;
    while(n>0) {n--;}
}
```

```
int main(void) {

    // Port A access
    TRISA = 0x0;    // set all port bits to be output

    while(1) {   ◄──── 请参见第2.1节

        LATA = LEDS_ON_OFF; // write to port latch

        // delay value change   ◄──── 请参见第2.2节
        delay();

        LATA = LEDS_OFF_ON; // write to port latch

        // delay value change
        delay();
    }
    return -1;
}
```

2.1 while()循环和变量值

要使端口A上的LED发生变化，需在循环的第一部分为变量portValue赋一个值，然后在循环的第二部分为其赋一个相反值。使用了while(1) { }执行循环。

如果主函数返回，则意味着存在错误，因为while循环通常不应结束。因此，将返回-1。

2.2 delay()函数

由于执行速度在大多数情况下都会导致LED的闪烁速度超出人眼的识别能力，因此需要降低执行速度。函数delay()在main()之前进行声明和定义，在main()代码中调用两次。

> **注：** 不要使用编译器优化，否则延时循环将被删除（使用-O0）。有关延迟代码执行的其他方法，请参见下一个示例。

3. 使用中断作为延时在LED上显示递增计数

本示例在上一示例代码的基础上进行修改。尽管上一示例中的延时函数对于降低循环执行速度很有用，但是会在程序中引入停滞时间。为避免这一问题，将使用内核定时器中断。在每次发生中断时，变量值都会增加并显示在LED上。

本示例中使用了内核定时器，因为它在所有PIC32 MCU中保持一致，并以恒定速率（每2个系统时钟周期）递增，而且无需设置预/后分频比。其他器件定时器也可以用于延时，但是如果其他模块也使用同一个定时器，则必须小心。已添加的代码以红色显示。

```
// PIC32MX470F512L Configuration Bit Settings
// 'C' source line config statements

// DEVCFG3
// USERID = No Setting
#pragma config FSRSSEL = PRIORITY_7 // Shadow Register Set Priority 7
#pragma config PMDL1WAY = ON // Peripheral Module - One Reconfig
#pragma config IOL1WAY = ON // Peripheral Pin Select - One Reconfig
#pragma config FUSBIDIO = ON // USB USID Selection - Port Function
#pragma config FVBUSONIO = ON // USB VBUS ON Selection - Port Function

// DEVCFG2
#pragma config FPLLIDIV = DIV_12 // PLL Input Divider - 12x
#pragma config FPLLMUL = MUL_24 // PLL Multiplier - 24x
#pragma config UPLLIDIV = DIV_12 // USB PLL Input Divider - 12x
#pragma config UPLLEN = OFF // USB PLL Disabled and Bypassed
#pragma config FPLLODIV = DIV_256 // Sys PLL Output Divide by 256

// DEVCFG1
#pragma config FNOSC = FRCDIV // Oscillator - Fast RC Osc w/Div-by-N
#pragma config FSOSCEN = ON // Secondary Oscillator Enabled
#pragma config IESO = OFF // Internal/External Switch Over Disabled
#pragma config POSCMOD = OFF // Primary Oscillator Disabled
#pragma config OSCIOFNC = OFF // CLKO on OSCO Pin Disabled
#pragma config FPBDIV = DIV_8 // Peripheral Clock Divisor: Sys_Clk/8
#pragma config FCKSM = CSDCMD // Clock Switch Disable, FSCM Disabled
#pragma config WDTPS = PS1048576 // WDT Postscaler 1:1048576
#pragma config WINDIS = OFF // Watchdog Timer is in Non-Window Mode
#pragma config FWDTEN = OFF // WDT Disabled (SWDTEN Control)
#pragma config FWDTWINSZ = WINSZ_25 // Watchdog Timer Window 25%

// DEVCFG0
#pragma config DEBUG = OFF // Background Debugger Disabled
#pragma config JTAGEN = OFF // JTAG Disabled
#pragma config ICESEL = ICS_PGx2 // ICE/ICD Comm Channel PGEC2/PGED2
#pragma config PWP = OFF // Program Flash Write Protect Disabled
#pragma config BWP = OFF // Boot Flash Write Protect Disabled
#pragma config CP = OFF // Code Protect Disabled

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>
#include <cp0defs.h>
#include <sys/attribs.h>
```

请参见第3.1节

```
// CORE_TICK_RATE = FOSC/2/TOGGLES_PER_SEC
// FOSC/2 = Core timer clock frequency = 8MHz/2=4MHz
// TOGGLES_PER_SEC = Toggle LED x times per second; x=5
#define CORE_TICK_RATE          800000u

// Interrupt function  <-- 请参见第3.2节

void __ISR(_CORE_TIMER_VECTOR, IPL2SOFT) CTInterruptHandler(void)
{
    // static variable for permanent storage duration
    static unsigned char portValue = 0;
    // variables for Compare period
    unsigned long ct_count = _CP0_GET_COUNT();
    unsigned long period = CORE_TICK_RATE;

    // write to port latch
    LATA = portValue++;
    // update the Compare period
    period += ct_count;
    _CP0_SET_COMPARE(period);
    // clear the interrupt flag
    IFS0CLR = _IFS0_CTIF_MASK;
}

int main(void) {

    unsigned int stat_gie, cause_val;

    // Disables interrupts by clearing the global interrupt enable bit
    // in the STATUS register.
    stat_gie = __builtin_disable_interrupts();

    // Port A access
    TRISA = 0x0; // set all port bits to be output
    LATA = 0x0;  // clear all bits

    // Configure the core timer  <-- 请参见第3.3节
    // clear the CP0 Count register
    _CP0_SET_COUNT(0);
    // set up the period in the CP0 Compare register
    _CP0_SET_COMPARE(CORE_TICK_RATE);
    // halt core timer and program at a debug breakpoint
    _CP0_BIC_DEBUG(_CP0_DEBUG_COUNTDM_MASK);

    // Set up core timer interrupt  <-- 请参见第3.4节
    // clear core timer interrupt flag
    IFS0CLR = _IFS0_CTIF_MASK;
    // set core time interrupt priority of 2
    IPC0CLR = _IPC0_CTIP_MASK;
    IPC0SET = (2 << _IPC0_CTIP_POSITION);
    // set core time interrupt subpriority of 0
    IPC0CLR = _IPC0_CTIS_MASK;
    IPC0SET = (0 << _IPC0_CTIS_POSITION);
    // enable core timer interrupt
    IEC0CLR = _IEC0_CTIE_MASK;
    IEC0SET = (1 << _IEC0_CTIE_POSITION);
```

```
    // set the CP0 Cause register Interrupt Vector bit
    cause_val = _CP0_GET_CAUSE();
    cause_val |= _CP0_CAUSE_IV_MASK;
    _CP0_SET_CAUSE(cause_val);

    // enable multi-vector interrupts
    INTCONSET = _INTCON_MVEC_MASK;

    // enable global interrupts
    __builtin_enable_interrupts();

    while(1);

    return -1;
}
```

3.1 其他头文件

除xc.h外，还需包含其他头文件：cp0defs.h（用于CP0宏）和sys/attribs.h（用于ISR宏）。

3.2 中断函数

本示例中使用ISR宏__ISR(v,IPL)将CTInterruptHandler()变为中断函数，其中v是内核定时器的中断向量，IPL是中断优先级（2），现场保护方法（通过软件）表示为IPL2SOFT。有关ISR的更多信息，请参见《MPLAB® XC32 C/C++编译器用户指南》（DS50001686G_CN）的“中断”一章。

在中断函数中，计数器portValue递增并显示在LED上。

要清除中断，必须写入CP0比较寄存器。比较寄存器中的值将与内核定时器的未来值进行比较，以产生下一个中断。内核定时器的当前值可以通过_CP0_GET_COUNT()找到。

最后，中断标志被清除。

3.3 内核定时器设置

32位内核定时器初始时设置为0。比较寄存器设置为CORE_TICK_RATE的初始值。当内核定时器达到比较值时，将触发中断。

此外，为辅助调试，已将内核定时器设置为在断点处暂停。

有关内核定时器的更多信息，请参见《PIC32系列参考手册》的第2章“带M4K®内核的器件的CPU”（DS61113E_CN）。

3.4 内核定时器中断

设置内核定时器中断需要遵循几个步骤。

在main代码起始处，使用__builtin_disable_interrupts()禁止全局中断。在while(1)循环之前，使用__builtin_enable_interrupts()允许全局中断。

内核定时器中断标志使用器件头文件（通过xc.h访问）中的宏清除。

中断优先级和子优先级使用器件宏进行设置。此处的优先级必须与中断函数的优先级（2）匹配。

内核定时器和多向量中断使用器件宏来允许。CP0 Cause寄存器中的中断向量位使用器件和CP0宏进行设置。

4. 使用ADC在LED上显示电位器值（MPLAB HARMONY）

本示例与示例3使用相同的器件以及端口A LED。但在本示例中，将使用演示板上电位器的值通过端口B（RB2/AN2）提供模数转换器（Analog-to-Digital Converter，ADC）输入，然后进行转换并显示在LED上。

代码无需手动编写，而是使用MPLAB Harmony生成。下载MPLAB Harmony集成软件框架，网址为：

http://www.microchip.com/mplab/mplab-harmony

MPLAB Harmony配置器（MPLAB Harmony Configurator，MHC）是用于MPLAB Harmony GUI设置的MPLAB X IDE插件。该插件可在MPLAB X IDE菜单*Tools>Plugins*（工具>插件）的**Available Plugins**（可用插件）选项卡下安装。有关如何安装插件的更多信息，请参见MPLAB X IDE帮助。

本示例基于Windows®中以下路径下的`adc_pot`示例：

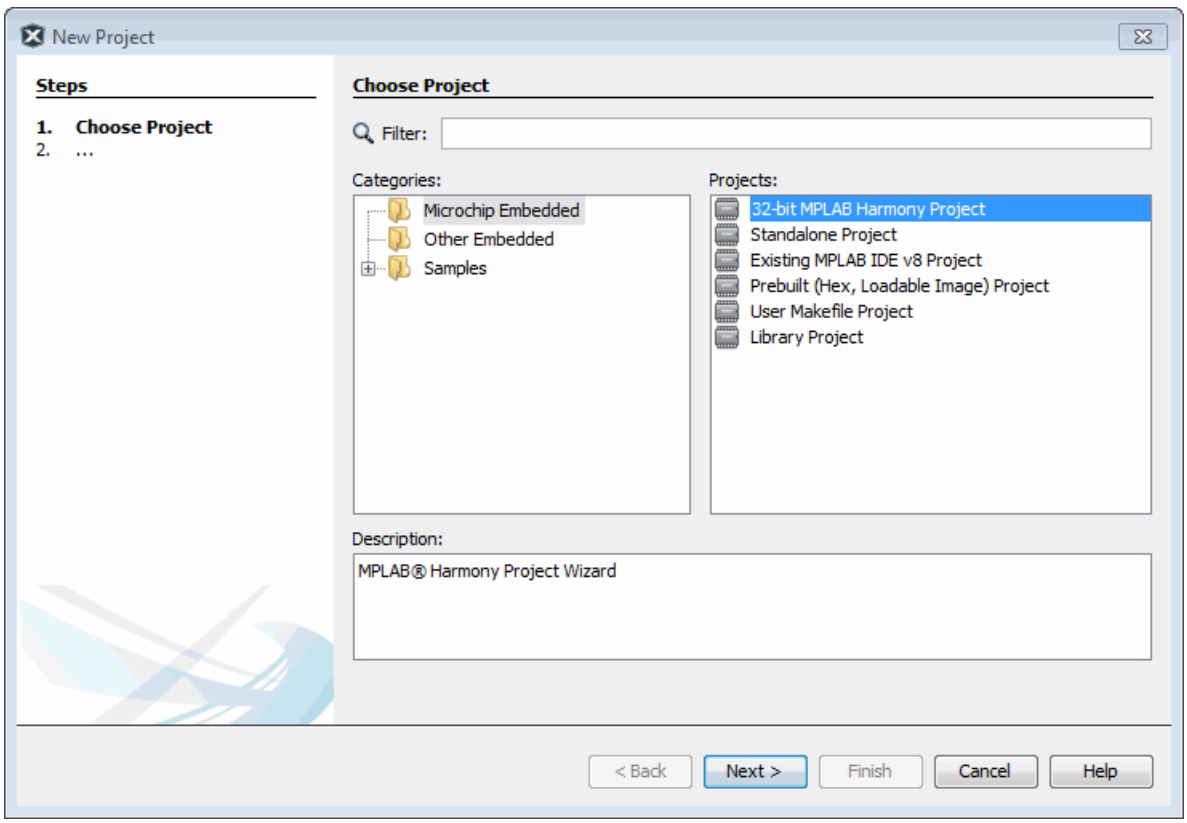
```
C:\microchip\harmony\v1_10\apps\examples\peripheral\adc\adc_pot
```

4.1 在MPLAB X IDE中创建一个MPLAB Harmony项目

为本示例设置以下对话框。

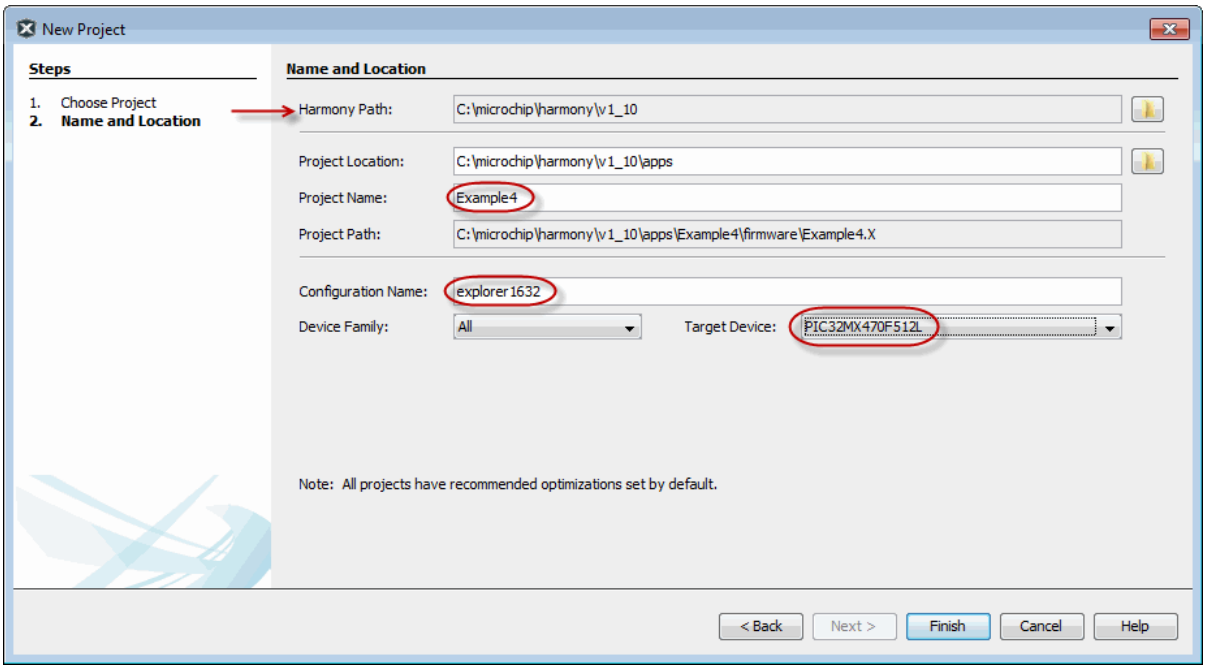
在MPLAB X IDE中，选择*File>New Project*（文件>新建项目）。

图2： 新建MPLAB® HARMONY项目——步骤1



确保Harmony路径指向您安装的MPLAB Harmony。

图3： 新建MPLAB® HARMONY项目——步骤2



4.2 配置MPLAB Harmony项目

根据您的项目设置，MHC打开时将提供部分时钟信息。以蓝色突出显示的文本表示更改。在本例中，不要对时钟设置进行任何更改。

配置ADC驱动程序（如图6所示）和板级支持包（Board Support Package，BSP）（如图7所示）。

图4： MPLAB® HARMONY项目和MHC

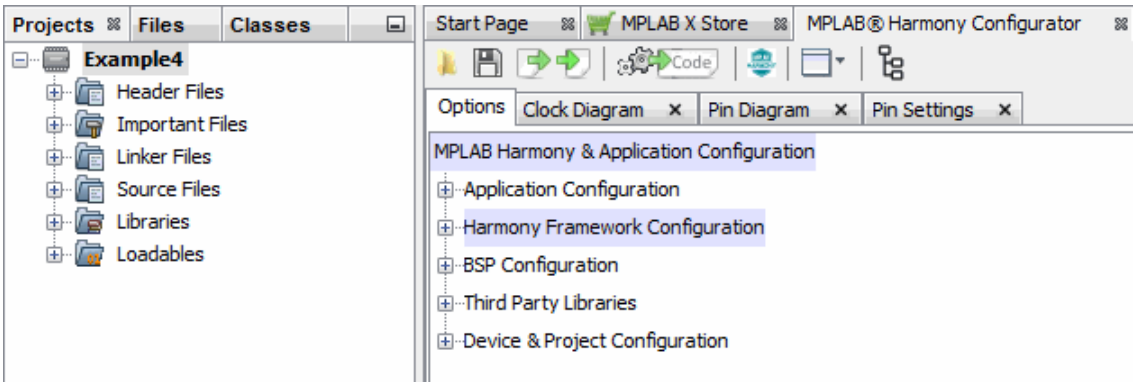


图5： HARMONY框架配置——时钟

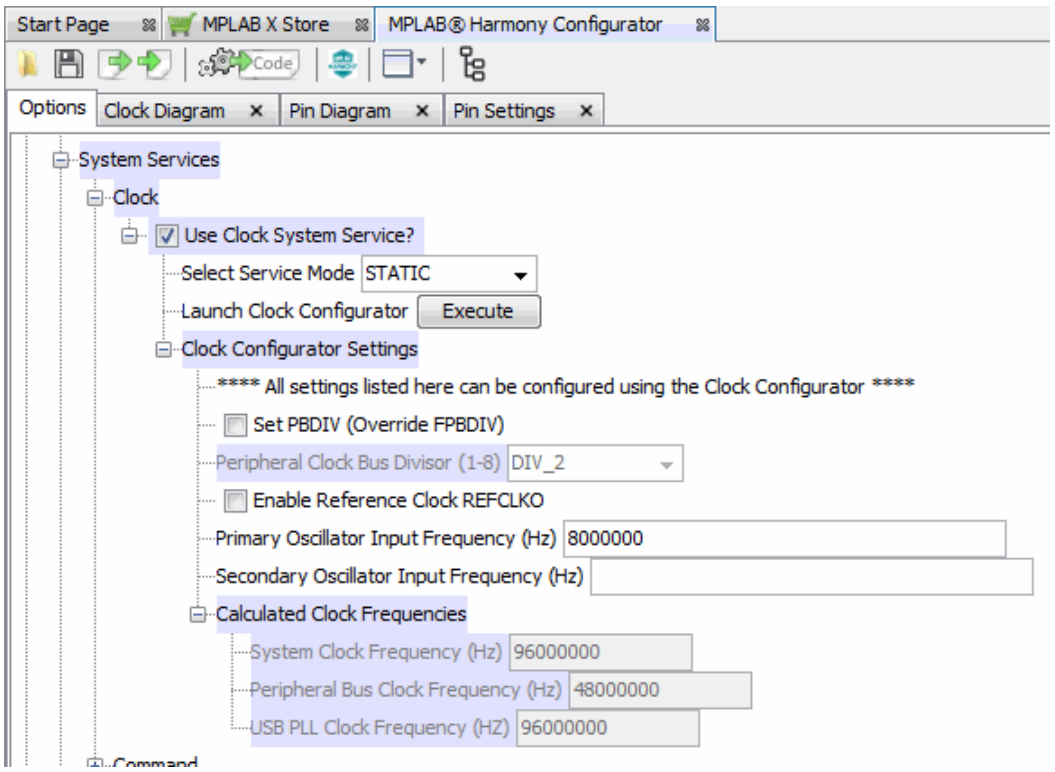


图6： HARMONY框架配置——ADC驱动程序

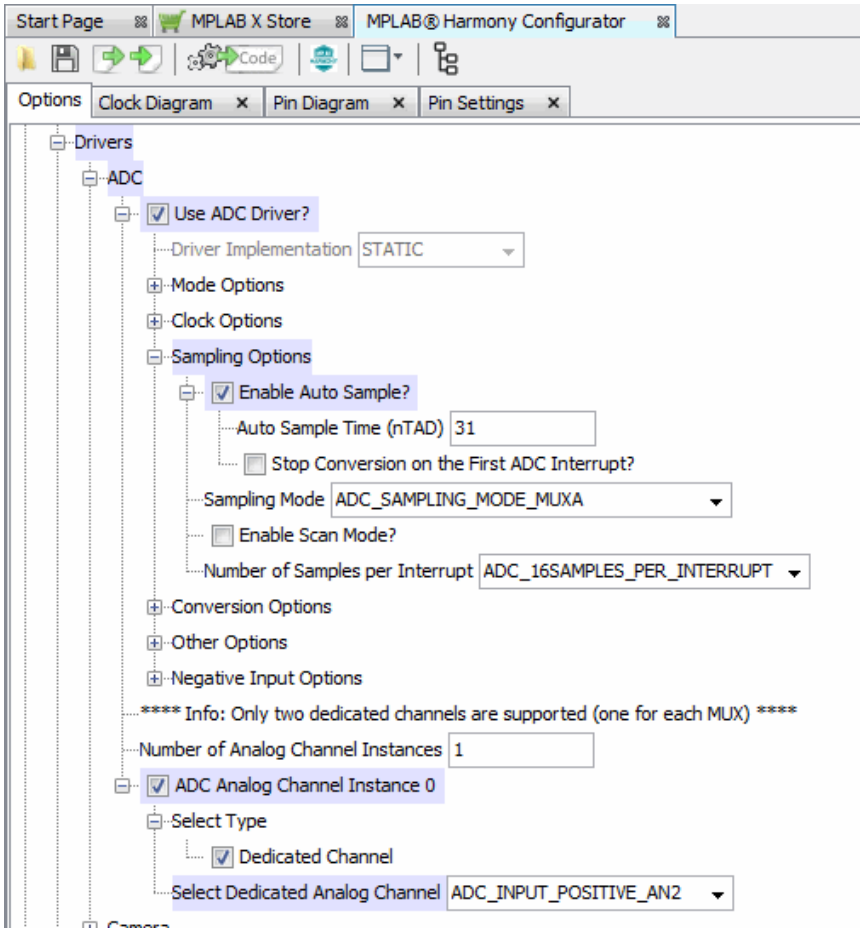


图7： ADC项目资源配置

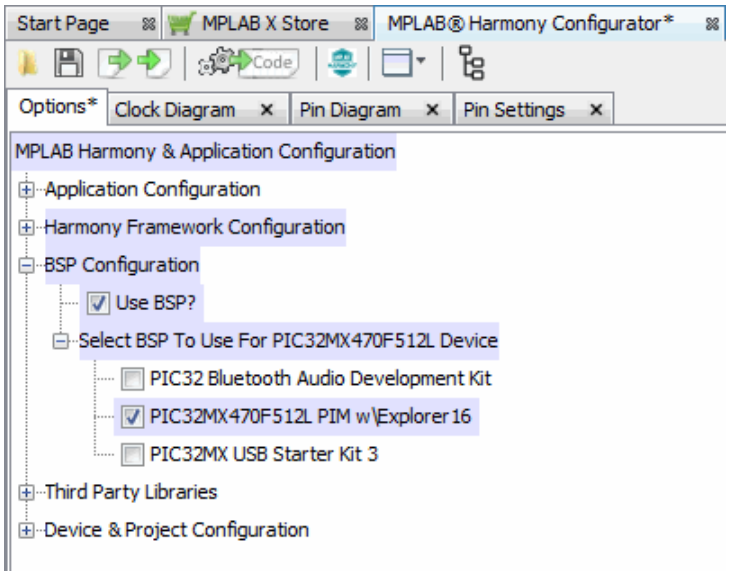
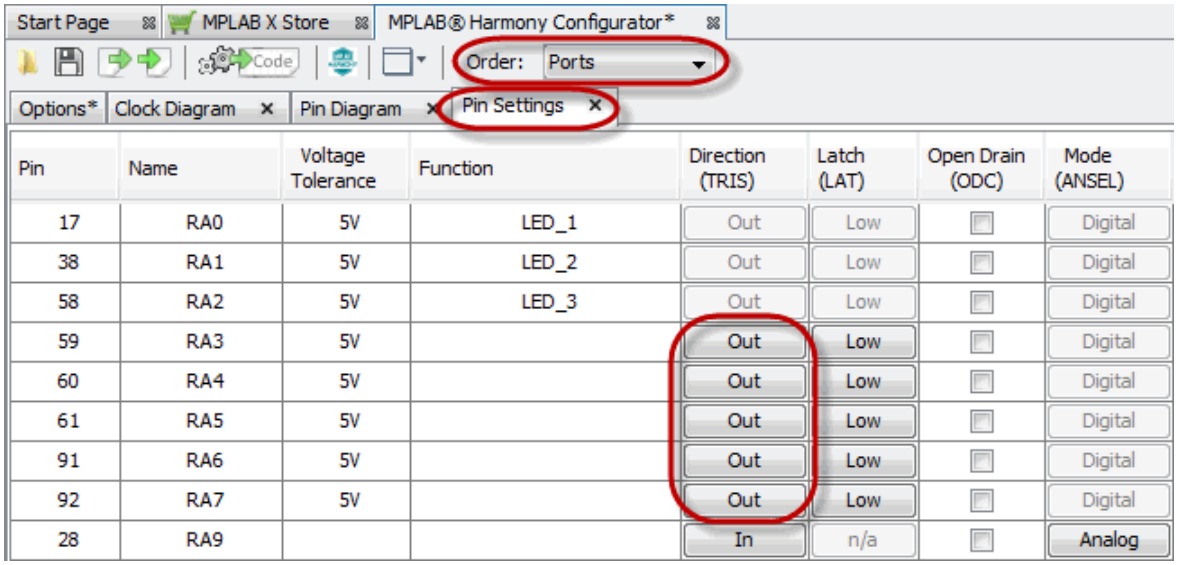


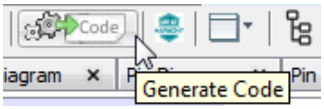
图8： ADC项目引脚设置



Pin	Name	Voltage Tolerance	Function	Direction (TRIS)	Latch (LAT)	Open Drain (ODC)	Mode (ANSEL)
17	RA0	5V	LED_1	Out	Low		Digital
38	RA1	5V	LED_2	Out	Low		Digital
58	RA2	5V	LED_3	Out	Low		Digital
59	RA3	5V		Out	Low		Digital
60	RA4	5V		Out	Low		Digital
61	RA5	5V		Out	Low		Digital
91	RA6	5V		Out	Low		Digital
92	RA7	5V		Out	Low		Digital
28	RA9			In	n/a		Analog

4.3 生成代码和编辑应用程序文件

按照上述各图设置完MHC后，请单击**MPLAB Harmony Configurator**（MPLAB Harmony配置器）选项卡上的**Generate Code**（生成代码）按钮。



保存配置（图9）并生成项目代码（图10）。

图9： 保存配置

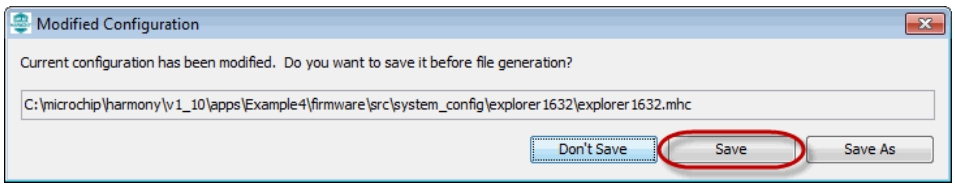
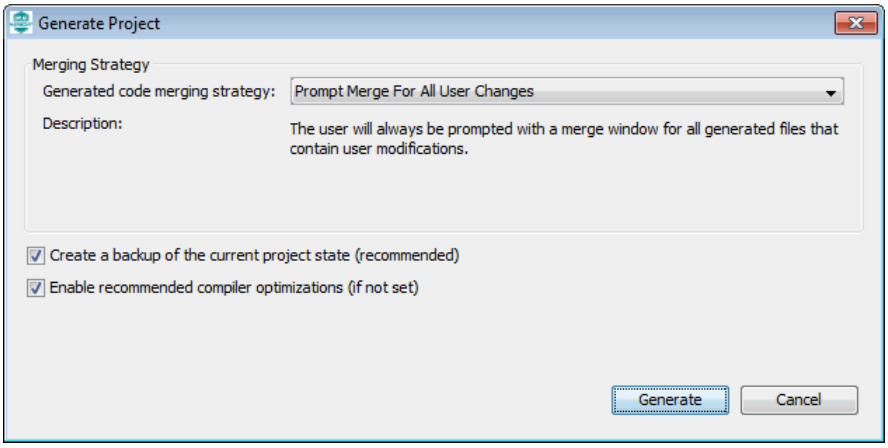


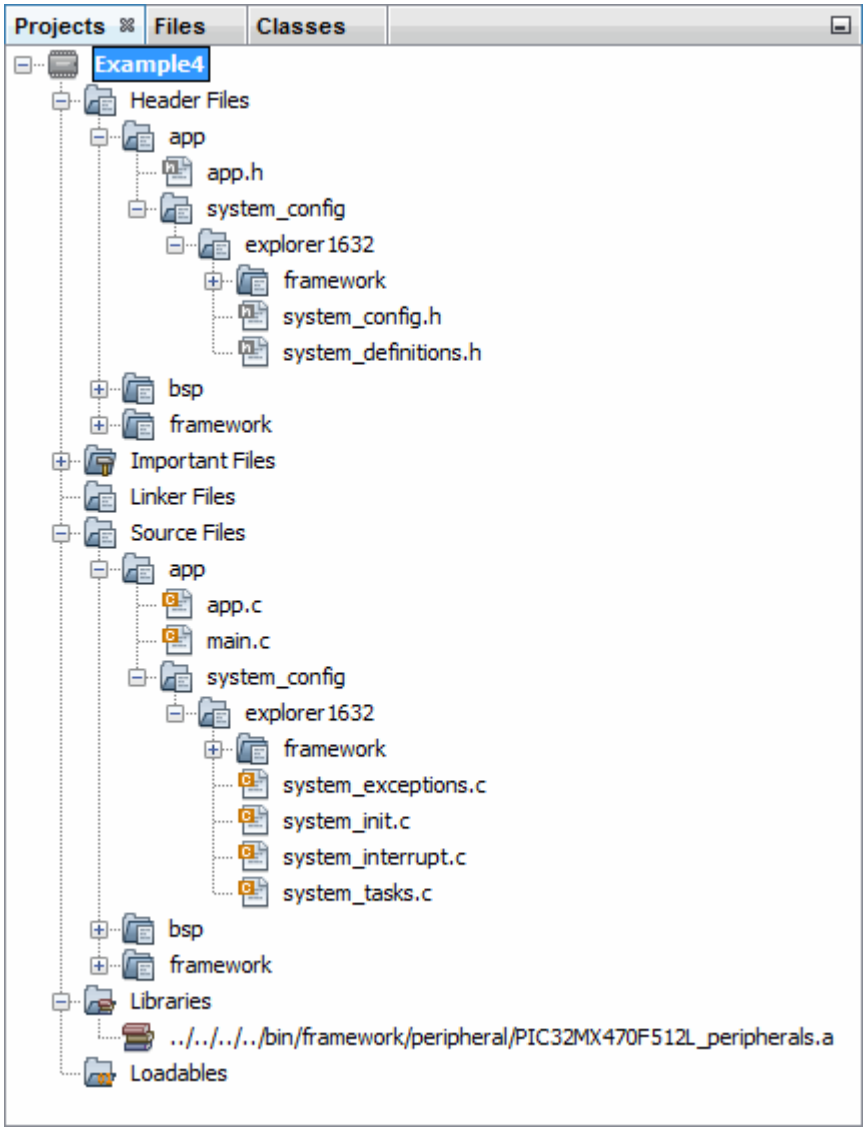
图10： 生成项目代码



MPLAB Harmony生成的代码是模块化的，如图11所示。应用程序文件（app.h和app.c）是为本示例编辑的文件。

有关使用ADC的更多信息，请参见《PIC32系列参考手册》的第17章“10位模数转换器”（DS61104D_CN）。

图11： 通过MHC生成的代码的ADC项目树



4.4 app.h修改后的代码

app.h模板文件已经过编辑，如下所示。部分注释用< >括起，表示已删除。已添加的代码以红色显示。

```
/*******************************************************************
  MPLAB Harmony Application Header File

<See generated app.h file for file information.>

*******************************************************************/

//DOM-IGNORE-BEGIN
/*******************************************************************
Copyright (c) 2013-2014 released Microchip Technology Inc. All rights
reserved.

<See generated app.h file for copyright information.>

*******************************************************************/
//DOM-IGNORE-END

#ifndef _APP_H
#define _APP_H

#define ADC_NUM_SAMPLE_PER_AVERAGE  16

// *****************************************************************
// *****************************************************************
// Section: Included Files
// *****************************************************************
// *****************************************************************

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "system_config.h"
#include "system_definitions.h"

// DOM-IGNORE-BEGIN
#ifdef __cplusplus  // Provide C++ Compatibility

extern "C" {

#endif
// DOM-IGNORE-END

// *****************************************************************
// *****************************************************************
// Section: Type Definitions
// *****************************************************************
// *****************************************************************

// *****************************************************************
/* Application states

  Summary:
    Application states enumeration

  Description:
```

```
    This enumeration defines the valid application states. These
    states determine the behavior of the application at various times.
*/

typedef enum
{
   /* Application's state machine's initial state. */
   APP_STATE_INIT=0,
   APP_ADC_WAIT,
   APP_ADC_DISPLAY

} APP_STATES;

// *****************************************************************
/* Application Data

  Summary:
    Holds application data

  Description:
    This structure holds the application's data.

  Remarks:
    Application strings and buffers are be defined outside
    this structure.
*/

typedef struct
{
    /* The application's current state */
    APP_STATES state;

    /* Values for the conversions */
    int potValue;
    int ledMask;

} APP_DATA;

// *****************************************************************
// *****************************************************************
// Section: Application Callback Routines
// *****************************************************************
// *****************************************************************
/* These routines are called by drivers when certain events occur.
*/

// *****************************************************************
// *****************************************************************
// Section: Application Initialization and State Machine Functions
// *****************************************************************
// *****************************************************************

/*******************************************************************
  Function:
    void APP_Initialize ( void )

  Summary:
     MPLAB Harmony application initialization routine.

<See generated app.h file for app init information.>
```

```
*/

void APP_Initialize ( void );

/*******************************************************************
  Function:
    void APP_Tasks ( void )

  Summary:
    MPLAB Harmony Demo application tasks function

<See generated app.h file for app tasks information.>

*/

void APP_Tasks( void );

#endif /* _APP_H */

//DOM-IGNORE-BEGIN
#ifdef __cplusplus
}
#endif
//DOM-IGNORE-END

/*******************************************************************
 End of File
*/
```

4.5 app.c修改后的代码

app.c模板文件已经过编辑，如下所示。部分注释用< >括起，表示已删除。已添加的代码以红色显示。

有些行很长，将在页面中换行显示。采用这种处理方式可从本文档中剪切相应内容并粘贴到编辑器中。

```
/*******************************************************************
  MPLAB Harmony Application Source File

<See generated app.c file for file information.>

*******************************************************************/

// DOM-IGNORE-BEGIN
/*******************************************************************
Copyright (c) 2013-2014 released Microchip Technology Inc. All rights
reserved.

<See generated app.c file for copyright information.>

*******************************************************************/
// DOM-IGNORE-END

// *****************************************************************
// *****************************************************************
// Section: Included Files
// *****************************************************************
// *****************************************************************

#include "app.h"

// *****************************************************************
// *****************************************************************
// Section: Global Data Definitions
// *****************************************************************
// *****************************************************************

// *****************************************************************
/* Application Data

  Summary:
    Holds application data

  Description:
    This structure holds the application's data.

  Remarks:
    This structure should be initialized by the APP_Initialize
    function.

    Application strings and buffers are be defined outside this
    structure.
*/

APP_DATA appData;

// *****************************************************************
// *****************************************************************
// Section: Application Callback Functions
```

```
// *****************************************************************************
// *****************************************************************************

/* TODO:  Add any necessary callback functions.
*/

// *****************************************************************************
// *****************************************************************************
// Section: Application Local Functions
// *****************************************************************************
// *****************************************************************************

/*******************************************************************************
  Function:
    void Set_LED_Status ( void )

   Description:
        Set LEDs to display the ADC average result.
*/

void Set_LED_Status(void)
{
    int i;

    appData.ledMask = 0;

    /* Creates a mask for the LEDs, corresponding to the value read
     * from the potentiometer */
    appData.potValue >>= 7; /* 10-bit value to 3-bit value */
    for (i = 0; i <= appData.potValue; i++)
    {
        appData.ledMask |=  1<<(i);
    }
   /* Write the mask to the LEDs */
    SYS_PORTS_Write( PORTS_ID_0, PORT_CHANNEL_A,
        (PORTS_DATA_MASK)appData.ledMask );
}

// *****************************************************************************
// *****************************************************************************
// Section: Application Initialization and State Machine Functions
// *****************************************************************************
// *****************************************************************************

/*******************************************************************************
  Function:
    void APP_Initialize ( void )

  Remarks:
    See prototype in app.h.
 */

void APP_Initialize ( void )
{
    /* Place the App state machine in its initial state. */
    appData.state = APP_STATE_INIT;


    /* TODO: Initialize your application's state machine and other
     * parameters.
```

```
     */
}

/*******************************************************************************
  Function:
    void APP_Tasks ( void )

  Remarks:
    See prototype in app.h.
*/

void APP_Tasks ( void )
{

    /* Check the application's current state. */
    switch ( appData.state )
    {
        /* Application's initial state. */
        case APP_STATE_INIT: ◄——— 请参见第4.6节
        {
            /* Enable ADC */
            DRV_ADC_Open();

            appData.state = APP_ADC_WAIT;

            break;
        }

        /* Display pot value on LEDs*/
        case APP_ADC_DISPLAY: ◄——— 请参见第4.7节
        {
            Set_LED_Status();
            appData.state = APP_ADC_WAIT;

            break;
        }

        /* Wait for ADC */
        case APP_ADC_WAIT: ◄——— 请参见第4.8节
        {
            /* Wait for conversion*/
            if (DRV_ADC_SamplesAvailable())
            {
                int i;

                //Read data
                for(i=0;i<ADC_NUM_SAMPLE_PER_AVERAGE;i++)
                    appData.potValue +=
                        PLIB_ADC_ResultGetByIndex(ADC_ID_1, i);
                appData.potValue = appData.potValue /
                    ADC_NUM_SAMPLE_PER_AVERAGE;

                appData.state = APP_ADC_DISPLAY;
            }

            break;
        }

        /* The default state should never be executed. */
        default:
```

```
            {
                /* TODO: Handle error in application's state machine. */
                break;
            }
        }
    }

/*******************************************************************
 End of File
 */
```

4.6 应用程序状态——APP_STATE_INIT

当任务循环开始时，应用程序处于初始状态。这种情况下，ADC将在自动采样模式下使能。随后，应用程序状态更改为等待（APP_ADC_WAIT）。应用程序状态在app.h中定义。

4.7 应用程序状态——APP_ADC_DISPLAY

在APP_ADC_WAIT case分支中捕捉到ADC值后，将通过调用“局部函数”部分中的函数Set_LED_Status()显示该值。该函数使用掩码（appData.ledMask）将来自电位器的ADC值（appData.potValue）显示到LED上。这些变量在app.h中定义。

函数返回后，应用程序状态即变回APP_ADC_WAIT，以等待下一次采样。

4.8 应用程序状态——APP_ADC_WAIT

初始化（APP_STATE_INIT）后，应用程序将等待对电位器值进行转换。之后，将ADC值赋给变量appData.potValue，以便在APP_ADC_DISPLAY case分支中显示在LED上。ADC_NUM_SAMPLE_PER_AVERAGE在app.h中定义。

5. 使用ADC在LED上显示电位器值（MCC）

本示例与示例3使用相同的器件以及端口A LED。但在本示例中，将使用演示板上电位器的值通过端口B（RB2/AN2）提供模数转换器（ADC）输入，然后进行转换并显示在LED上。

代码无需手动编写，而是使用MPLAB代码配置器（MPLAB Code Configurator，MCC）生成。MCC是一款插件，可在MPLAB X IDE菜单*Tools>Plugins*的**Available Plugins**选项卡下安装。有关如何安装插件的更多信息，请参见MPLAB X IDE帮助。

有关MCC的安装信息以及《MPLAB®代码配置器用户指南》（DS40001725B_CN），请访问MPLAB代码配置器网页：

http://www.microchip.com/mplab/mplab-code-configurator

本示例中的MCC设置如以下各图所示。

图12： ADC项目资源——系统模块

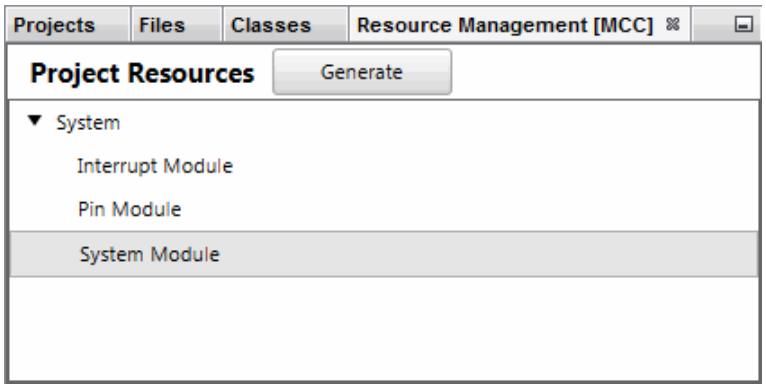


图13： ADC项目系统模块简单设置

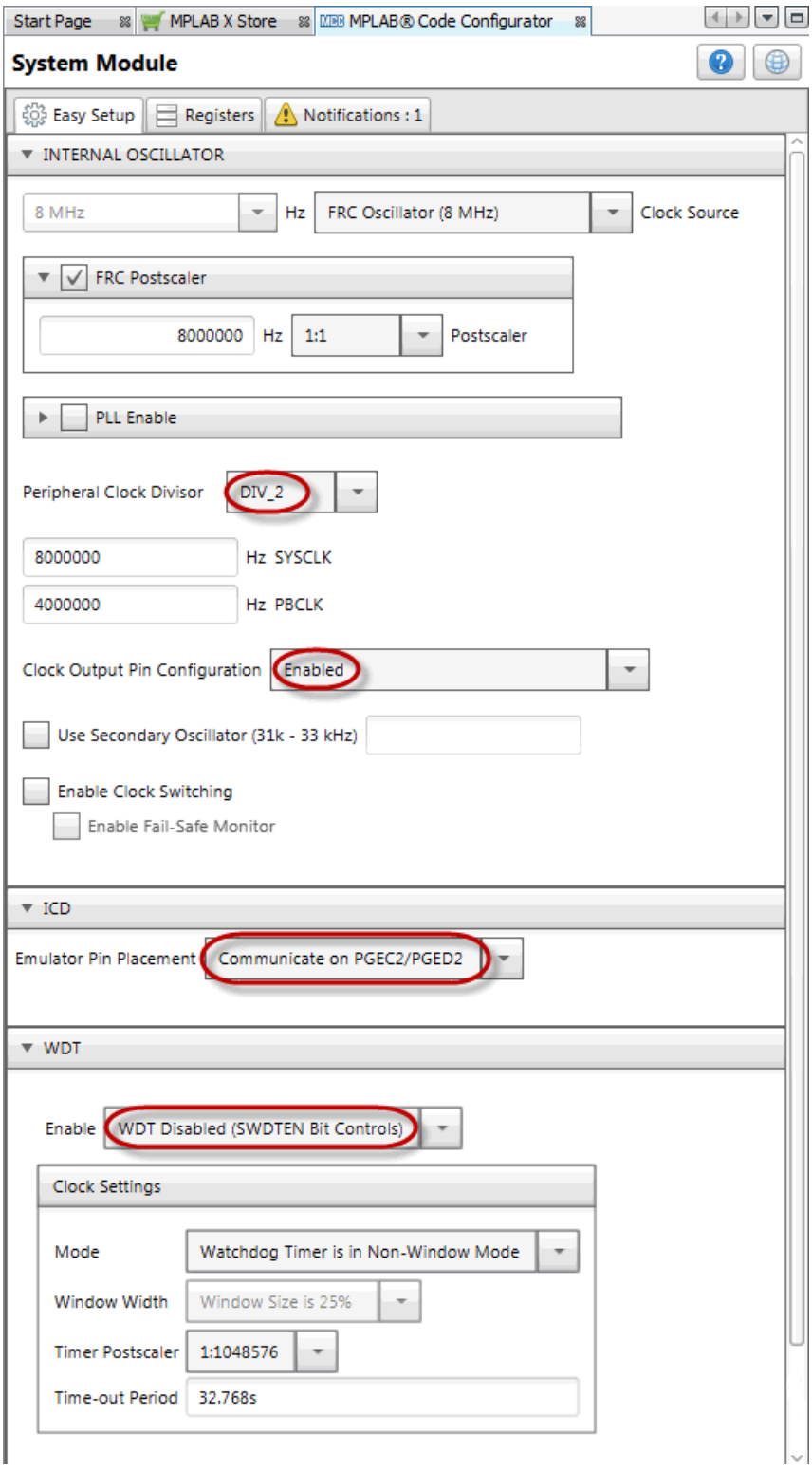


图14： ADC项目系统模块寄存器

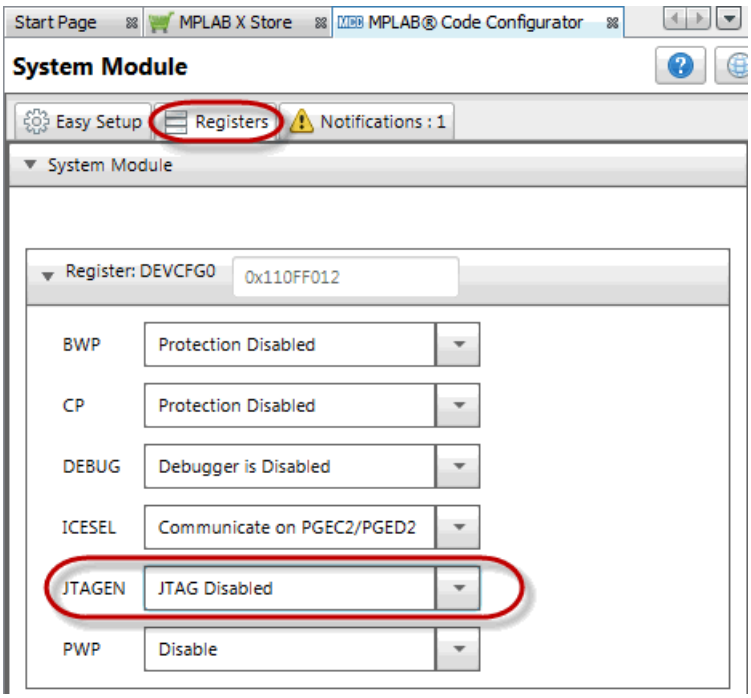


图15： ADC项目资源——ADC1

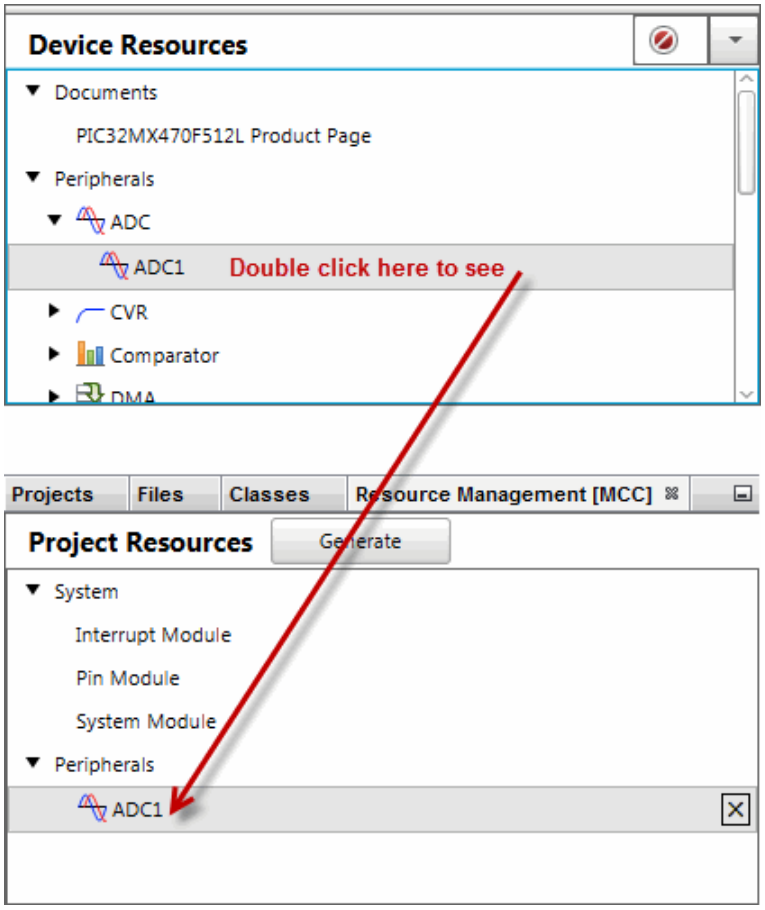
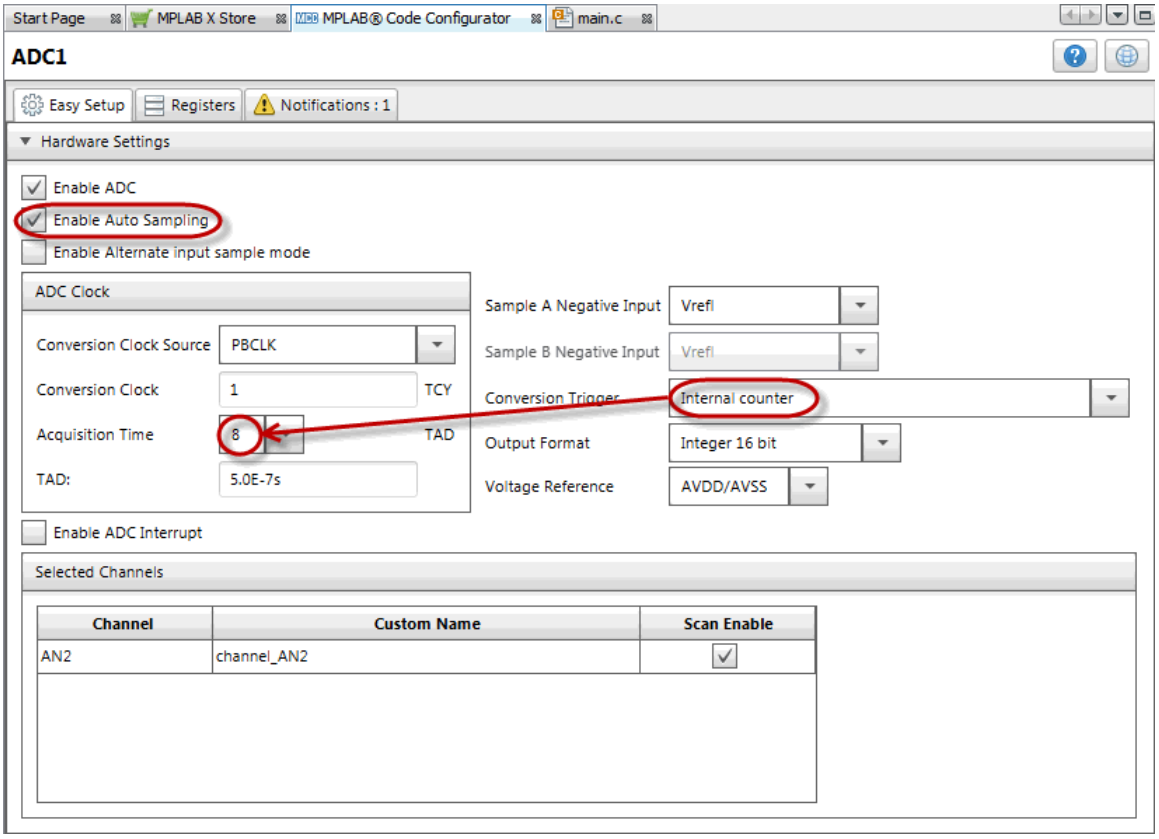


图16： ADC项目ADC1简单设置



RB2至AN2映射将在图17中的选择完成后显示。

图17： ADC项目ADC1引脚资源

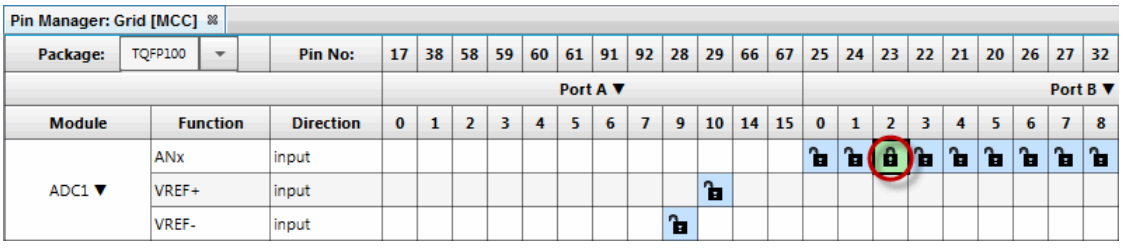


图18： ADC项目资源——引脚模块

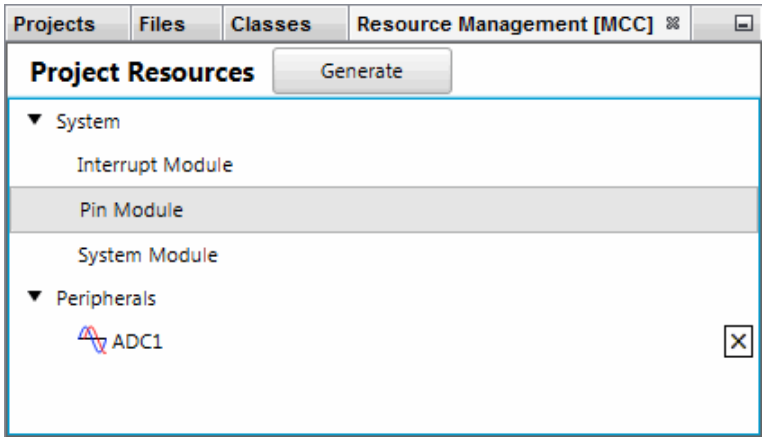
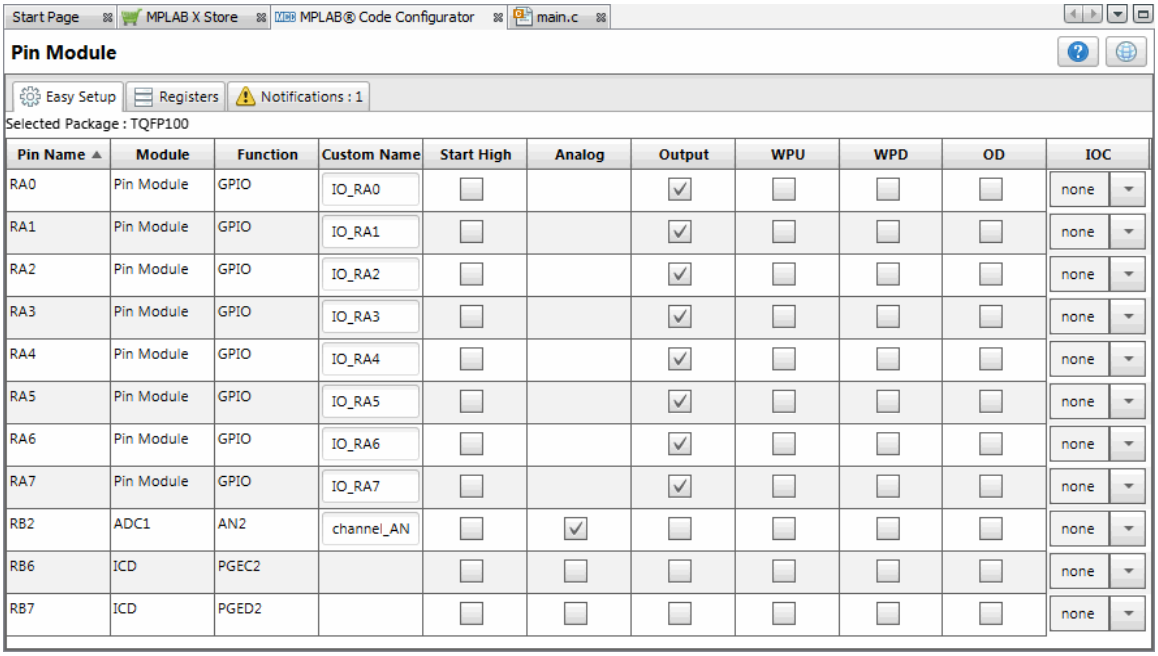


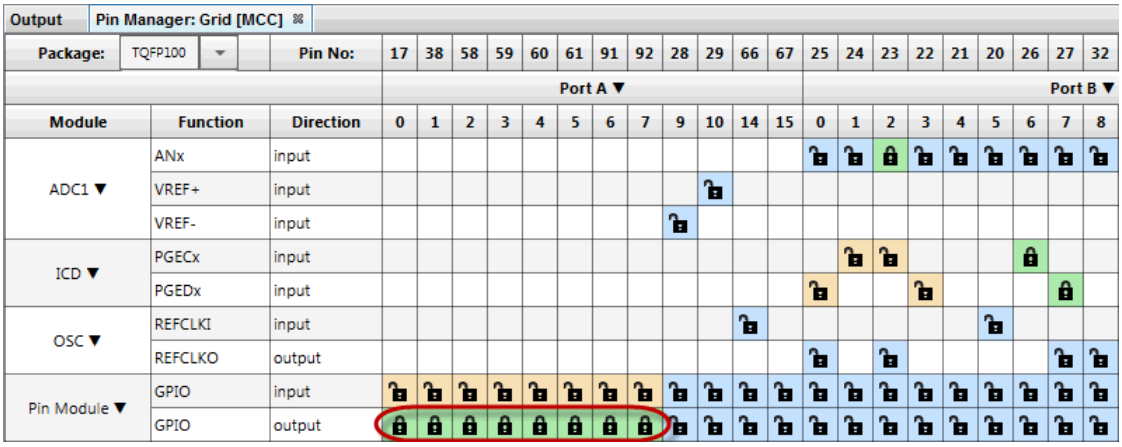
图19： ADC项目引脚模块简单设置



Pin Name	Module	Function	Custom Name	Start High	Analog	Output	WPU	WPD	OD	IOC
RA0	Pin Module	GPIO	IO_RA0	☐		☑	☐	☐	☐	none
RA1	Pin Module	GPIO	IO_RA1	☐		☑	☐	☐	☐	none
RA2	Pin Module	GPIO	IO_RA2	☐		☑	☐	☐	☐	none
RA3	Pin Module	GPIO	IO_RA3	☐		☑	☐	☐	☐	none
RA4	Pin Module	GPIO	IO_RA4	☐		☑	☐	☐	☐	none
RA5	Pin Module	GPIO	IO_RA5	☐		☑	☐	☐	☐	none
RA6	Pin Module	GPIO	IO_RA6	☐		☑	☐	☐	☐	none
RA7	Pin Module	GPIO	IO_RA7	☐		☑	☐	☐	☐	none
RB2	ADC1	AN2	channel_AN	☐	☑	☐	☐	☐	☐	none
RB6	ICD	PGEC2		☐	☐	☐	☐	☐	☐	none
RB7	ICD	PGED2		☐	☐	☐	☐	☐	☐	none

在图20中选择引脚RA0:7后，这些引脚将出现在上面的窗口中。
图17中之前选择了RB2。
图13中选择了PGEC2/PGED2（即RB6和RB7）。
在窗口中显示后，可为每个引脚查看或选择引脚配置。

图20： ADC项目引脚资源



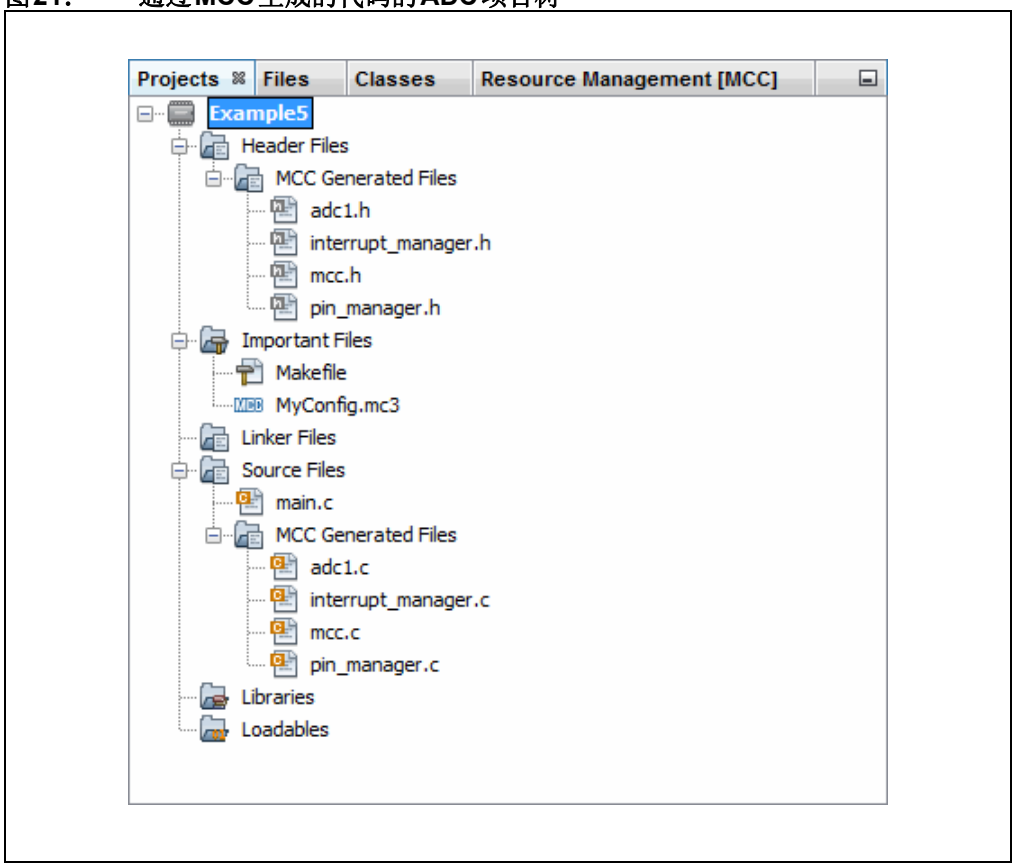
按照上述各图配置完代码后，请单击“Project Resources”（项目资源）窗口上的**Generate**（生成）按钮（图12）。通过MCC生成的代码是模块化的。因此，主函数代码、系统代码和外设代码均位于单独的文件中。此外，每个外设都有自己的头文件。

可通过生成中断管理器文件来捕捉潜在的错误。尽管本应用程序中不会使用中断，但会生成中断管理器文件供将来使用。

向程序中添加功能时始终需要编辑`main.c`。请查看生成的文件以找到您的代码中可能需要的任何函数或宏。

有关使用ADC的更多信息，请参见《PIC32系列参考手册》的第17章“10位模数转换器”（DS61104D_CN）。

图21：　通过MCC生成的代码的ADC项目树



5.1 main.c修改后的代码

main.c模板文件已经过编辑，如下所示。部分注释用< >括起，表示已删除。已添加到main()的代码以红色显示。

```
/**
  Generated Main Source File

<See generated main.c file for file information.>
 */

/*
(c) 2016 Microchip Technology Inc. and its subsidiaries. You may use
this software and any derivatives exclusively with Microchip products.

<See generated main.c file for additional copyright information.>
 */

#include "mcc_generated_files/mcc.h"

unsigned int value = 0;

/* Creates a mask for the LEDs, corresponding
 * to the value read from the potentiometer */
unsigned int Mask_Value(unsigned int pot_value){
    int i;
    unsigned int mask_value = 0;

    pot_value >>= 7; /* 10-bit value to 3-bit value */
    for (i = 0; i <= pot_value; i++)
    {
        mask_value |=  1<<(i);
    }

    return mask_value;
}

/*
                         Main application
 */
int main(void) {
    // initialize the device
    SYSTEM_Initialize();

    while (1) {

        // Wait for conversion
        // and then get result
        while(!ADC1_IsConversionComplete());
        value = ADC1_ConversionResultGet();

        // Mark value
        value = Mask_Value(value);

        // Write to Port Latch/LEDs
        LATA = value;

    }
    return -1;
}
```

请参见第5.2节

请参见第5.3节

请参见第5.4节

```
/**
 End of File
 */
```

5.2 ADC转换及结果

MCC通过设置AD1CON1中的各位来开启ADC、使用自动采样采集以及使用内部计数器结束采样和启动转换。因此，`main()`代码只需要等待转换结束并获取结果。

使用`adc1.c`模块中的如下函数：

```
bool ADC1_IsConversionComplete(void)
uint16_t ADC1_ConversionResultGet(void)
```

有关设置其他ADC功能的信息，请参见《PIC32系列参考手册》的第17章“10位模数转换器（ADC）”（DS61104D_CN）。

5.3 ADC转换结果掩码

由于只有8个LED可用，且ADC转换结果为10位，因此变量`value`中的转换结果通过函数`Mask_Value()`掩码，以便将值以3位一组的形式显示在LED上。

5.4 写入端口锁存器和LED

ADC转换掩码结果将显示在端口A的LED上。

6. 在LED上显示闪存值（MPLAB HARMONY）

本示例与示例4使用相同的器件以及端口A LED。但在本示例中，将通过（非易失性）闪存读写值，并会显示这些操作成功（LED2）还是失败（LED0）。

代码无需手动编写，而是使用MPLAB Harmony生成。有关如何下载和安装MPLAB Harmony集成软件框架和MPLAB Harmony配置器（MHC）MPLAB X IDE插件的信息，请参见**第4节“使用ADC在LED上显示电位器值（MPLAB Harmony）”**。

有关在MPLAB X IDE中创建MPLAB Harmony项目的信息，请参见**第4.1节“在MPLAB X IDE中创建一个MPLAB Harmony项目”**。本示例中将项目命名为“Example6”。

本示例基于以下路径下的闪存驱动程序（以Windows操作系统为例）：

```
C:\microchip\harmony\v1_10\apps\examples\peripheral\flash\flash_modify
```

6.1 配置MPLAB Harmony项目

根据您的项目设置，MPLAB Harmony配置器（MHC）打开时将提供部分时钟信息。以蓝色突出显示的文本表示更改。在本例中，不要对时钟设置进行任何更改。

配置闪存驱动程序（如图22所示）和板级支持包（BSP）（如图23所示）。

图22： HARMONY框架配置——闪存驱动程序

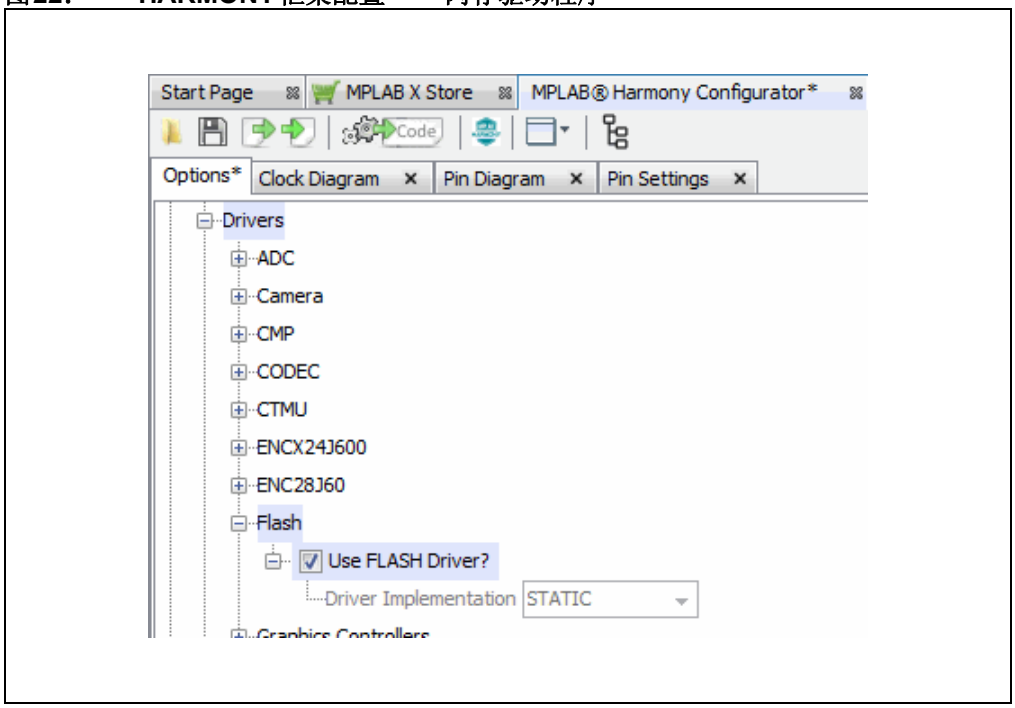
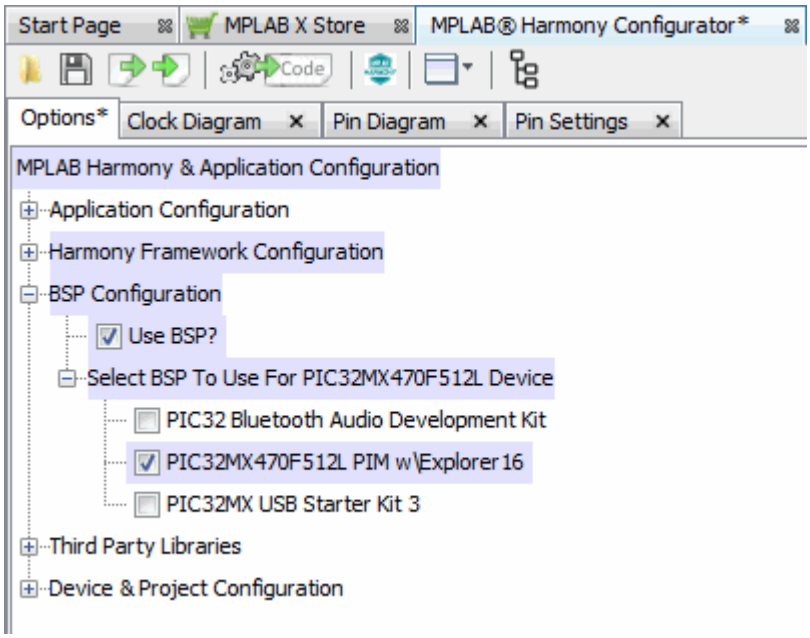
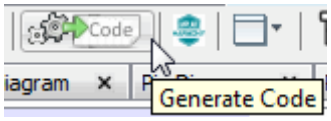


图23：　　闪存项目资源配置



6.2 生成代码和编辑应用程序文件

按照上述各图设置完MHC后，请单击**MPLAB Harmony Configurator**选项卡上的**Generate Code**按钮。

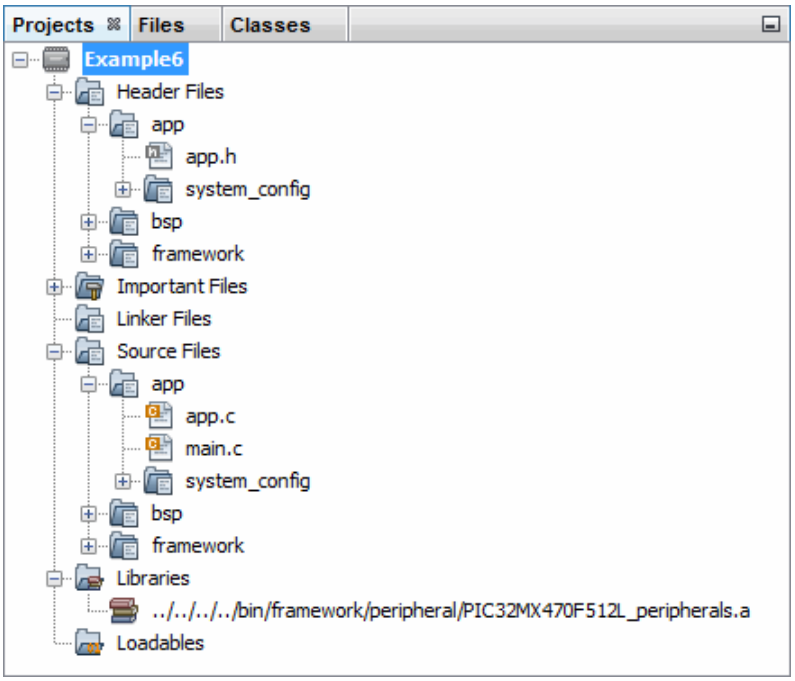


保存配置并按照**第4.3节“生成代码和编辑应用程序文件”**生成项目代码。

MPLAB Harmony生成的代码是模块化的，如图24所示。应用程序文件（app.h和app.c）是为本示例编辑的文件。

有关使用闪存的更多信息，请参见《PIC32系列参考手册》的第5章“闪存编程”（DS60001121G_CN）。

图24： 通过MHC生成的代码的闪存项目树



6.3 app.h修改后的代码

app.h模板文件已经过编辑，如下所示。部分注释用< >括起，表示已删除。已添加的代码以红色显示。

```
/*******************************************************************
  MPLAB Harmony Application Header File

<See generated app.h file for file information.>

*******************************************************************/

//DOM-IGNORE-BEGIN
/*******************************************************************
Copyright (c) 2013-2014 released Microchip Technology Inc. All rights
reserved.

<See generated app.h file for copyright information.>

 *******************************************************************/
//DOM-IGNORE-END

#ifndef _APP_H
#define _APP_H

#define USERLED_SUCCESS             LED_2 //D5 on Explorer 16/32
#define USERLED_ERROR               LED_0 //D3 on Explorer 16/32

// *****************************************************************
// *****************************************************************
// Section: Included Files
// *****************************************************************
// *****************************************************************

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "system_config.h"
#include "system_definitions.h"

// DOM-IGNORE-BEGIN
#ifdef __cplusplus  // Provide C++ Compatibility

extern "C" {

#endif
// DOM-IGNORE-END

// *****************************************************************
// *****************************************************************
// Section: Type Definitions
// *****************************************************************
// *****************************************************************

#define APP_DATABUFF_SIZE         (sizeof(databuff) /
sizeof(uint32_t))

/* Row size for device is 2Kbytes */
#define APP_DEVICE_ROW_SIZE_DIVIDED_BY_4
(DRV_FLASH_ROW_SIZE/4)
```

```
/* Page size for device is 16Kbytes */
#define APP_DEVICE_PAGE_SIZE_DIVIDED_BY_4
(DRV_FLASH_PAGE_SIZE/4)

#define APP_PROGRAM_FLASH_BASE_ADDRESS_VALUE  (unsigned int)
0x9D008000
#define APP_PROGRAM_FLASH_BASE_ADDRESS  (unsigned int *)
APP_PROGRAM_FLASH_BASE_ADDRESS_VALUE

// *****************************************************************
/* Application states

  Summary:
    Application states enumeration

  Description:
    This enumeration defines the valid application states.  These
states
    determine the behavior of the application at various times.
*/

typedef enum
{
    /* Application's state machine's initial state. */
    APP_STATE_INIT=0,
    APP_STATE_NVM_FILL_DATABUF_AND_ERASE_STATE,
    APP_STATE_NVM_ERASE_COMPLETION_CHECK,
    APP_STATE_NVM_WRITE_START,
    APP_STATE_NVM_WRITE_COMPLETION_CHECK_AND_VERIFY_CHECK,
    APP_STATE_NVM_ERROR_STATE,
    APP_STATE_NVM_SUCCESS_STATE,

} APP_STATES;


// *****************************************************************
/* Application Data

  Summary:
    Holds application data

  Description:
    This structure holds the application's data.

  Remarks:
    Application strings and buffers are be defined outside this
structure.
 */

typedef struct
{
    /* The application's current state */
    APP_STATES state;
    DRV_HANDLE flashHandle;

} APP_DATA;


// *****************************************************************
// *****************************************************************
```

```
// Section: Application Callback Routines
// *****************************************************************
// *****************************************************************
/* These routines are called by drivers when certain events occur.
*/

// *****************************************************************
// *****************************************************************
// Section: Application Initialization and State Machine Functions
// *****************************************************************
// *****************************************************************

/*******************************************************************
  Function:
    void APP_Initialize ( void )

  Summary:
     MPLAB Harmony application initialization routine.

<See generated app.h file for app init information.>

*/

void APP_Initialize ( void );


/*******************************************************************
  Function:
    void APP_Tasks ( void )

  Summary:
    MPLAB Harmony Demo application tasks function

<See generated app.h file for app tasks information.>

 */

void APP_Tasks( void );


#endif /* _APP_H */

//DOM-IGNORE-BEGIN
#ifdef __cplusplus
}
#endif
//DOM-IGNORE-END

/*******************************************************************
 End of File
 */
```

6.4 app.c修改后的代码

app.c模板文件已经过编辑，如下所示。部分注释用< >括起，表示已删除。已添加的代码以红色显示。

有些行很长，将在页面中换行显示。采用这种处理方式可从本文档中剪切相应内容并粘贴到编辑器中。

```
/*******************************************************************
  MPLAB Harmony Application Source File

<See generated app.c file for file information.>

 *******************************************************************/

// DOM-IGNORE-BEGIN
/*******************************************************************
Copyright (c) 2013-2014 released Microchip Technology Inc. All rights
reserved.

<See generated app.c file for copyright information.>

 *******************************************************************/
// DOM-IGNORE-END


// *****************************************************************
// *****************************************************************
// Section: Included Files
// *****************************************************************
// *****************************************************************

#include "app.h"

// *****************************************************************
// *****************************************************************
// Section: Global Data Definitions
// *****************************************************************
// *****************************************************************

/***************************************************
 * Initialize the application data structure. All
 * application related variables are stored in this
 * data structure.
 ***************************************************/

/* Array in the KSEG1 RAM to store the data */
uint32_t databuff[APP_DEVICE_ROW_SIZE_DIVIDED_BY_4]
__attribute__((coherent, aligned(16)));

// *****************************************************************
/* Application Data

  Summary:
    Holds application data

  Description:
    This structure holds the application's data.

  Remarks:
```

```
    This structure should be initialized by the APP_Initialize
function.

    Application strings and buffers are be defined outside this
structure.
*/

APP_DATA appData;

// *****************************************************************
// *****************************************************************
// Section: Application Callback Functions
// *****************************************************************
// *****************************************************************

/* TODO:  Add any necessary callback functions.
*/

// *****************************************************************
// *****************************************************************
// Section: Application Local Functions
// *****************************************************************
// *****************************************************************


/* TODO:  Add any necessary local functions.
*/


// *****************************************************************
// *****************************************************************
// Section: Application Initialization and State Machine Functions
// *****************************************************************
// *****************************************************************

/*******************************************************************
  Function:
    void APP_Initialize ( void )

  Remarks:
    See prototype in app.h.
 */

void APP_Initialize ( void )
{
    /* Place the App state machine in its initial state. */
    appData.state = APP_STATE_INIT;


    /* TODO: Initialize your application's state machine and other
     * parameters.
     */
}


/*******************************************************************
  Function:
    void APP_Tasks ( void )

  Remarks:
```

```
    See prototype in app.h.
 */

void APP_Tasks ( void )
{
    unsigned int x;
    /* Check the application's current state. */
    switch ( appData.state )
    {
        /* Application's initial state. */  <-- 请参见第6.5节
        case APP_STATE_INIT:
          appData.flashHandle = DRV_FLASH_Open(DRV_FLASH_INDEX_0,
              intent);
          appData.state = APP_STATE_NVM_FILL_DATABUF_AND_ERASE_STATE;
          break;

        /* Fill data buffer, clear LEDs,  <-- 请参见第6.6节
         * and begin erase page */
        case APP_STATE_NVM_FILL_DATABUF_AND_ERASE_STATE:
          for (x = 0; x < APP_DATABUFF_SIZE; x++)
          {
              databuff[x] = x;
          }
          BSP_LEDOff(USERLED_SUCCESS);
          BSP_LEDOff(USERLED_ERROR);

          /* Erase the page which consist of the row to be written */
          DRV_FLASH_ErasePage(appData.flashHandle,
              APP_PROGRAM_FLASH_BASE_ADDRESS_VALUE);
          appData.state = APP_STATE_NVM_ERASE_COMPLETION_CHECK;
          break;

        /* Check for erase complete */  <-- 请参见第6.7节
        case APP_STATE_NVM_ERASE_COMPLETION_CHECK:
          if(!DRV_FLASH_IsBusy(appData.flashHandle))
          {
              appData.state = APP_STATE_NVM_WRITE_START;
          }
          break;

        /* Write row of Flash */  <-- 请参见第6.8节
        case APP_STATE_NVM_WRITE_START:
          /* Erase Success */
          /* Write a row of data to PROGRAM_FLASH_BASE_ADDRESS,
           * using databuff array as the source */
          DRV_FLASH_WriteRow(appData.flashHandle,
              APP_PROGRAM_FLASH_BASE_ADDRESS_VALUE, databuff);
          appData.state =
              APP_STATE_NVM_WRITE_COMPLETION_CHECK_AND_VERIFY_CHECK;
          break;

        /* Check for write complete  <-- 请参见第6.9节
         * and verify write operation */
        case APP_STATE_NVM_WRITE_COMPLETION_CHECK_AND_VERIFY_CHECK:
          if(!DRV_FLASH_IsBusy(appData.flashHandle))
          {
              /* Verify that data written to flash memory is valid
               * (databuff array read from kseg1) */
              if (!memcmp(databuff,
                (void *)KVA0_TO_KVA1(APP_PROGRAM_FLASH_BASE_ADDRESS),
```

```
                    sizeof(databuff)))
                {
                    appData.state = APP_STATE_NVM_SUCCESS_STATE;
                }
                else
                {
                    appData.state = APP_STATE_NVM_ERROR_STATE;
                }
            }
            break;

        /* Write Failure */  ◄——— 请参见第6.10节
        case APP_STATE_NVM_ERROR_STATE:
            /*stay here, nvm had a failure*/
            BSP_LEDOn(USERLED_ERROR);
            BSP_LEDOff(USERLED_SUCCESS);
            break;

        /* Write Success */  ◄——— 请参见第6.11节
        case APP_STATE_NVM_SUCCESS_STATE:
            BSP_LEDOn(USERLED_SUCCESS);
            BSP_LEDOff(USERLED_ERROR);
            break;

    }
}


/*******************************************************************************
 End of File
 */
```

6.5 应用程序状态——初始状态

当任务循环开始时，应用程序处于初始状态（APP_STATE_INIT）。在这种情况下，闪存驱动程序将进行初始化，状态将变为下一个状态（APP_STATE_NVM_FILL_DATABUF_AND_ERASE_STATE）。应用程序状态在app.h中定义。

6.6 应用程序状态——填充数据缓冲区和擦除页

初始化完成后，即执行写入准备操作。首先，向数据缓冲区填充将写入闪存的值（数据缓冲区在“全局数据定义”部分定义）。接着，清除指定成功和失败的LED的状态（这些值在app.h中设置）。然后，开始擦除闪存（NVM）页。最后，应用程序状态将更改为等待页擦除完成（APP_STATE_NVM_ERASE_COMPLETION_CHECK）。

6.7 应用程序状态——页擦除完成

该状态等待在前一个状态下开始的页擦除完成。完成后，应用程序状态即更改为开始写入闪存（NVM）（APP_STATE_NVM_WRITE_START）。

6.8 应用程序状态——写入闪存行

现在开始写入闪存中已擦除页的一行。数据缓冲区中的值将写入该行。随后，应用程序状态将更改为等待写入完成并验证结果（APP_STATE_NVM_WRITE_COMPLETION_CHECK_AND_VERIFY_CHECK）。

6.9 应用程序状态——行写入完成并验证

该状态等待在前一个状态下开始的行写入完成。完成后，即根据数据缓冲区中的值验证写入操作。如果写入成功，则应用程序状态将更改为APP_STATE_NVM_SUCCESS_STATE。如果写入失败，则应用程序状态将更改为APP_STATE_NVM_ERROR_STATE。

6.10 应用程序状态——错误状态

如果闪存写入失败，则进入错误状态。演示板上的LED 3（D3）将点亮，表示发生了错误。

6.11 应用程序状态——成功状态

闪存成功擦除并写入后，即进入成功状态。演示板上的LED 5（D5）将点亮，表示应用程序成功执行。

7. 在LED上显示闪存值（MCC）

本示例与示例5使用相同的器件以及端口A LED。但在本示例中，将通过（非易失性）闪存读写值，并会显示这些操作成功（LED2）还是失败（LED0）。

使用MPLAB代码配置器（MCC）来生成部分代码。欲了解如何安装MCC和获取MCC用户指南的信息，请参见：**第5节“使用ADC在LED上显示电位器值（MCC）”**。

本示例中的MCC设置如以下各图所示。

图25： 闪存项目资源——系统模块

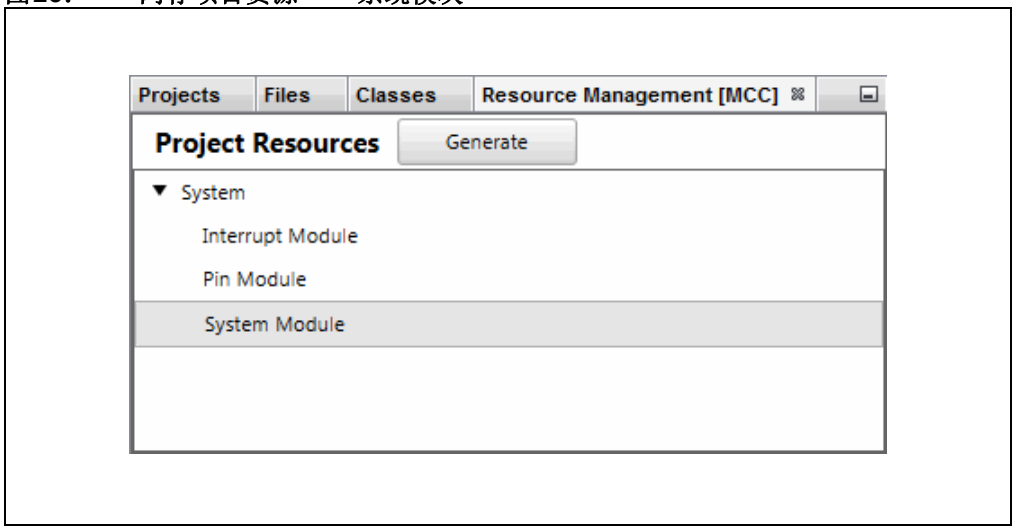


图26：　闪存项目系统模块简单设置

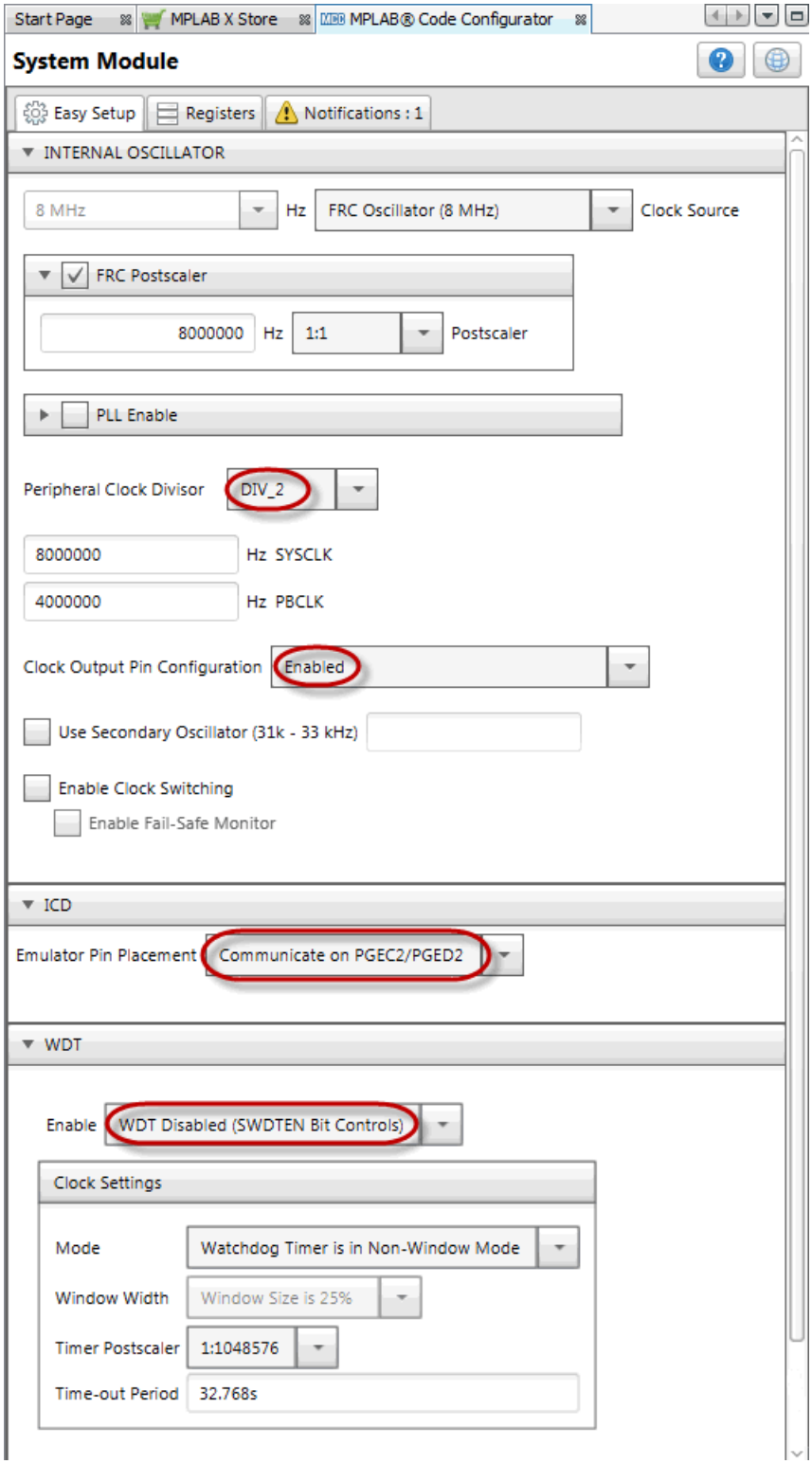


图27：　　闪存项目系统模块寄存器

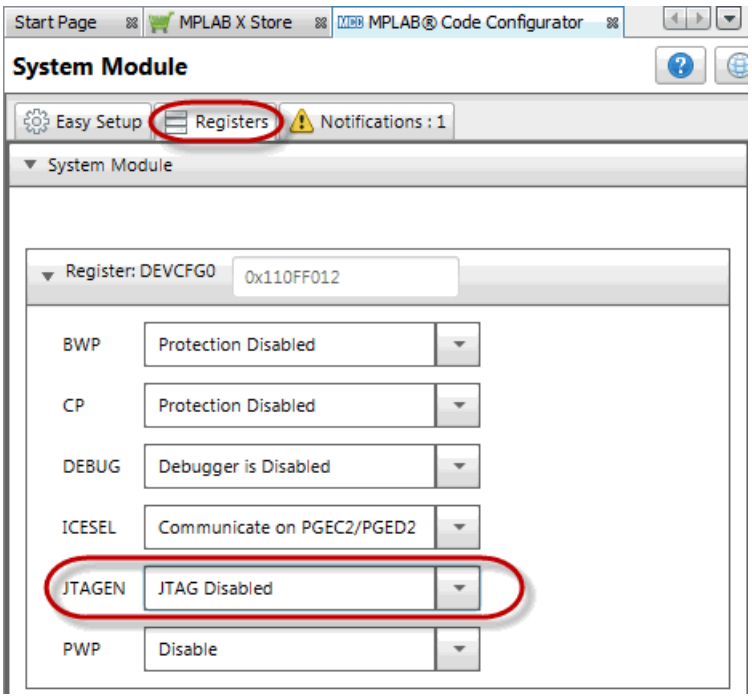


图28：　　闪存项目资源——NVM

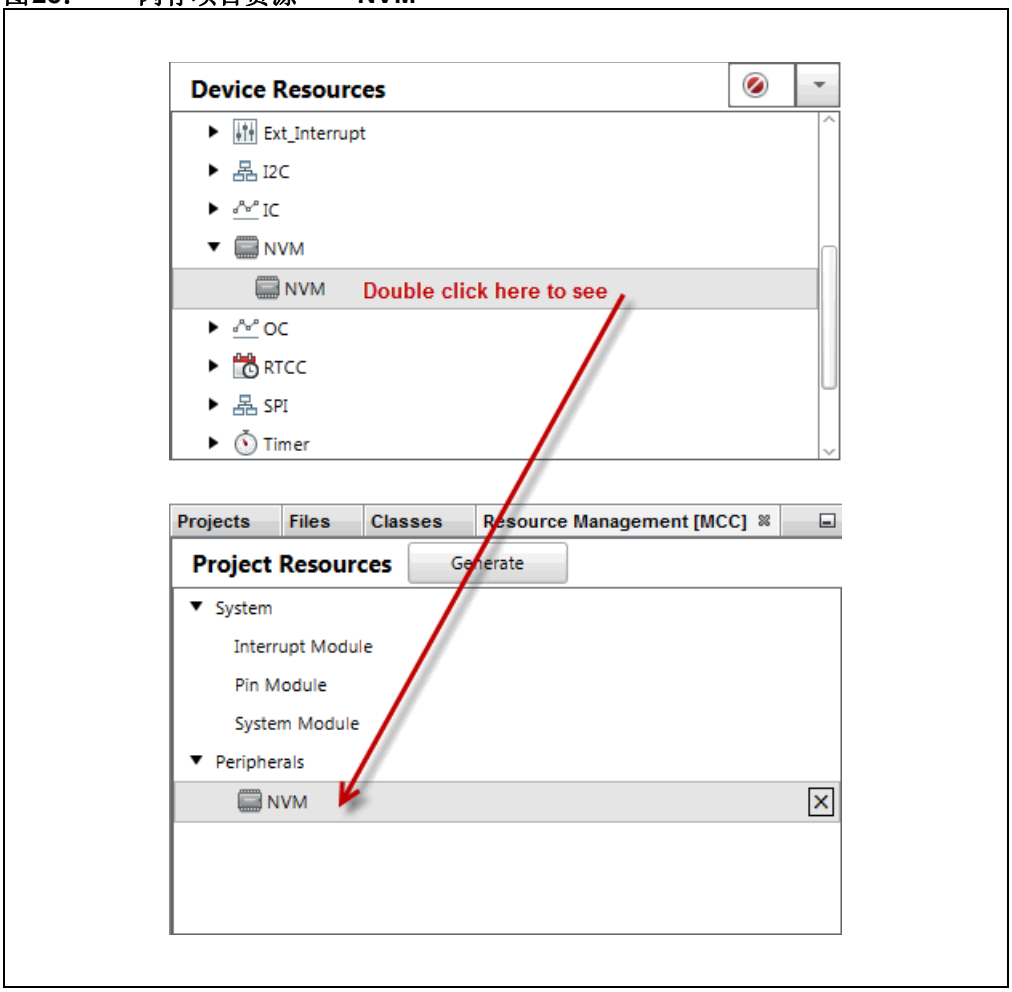


图29：　　闪存项目NVM寄存器

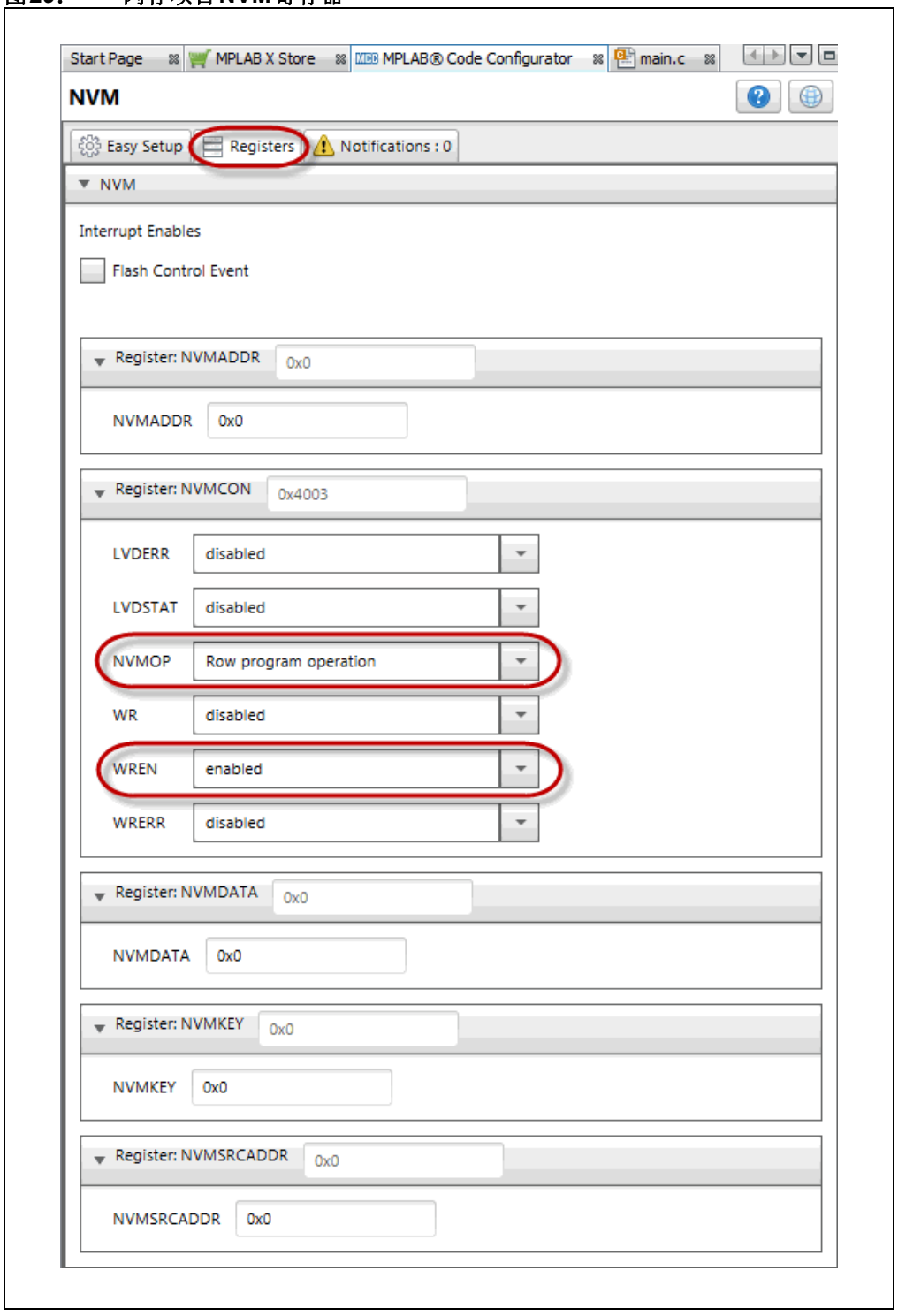


图30：　闪存项目资源——引脚模块

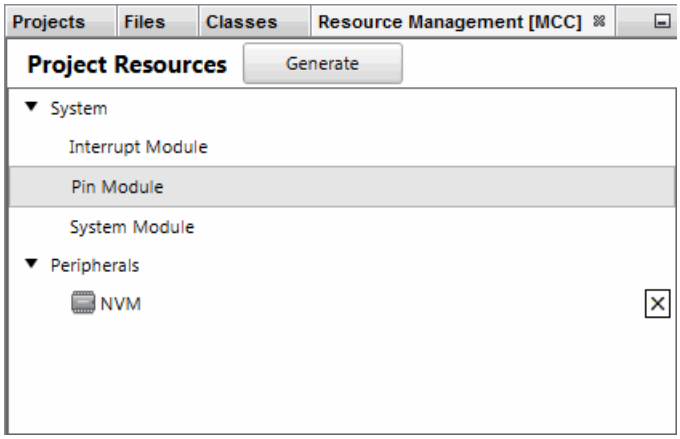
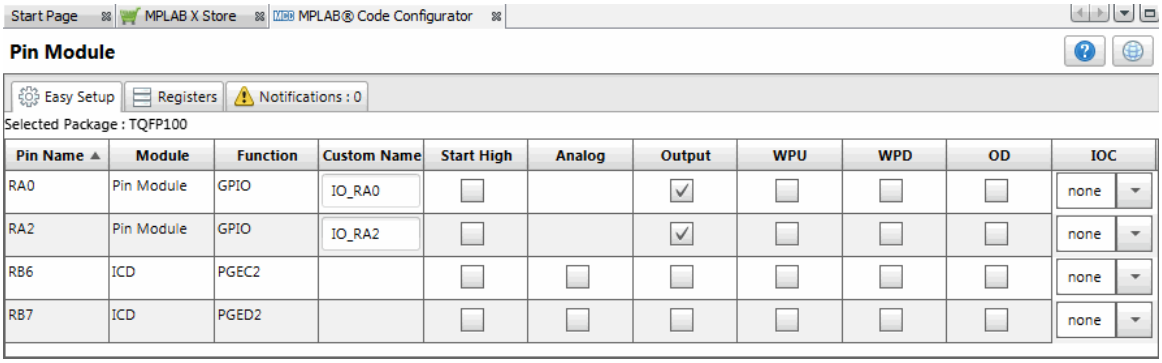
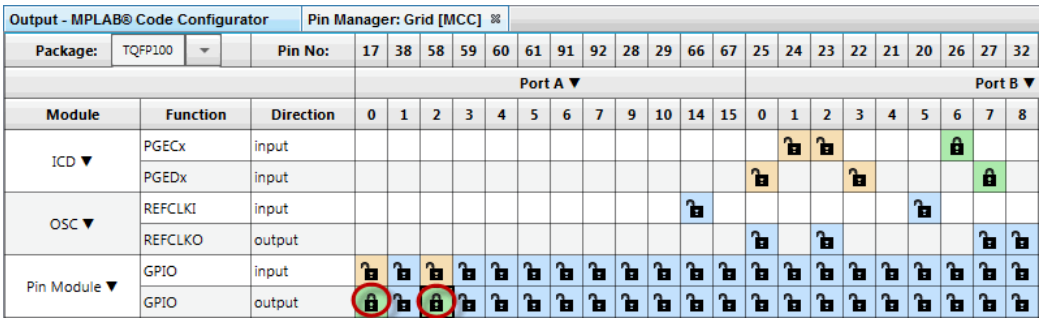


图31：　闪存项目I/O引脚配置



在图32中选择引脚RA0和RA2后，这些引脚将出现在上面的窗口中。
图26中选择了PGEC2/PGED2（即RB6和RB7）。
在窗口中显示后，可为每个引脚查看或选择引脚配置。

图32：　闪存项目I/O引脚资源



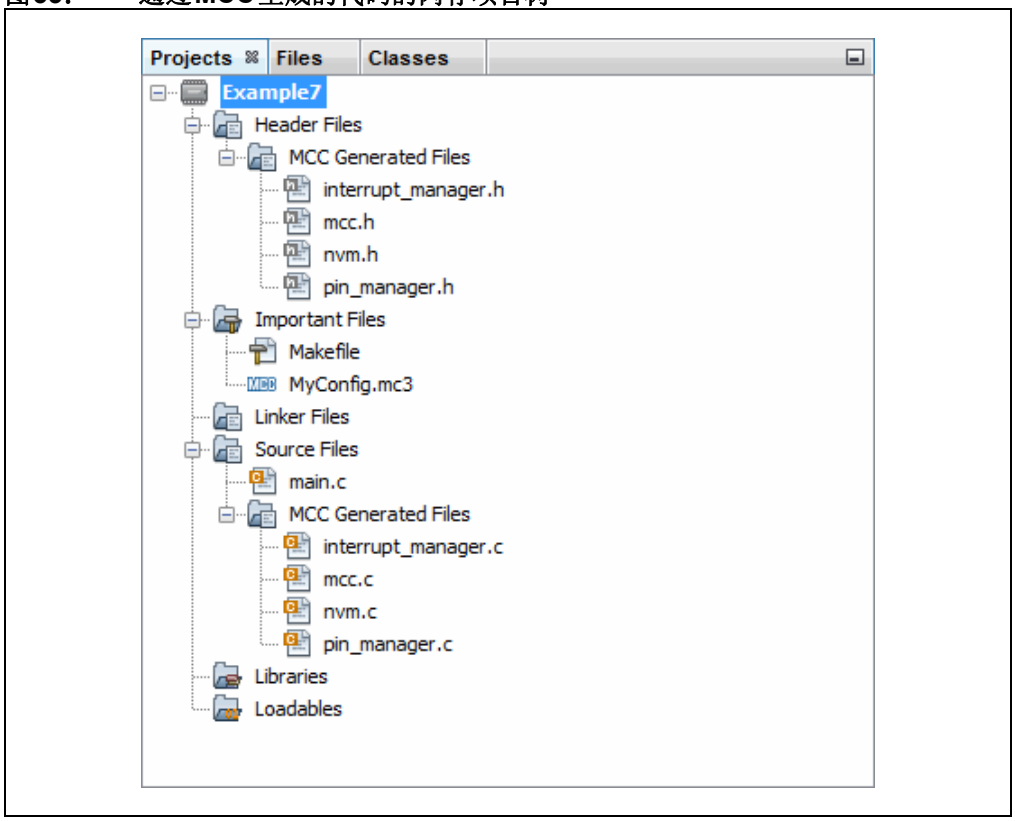
按照上述各图配置完代码后，请单击“Project Resources”窗口上的**Generate**按钮（图12）。通过MCC生成的代码是模块化的。因此，主函数代码、系统代码和外设代码均位于单独的文件中。此外，每个外设都有自己的头文件。

可通过生成中断管理器文件来捕捉潜在的错误。尽管本应用程序中不会使用中断，但会生成中断管理器文件供将来使用。

向程序中添加功能时始终需要编辑`main.c`。请查看生成的文件以找到您的代码中可能需要的任何函数或宏。

有关使用闪存的更多信息，请参见《PIC32系列参考手册》的第5章“闪存编程”（DS60001121G_CN）。

图33：　通过MCC生成的代码的闪存项目树



7.1 main.c修改后的代码

main.c模板文件已经过编辑，如下所示。部分注释用< >括起，表示已删除。已添加的代码以红色显示。

```
/**
  Generated Main Source File

<See generated main.c for file information.>
 */

/*
(c) 2016 Microchip Technology Inc. and its subsidiaries. You may use
this software and any derivatives exclusively with Microchip products.

<See generated main.c for additional copyright information.>
 */

#include "mcc_generated_files/mcc.h"

// Program Flash Physical Addresses:  0x1D00_0000 - 0x1D07_FFFF
// Program Flash Virtual Addresses:   KSEG0: 0x9D00_0000 - 0x9D07_FFFF
//                                    KSEG1: 0xBD00_0000 - 0xBD07_FFFF
#define NVM_PROGRAM_PAGE 0xbd008000

unsigned int databuff[128];

/*
                         Main application
 */
int main(void) {

    unsigned int x;

    // initialize the device
    SYSTEM_Initialize();

    // Fill databuff with some data
   for(x =0; x < sizeof(databuff); x++)
       databuff[x] = x;

   // Erase second page of Program Flash  <-- 请参见第7.2节
   NVM_ErasePage((void *)NVM_PROGRAM_PAGE);

   // Write 128 words starting at  <-- 请参见第7.3节
   // Row Address NVM_PROGRAM_PAGE
   NVM_WriteRow((void *)NVM_PROGRAM_PAGE, (void*)databuff);

   // Verify data matches  <-- 请参见第7.4节

   if(memcmp(databuff, (void *)NVM_PROGRAM_PAGE, sizeof(databuff)))
   {
       // If not turn led0 on to indicate an error
       IO_RA0_SetHigh();
   }

    else {
       // If true turn led2 on to indicate success
       IO_RA2_SetHigh();
    }
```

```
    while (1) {
       // End of program
    }

    return -1;
}
/**
 End of File
 */
```

7.2 擦除闪存页

闪存中可以擦除的最小部分是页。

在nvm.c文件中找到NVM_ErasePage()函数。

7.3 写入闪存行

数据缓冲区的内容将写入一个闪存行中。

在nvm.c中找到NVM_WriteRow()函数。

7.4 验证写操作并将数据显示在LED上

写入的数据与数据缓冲区的内容进行比较。如果内容不匹配，LED D0/D3将点亮，表示存在错误。如果内容匹配，Explorer 16/32电路板上的LED D2/D5将点亮，表示成功。

A. 在MPLAB X IDE中运行代码

对于示例1、2和3，按以下步骤创建一个项目：

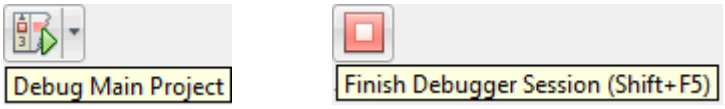
1. 启动MPLAB X IDE。
2. 从IDE中启动新建项目向导（*File>New Project*）。
3. 按照屏幕指示创建一个新项目：
   a) **Choose Project（选择项目）：**选择“Microchip Embedded”（Microchip嵌入式），然后选择“Standalone Project”（独立项目）。
   b) **Select Device（选择器件）：**选择示例器件。
   c) **Select Header（选择调试头）：**无。
   d) **Select Tool（选择工具）：**通过序列号（Serial Number，SN）SNxxxxxx选择硬件调试工具。如果调试工具名称下未显示SN，请确保调试工具已正确安装。有关详细信息，请参见调试工具文档。
   e) **Select Plugin Board（选择接插板）：**无。
   f) **Select Compiler（选择编译器）：**选择XC32（*最新版本号*）[*bin*文件夹位置]。如果XC32下未显示编译器，请确保编译器已正确安装且MPLAB X IDE可找到它。选择*Tools>Options*（工具>选项），单击**Build Tools**（编译工具）选项卡上的**Embedded**（已安装工具）按钮，查找您的编译器。有关详细信息，请参见MPLAB XC32和MPLAB X IDE文档。
   g) **Select Project Name and Folder（选择项目名和文件夹）：**为项目命名。
4. 右键单击Projects（项目）窗口中的项目名称。选择*New>Empty File*（新建>空白文件）。随即将显示New Empty File（新建空白文件）对话框。
5. 在“File name”（文件名）下输入名称。
6. 单击**Finish**（完成）。
7. 从本用户指南中剪切示例代码并粘贴到空白编辑器窗口中，然后选择*File>Save*（文件>保存）。

对于示例4和示例6，按照**第4.1节“在MPLAB X IDE中创建一个MPLAB Harmony项目”**指定的步骤创建一个项目。然后，按照规定设置MHC、生成代码并编辑应用程序。

对于示例5和7，请按照上面的步骤1到3进行操作。然后按照规定设置MCC、生成代码并编辑应用程序。

最后，选择Debug Run（调试运行）来编译代码、将代码下载到器件中以及执行代码。在LED上观察程序输出。单击Halt（暂停）结束执行。

图34：　工具栏图标



调试运行　　暂停

B. 获取软件和硬件

对于本文档中的MPLAB XC32项目，装有PIC32 PIM的Explorer 16/32板由9V外部电源供电，并使用标准（ICSP™）通信。使用MPLAB X IDE进行开发。

B.1 获取MPLAB X IDE和MPLAB XC32 C编译器

可从以下网址找到MPLAB X IDE v3.55及以上版本：
http://www.microchip.com/mplabx

可从以下网址找到MPLAB XC32 C编译器v1.42及以上版本：
http://www.microchip.com/mplabxc

B.2 获取MPLAB Harmony和配置器插件

可通过以下方式在MPLAB X IDE中找到MPLAB Harmony配置器v1.0.10.xx及更高版本：
*Tools>Plugins*中的**Available Plugins**选项卡。
可从以下网址找到MPLAB Harmony v1.10及以上版本：
http://www.microchip.com/mplab/mplab-harmony

B.3 获取MPLAB代码配置器（MCC）

可从以下网址找到MCC v3.26及以上版本：
http://www.microchip.com/mplab/mplab-code-configurator

B.4 获取PIC® MCU接插模块（PIM）

Microchip Technology网站上提供了示例中使用的PIC MCU PIM：
PIC32MX470F512L：http://www.microchip.com/MA320002-2

B.5 获取并设置Explorer 16/32板

Explorer 16/32开发板、原理图和文档可在以下网站上找到：
http://www.microchip.com/dm240001-2

下表列出了跳线和开关设置。

表1-1： 项目的跳线/开关选择

跳线/开关	选择	跳线/开关	选择
JP2	闭合	J37	断开
J19	断开	J38	断开
J22	断开	J39	默认
J23	默认	J41	断开
J25	闭合	J42	断开
J26	闭合	J43	默认
J27	断开	J44	默认
J28	断开	J45	默认
J29	断开	J50	闭合
J33	断开		

B.6 获取Microchip调试工具

可在如下开发工具网页中找到仿真器和调试器：
http://www.microchip.com/development-tools

B.7 获取示例代码

可从以下网址下载本文档中讨论的代码示例：
http://www.microchip.com/mplabxc

Documentation（文档）选项卡下。将MPLAB Harmony示例置于以下文件夹中：
`C:\microchip\harmony\v1_10\apps`

注：

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下， Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是 “牢不可破”的。

代码保护功能处于持续发展中。 Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案 （Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

Microchip 位于美国亚利桑那州 Chandler 和 Tempe 与位于俄勒冈州 Gresham的全球总部、设计和晶圆生产厂及位于美国加利福尼亚州和印度的设计中心均通过了 ISO/TS-16949:2009 认证。 Microchip 的 PIC® MCU 与 dsPIC® DSC、KEELOQ® 跳码器件、串行 EEPROM、单片机外设、非易失性存储器和模拟产品严格遵守公司的质量体系流程。此外，Microchip 在开发系统的设计和生产方面的质量体系也已通过了 ISO 9001:2000 认证。

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949 ═

ISBN：978-1-5224-2659-2

全球销售及服务网点

美洲

公司总部 Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 1-480-792-7200
Fax: 1-480-792-7277
技术支持：
http://www.microchip.com/support
网址：www.microchip.com

亚特兰大 Atlanta
Duluth, GA
Tel: 1-678-957-9614
Fax: 1-678-957-1455

奥斯汀 Austin, TX
Tel: 1-512-257-3370

波士顿 Boston
Westborough, MA
Tel: 1-774-760-0087
Fax: 1-774-760-0088

芝加哥 Chicago
Itasca, IL
Tel: 1-630-285-0071
Fax: 1-630-285-0075

达拉斯 Dallas
Addison, TX
Tel: 1-972-818-7423
Fax: 1-972-818-2924

底特律 Detroit
Novi, MI
Tel: 1-248-848-4000

休斯敦 Houston, TX
Tel: 1-281-894-5983

印第安纳波利斯 Indianapolis
Noblesville, IN
Tel: 1-317-773-8323
Fax: 1-317-773-5453
Tel: 1-317-536-2380

洛杉矶 Los Angeles
Mission Viejo, CA
Tel: 1-949-462-9523
Fax: 1-949-462-9608
Tel: 1-951-273-7800

罗利 Raleigh, NC
Tel: 1-919-844-7510

纽约 New York, NY
Tel: 1-631-435-6000

圣何塞 San Jose, CA
Tel: 1-408-735-9110
Tel: 1-408-436-4270

加拿大多伦多 Toronto
Tel: 1-905-695-1980
Fax: 1-905-695-2078

亚太地区

中国 - 北京
Tel: 86-10-8569-7000

中国 - 成都
Tel: 86-28-8665-5511

中国 - 重庆
Tel: 86-23-8980-9588

中国 - 东莞
Tel: 86-769-8702-9880

中国 - 广州
Tel: 86-20-8755-8029

中国 - 杭州
Tel: 86-571-8792-8115

中国 - 南京
Tel: 86-25-8473-2460

中国 - 青岛
Tel: 86-532-8502-7355

中国 - 上海
Tel: 86-21-3326-8000

中国 - 沈阳
Tel: 86-24-2334-2829

中国 - 深圳
Tel: 86-755-8864-2200

中国 - 苏州
Tel: 86-186-6233-1526

中国 - 武汉
Tel: 86-27-5980-5300

中国 - 西安
Tel: 86-29-8833-7252

中国 - 厦门
Tel: 86-592-238-8138

中国 - 香港特别行政区
Tel: 852-2943-5100

中国 - 珠海
Tel: 86-756-321-0040

台湾地区 - 高雄
Tel: 886-7-213-7830

台湾地区 - 台北
Tel: 886-2-2508-8600

台湾地区 - 新竹
Tel: 886-3-577-8366

亚太地区

澳大利亚 Australia - Sydney
Tel: 61-2-9868-6733

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

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

印度 India - Pune
Tel: 91-20-4121-0141

日本 Japan - Osaka
Tel: 81-6-6152-7160

日本 Japan - Tokyo
Tel: 81-3-6880-3770

韩国 Korea - Daegu
Tel: 82-53-744-4301

韩国 Korea - Seoul
Tel: 82-2-554-7200

马来西亚 Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

马来西亚 Malaysia - Penang
Tel: 60-4-227-8870

菲律宾 Philippines - Manila
Tel: 63-2-634-9065

新加坡 Singapore
Tel: 65-6334-8870

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

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

欧洲

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

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

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

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

德国 Germany - Garching
Tel: 49-8931-9700

德国 Germany - Haan
Tel: 49-2129-3766400

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

德国 Germany - Karlsruhe
Tel: 49-721-625370

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

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

以色列 Israel - Ra'anana
Tel: 972-9-744-7705

意大利 Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

意大利 Italy - Padova
Tel: 39-049-7625286

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

挪威 Norway - Trondheim
Tel: 47-7289-7561

波兰 Poland - Warsaw
Tel: 48-22-3325737

罗马尼亚 Romania - Bucharest
Tel: 40-21-407-87-50

西班牙 Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

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

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

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