

# **BTL101**

## **8-bit Bootloader Using MCC Classic**

### **Lab Manual for PIC16F18446**

# Table of Contents

Lab Equipment .....	3
Required Hardware .....	3
Required Software .....	3
Overview .....	4
Lab 1: Build and Run Demo Application .....	5
Lab 2: Generate UART Bootloader .....	7
Lab 3: Create Project Configurations .....	12
Lab 4: Fix the Configuration Word Problem .....	17
Lab 5: Bootloader Offset Application .....	19
Lab 6: Checksum Validation Method .....	21
Lab 7: Deep-Dive to Self-Write Protection .....	23
Reference .....	25

# Lab Equipment

## Required Hardware

Personal Computer (PC) with USB port and internet connection

USB-A to micro-USB cable

[DM164144](#) PIC16F18446 Curiosity Nano board

## Required Software

MPLAB® X IDE v6.15

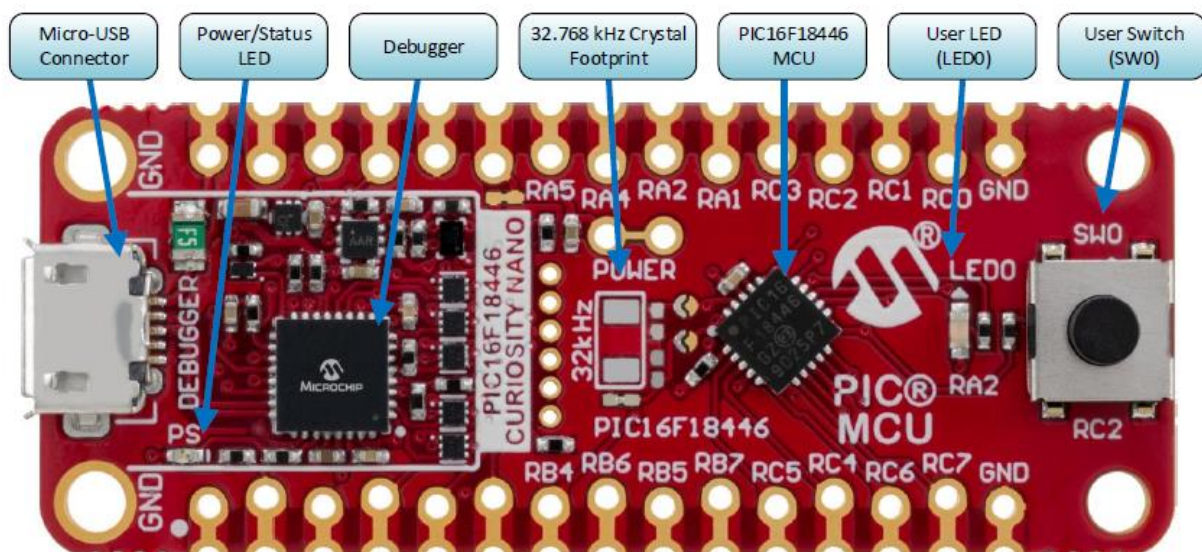
MPLAB® Code Configurator (MCC) Plugin v5.3.7

XC8 Compiler v2.41 or later

PIC16F1xxxx\_DFP v1.19.363

PKOB nano v1.11.554

MCC Core version v5.5.7



# Overview

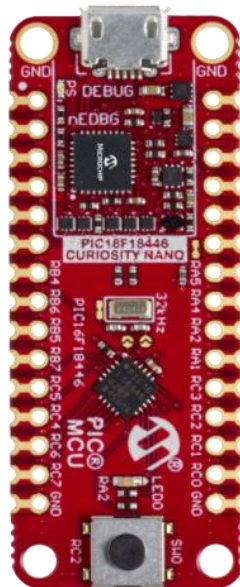
This lab manual will guide you through the process of generating a bootloader and integrating it with an end application. It will also show MPLAB X's project configurations feature to build the code with variations.

Specifically, we find it useful to build the application in three variations: Stand Alone, Offset to leave room for the bootloader, and Combined offset with the bootloader. Also we find two bootloader configurations useful: With and without configuration words.

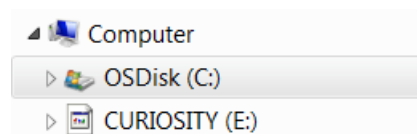
This manual is intended to remind the user how to set up the configurations once they get back to the office.

The labs are designed around the PIC16F18446 Curiosity Nano board ([DM164144](#)). This board was chosen because it does not require a separate programmer.

The top portion of the board is the on-board programmer/debugger and USB serial port bridge. That allows the PIC16F18446 to send data to a terminal program on a PC. The bottom portion contains the PIC16F18446 with a switch (SW0) and LED (LED0).



The board shows up as a removable drive “CURIOSITY” when connected to the computer via a USB cable.



Programming the PIC16F18446 is as simple as dragging and dropping the hex file from the project directory to the CURIOSITY disk.

# Lab 1: Build and Run Demo Application

## Purpose:

Lab 1 starts the process by verifying the provided end application works and look at some of the features of the end application necessary to work with the bootloader.

## Setup:


1. Connect both the PIC16F18446 Curiosity Nano board to the host via USB-A to micro-B cable.
2. Need: MPLAB X v6.15 or later with MCC v5.3.7 installed and XC8 v2.41 or later.

## Overview:

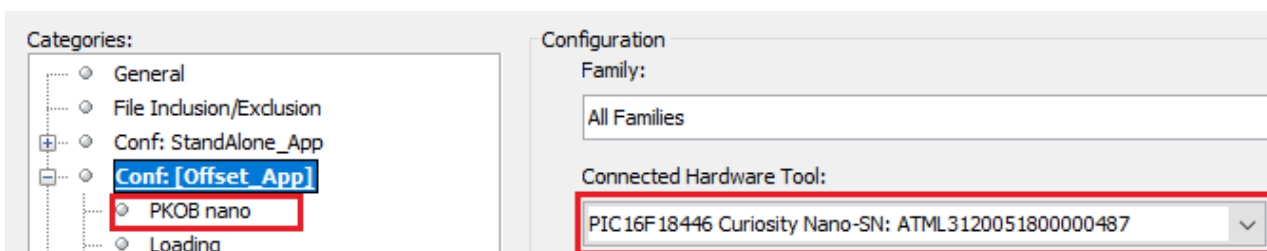
This introduces our end application. This represents the customer's widget and now wants to integrate a bootloader.

The application is a basic blinking LED project but also includes an escape to the bootloader. When SW0 is pressed the application will erase the "application valid" indications and reset the device. On restart the BL will note that no application is loaded and run the bootloader instead.

## Procedure (DEVICE):

1. Start MPLAB and open project in C:\RTC\BTL101\Labs\Demo\_APP\PIC16F18446\_Demo\_APP1.X
2. Build the project. 
3. Open a file browser window (Windows key/E). Navigate to the hex file (C:\RTC\BTL101\Labs\Demo\_APP\PIC16F18446\_Demo\_APP1.X\dist\default\production)
4. Drag **PIC16F18446\_Demo\_APP1.X.production.hex** to the CURIOSITY disk drive or use the on-board programmer(PKOB nano) to program the device.

Project Properties - PIC16F18446\_Demo\_APP1





5. Verify blinking LED pattern.

6. In MPLAB X, Open main.c.

7. Find code that tests the state of SW0. Note the code below invalidates all three supported application valid indications:

- \* Erase the Reset Vector.
- \* Erase the NV Flag and Checksum locations.

# Lab 2: Generate UART Bootloader

## Purpose:

To show how easy it is to generate a bootloader for an end application.

## Overview:

This set of procedures shows the typical steps you would have to take to create a custom bootloader.

## Procedure

1. Create a new directory in the C:\RTC\BTL101\Labs directory called "Lab2". We will create the bootloader here.
2. Create a new Microchip Embedded → Standalone Project  
Device: PIC16F18446  
Tool: Select PIC16F18446 Curiosity Nano as the programmer  
Compiler Toolchains: XC8 v2.41 or later  
Project name: "PIC16F18446\_Bootloader"

3. Start MCC



4. Click *Select MCC Classic* in the MCC Content Manager Wizard and then *Finish*.

## MCC Content Manager Wizard

1. Content Type 2. Required Device Content

### Select a Content Type

MCC Melody	MCC Classic	MPLAB® Harmony
Supports the MCC Builder Supports content versioning at driver level An iteration of MCC Generated Code Works both on- and off-line	Development process you are accustomed to All components and libraries that you have used before	Embedded Software Development Framework for 32-bit Microcontrollers and Microprocessors
<a href="#">Release notes and supported devices</a>	<a href="#">Release notes and supported devices</a>	<a href="#">Release notes and supported devices</a>

Library support may be a key factor in your choice of MCC flavor:

> MCC Melody and MCC Classic - Library Summary

> MPLAB Harmony - Library Summary

Still unsure which content type is right for your project?

[See More Details](#)

## MCC Content Manager Wizard



1. Content Type

2. Required Device Content

Finish


### Required Content

Some required content must be downloaded. The following content will be downloaded when you click on "Finish".  
To change content versions later, access the Content Manager from Device Resources.

Required Content			
Component 	Version	Update progress	Description
 Devices			



### Optional Content

Select optional content to be made available in Device Resources for selection

 Optional Content
--

5. In the System Module Window, Select Oscillator: HFINTOSC with HF Internal Clock = 4MHz and Clock Divider = 1:4.
6. Select EUSART1 in Device Resources Window.
7. Enable Transmit and Continuous Receive. The code will use autobaud so we don't have to set it here.

#### EUSART1

 Easy Setup  Registers

Hardware Settings

Mode asynchronous

☒ Enable EUSART

Baud Rate: 9600 Error: 0.160 %

☒ Enable Transmit

Transmission Bits: 8-bit

☐ Enable Wake-up

Reception Bits: 8-bit

☐ Auto-Baud Detection

Data Polarity Non-Inverted

☐ Enable Address Detect

☒ Enable Receive

8. Select Memory from the Device Resources window. This will give us #defines of the number of write latches, erase row size and flash memory size.
9. Select Bootloader from Libraries section in MCC Device Resources window. Ensure UART comms method, Source EUSART1.
10. Verification: Check\_State\_Flag.
11. Reset Vector: 0x400



Bootloader Generator

Easy Setup

User Defined Bootload Options

Transport Type: UART EUSART1

Verification: Check\_State\_Flag

Application Reset Vector: 0x400

Application Interrupt Vector: 0x404

IO Pin Indicator: Enabled

I/O Pin Entry: Disabled

Software Protection: Enabled

Flash Read: Disabled

EEData Read: Disabled

EEData Write: Disabled

12. Enable IO Pin Indicator and Software Protection.

13. In the **Pin Manager Grid** window, Select TX1 on Port B4 and RX1 on Port B6. Disable TX on Port C6.

Search Results	Output	Program Memory	Notifications [MCC]	Pin Manager: Grid View																
Package:	QFN20	Pin No:	16	15	14	1	20	19	10	9	8	7	13	12	11	4	3	2	5	6
			Port A ▼				Port B ▼				Port C ▼									
Module	Function	Direction	0	1	2	3	4	5	4	5	6	7	0	1	2	3	4	5	6	7
Bootloader G...	BL_INDIC...	output	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
EUSART1 ▼	RX1	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
	TX1	output	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
OSC	CLKOUT	output					🔒													
Pin Module ▼	GPIO	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
	GPIO	output	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
RESET	MCLR	input				🔒														

14. Select BL\_INDICATOR on Port A2.

Pin Module

Easy Setup Registers

Selected Package : QFN20

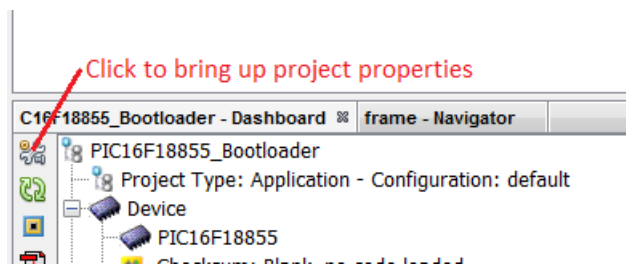
Pin Name ▲	Module	Function	Custom Na...	Start High	Analog	Output	WPU	OD	IOC
RA2	Bootloader ...	BL_INDICAT...		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none ▼
RB4	EUSART1	TX1		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none ▼
RB6	EUSART1	RX1		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none ▼

15. Select **Pin Module** in **Project Resources** window.

16. Deselect All analog functions.

17. Select Generate (disregard the warnings).

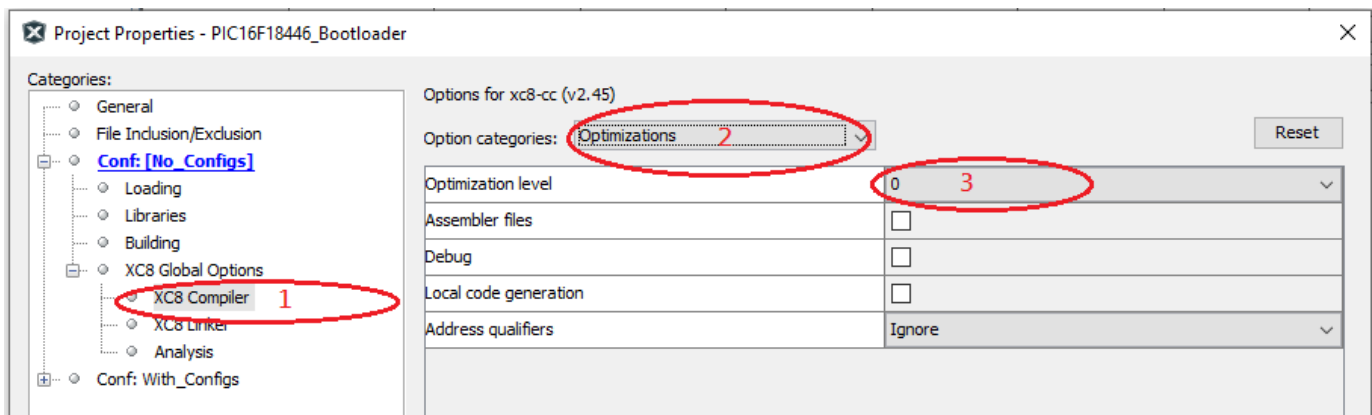
18. Click on the icon in the project dashboard to bring up the project properties window.



19. Click on **XC8 Compiler**

20. Select the **Optimizations** Pulldown Tab

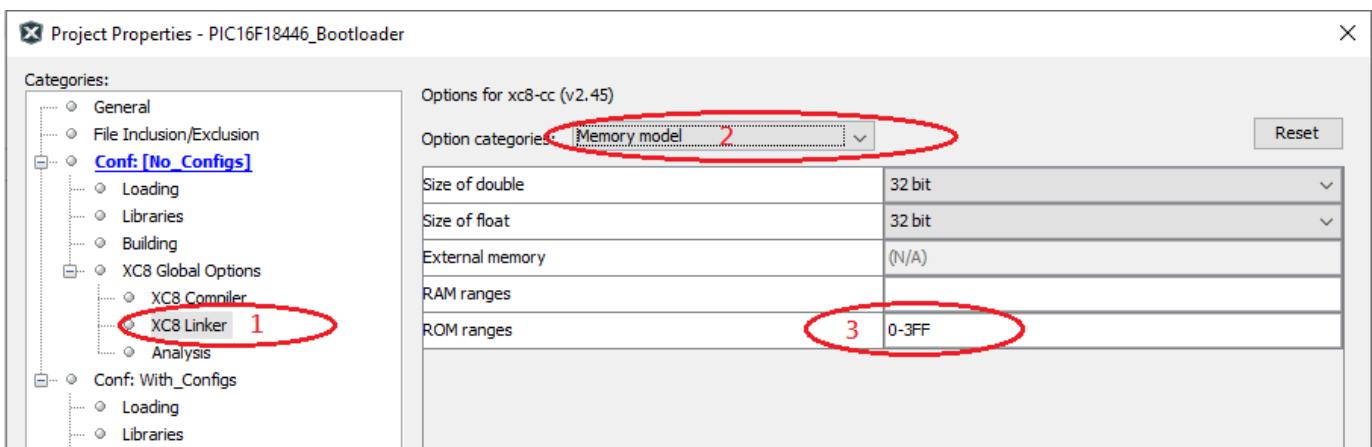
21. Select **optimization level '0'**



22. Click on **XC8 Linker**.

23. Select the **Memory Model** pulldown tab.

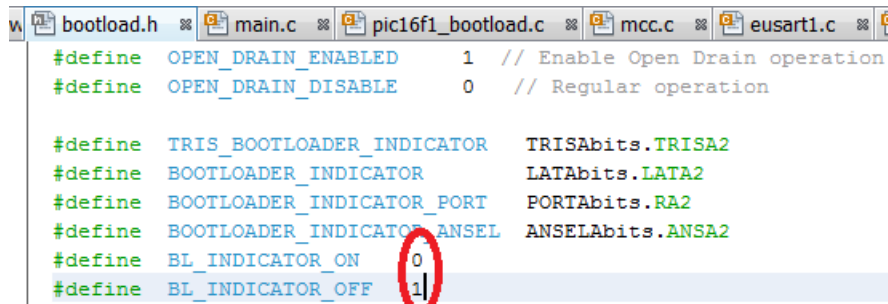
24. Enter **0-3FF** on the ROM ranges. This keeps the bootloader in the first 0x400 words of memory. Press Apply and OK.



25. Open file bootloader.h

26. Change the indicator #defines to reverse the sense of the indicator #defines.

Make BL\_INDICATOR\_ON 0 and BL\_INDICATOR\_OFF 1:



```
bootload.h  main.c  pic16f1_bootload.c  mcc.c  eusart1.c
#define OPEN_DRAIN_ENABLED 1 // Enable Open Drain operation
#define OPEN_DRAIN_DISABLE 0 // Regular operation

#define TRIS_BOOTLOADER_INDICATOR TRISAbits.TRISA2
#define BOOTLOADER_INDICATOR LATAbits.LATA2
#define BOOTLOADER_INDICATOR_PORT PORTAbits.RA2
#define BOOTLOADER_INDICATOR_ANSEL ANSELAbits.ANSA2
#define BL_INDICATOR_ON 0
#define BL_INDICATOR_OFF 1
```

27. Build the project

28. Navigate to the hex file (C:\RTC\BTL101\Labs\Lab2\PIC16F18446\_Bootloader\PIC16F18446 Bootloader.X\dist\default\production. Copy the .hex file to the CURIOSITY disk drive to program the device.

29. LED should be on solid indicating the bootloader is running.

## Lab Summary

This lab walked through the steps to generate and build a bootloader. The next lab will show how to combine the bootloader with the application.

# Lab 3: Create Project Configurations

## Overview

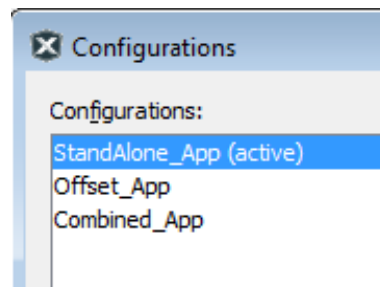
This lab will walk through the process to create three project configurations as discussed in the lecture. The StandAlone configuration would be the one originally developed. The Offset configuration builds the hex file that would later be bootloaded. The Combined configuration builds the offset user application combined with the bootloader. This is the version that would be programmed in at the factory.

## Procedure:

1. Right click on the Demo\_APP1 project and select "Set as Main Project" from the drop down menu.
2. Add "const char NVFlag \_\_at(FLASH\_MEM\_SIZE - 1) = 0x55;" to the demo app main.c before the main function.

```
43
44  #include "mcc_generated_files/mcc.h"
45
46  //Ken:2023/10/20
47  #define RESET_VECTOR 0x400
48  #define FLASH_MEM_SIZE 0x4000
49
50  void EraseResetVector ();
51  void Signal_BL_Requested ();
52
53  const char NVFlag __at(FLASH_MEM_SIZE - 1) = 0x55;
54
55  /*
```

3. Open the Demo\_APP1 project Properties window (Right click on project, project properties is at the bottom of the menu).
4. Select the Manage Configurations button (Bottom Left of the form).
5. Duplicate the default project twice so you have three configurations.
6. Rename the first configuration to "**StandAlone\_App**".
7. Rename the second to "**Offset\_App**".
8. Rename the third to "**Combined\_App**".



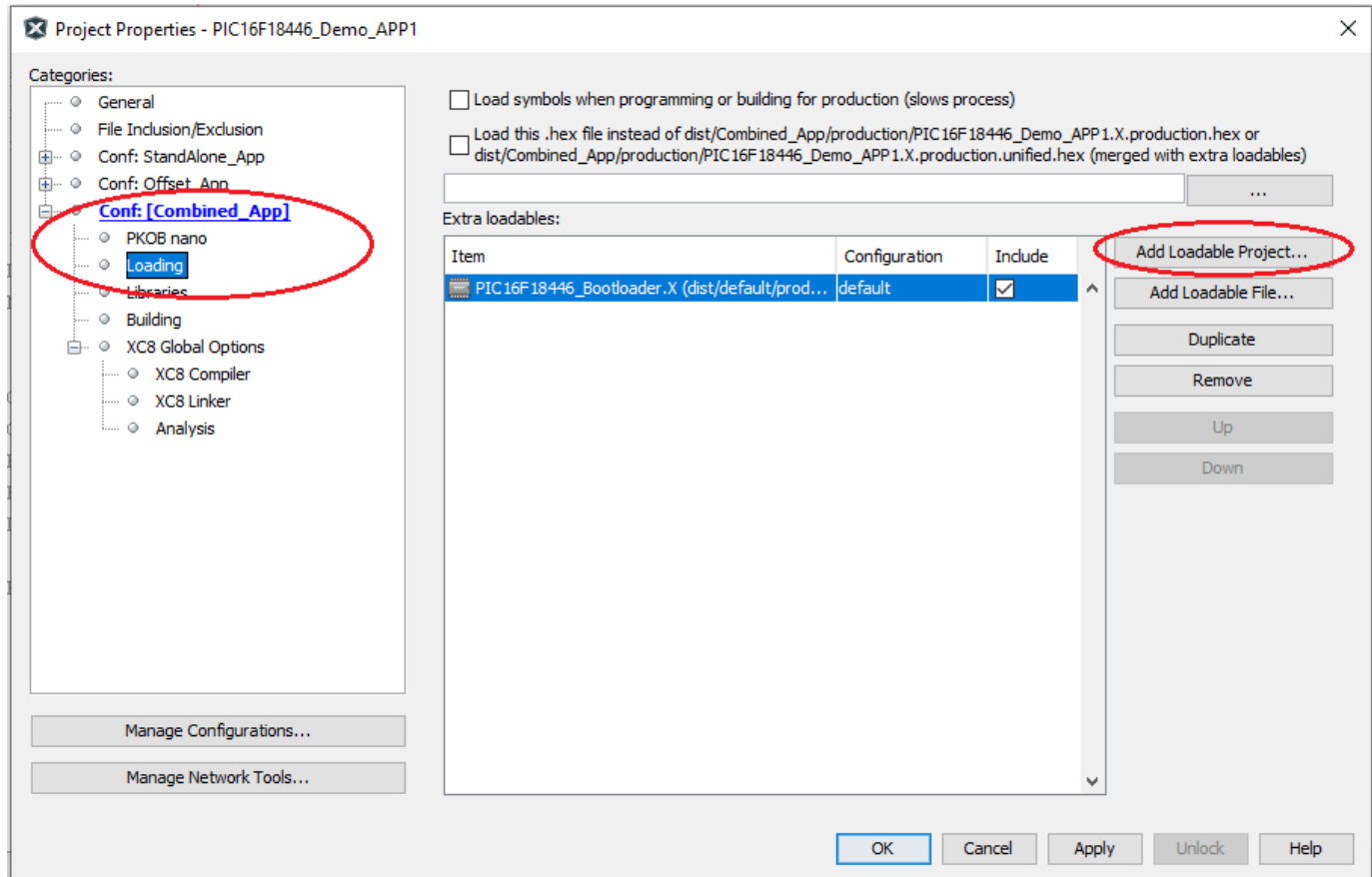
9. Click OK. You should be back to the project properties window.
10. Select XC8 Linker of the “Offset\_App” configuration, then select Additional options from the Options Categories drop down menu.
11. Offset project by 0x400. Click Apply.
12. Do the same to the Combined\_App configuration.

The screenshot shows the 'Project Properties' dialog for 'PIC16F18446\_Demo\_APP1'. The 'Categories' list on the left has 'XC8 Linker' selected. The main area is titled 'Options for xc8-cc (v2.45)'. The 'Option categories' dropdown is set to 'Additional options'. A table lists various linker options, with 'Codeoffset' set to '0x400'. Below the table is an 'Additional options' text field. At the bottom, there are tabs for 'Option Description' and 'Generated Command Line', with the latter being active. The dialog includes buttons for 'Manage Configurations...', 'Manage Network Tools...', 'OK', 'Cancel', 'Apply', 'Unlock', and 'Help'.

Options for xc8-cc (v2.45)	
Option categories:	Additional options
Extra linker options	
Serial	
Codeoffset	0x400
Checksum	
Errata	(N/A)
Trace type	(N/A)
Extend address 0 in HEX file	<input type="checkbox"/>
Use response file to link	<input type="checkbox"/>
Additional options:	
<div> <div>Option Description</div> <div>Generated Command Line</div> </div>	

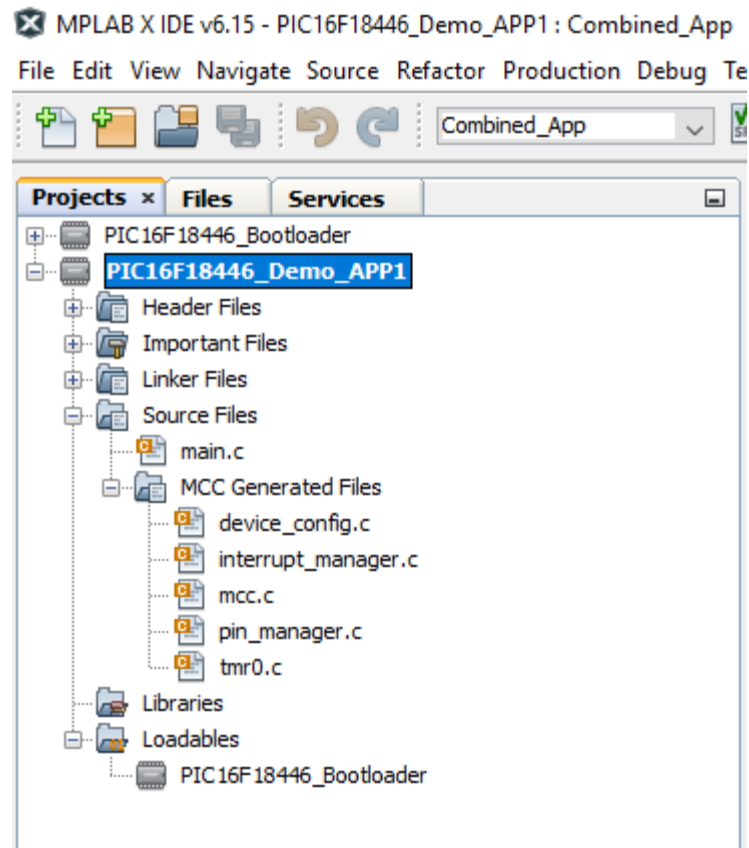
13. Select the “Loading” entry below the “Combined\_App” configuration.

14. Select the “Add Loadable Project” and navigate to the bootloader generated in Lab 2. Click ADD.



15. Click on “Apply” and “OK” to close the Project Properties window.

16. Select Offset\_App and do a clean and build of the project. This build will be used in Lab 5.



17. Now select the “Combined\_App” configuration and do a clean and build.

18. Which results in error “(1600) data conflict at address 1000Fh”. This is the “config word problem” discussed in the lecture. For now let's just comment out the config words in the bootloader and rebuild. We'll fix it properly in the next lab. You'll find the config words in **bootloader's** device\_config.c

Hints:

- Adding two forward slashes “//” to the beginning of a line comments out the line.
- MPLAB X allows highlighting a block of text, then Ctrl-Shift-C will toggle comments on the entire block.

19. Do a Clean and Build.

20. Navigate to the hex file directory

(C:\RTC\BTL101\Labs\Demo\_APP\PIC16F18446\_Demo\_APP1.X\dist\Combined\_App\production),

21. Drag and drop **PIC16F18446\_Demo\_APP1.X.production.unified.hex** file to the CURIOSITY disk drive.

22. The LED displays the blinking pattern indicating the application is running.
23. Press the SW0 button. The code erases the memory flag and resets the device. The steady LED indicates the bootloader is now running.

## Lab Summary

MPLAB X Project configurations are a powerful tool to help in the development of Application with a bootloader. The lab demonstrates loading the combined hex file. This is the version that would be factory programmed into the device.



# Lab 4: Fix the Configuration Word Problem

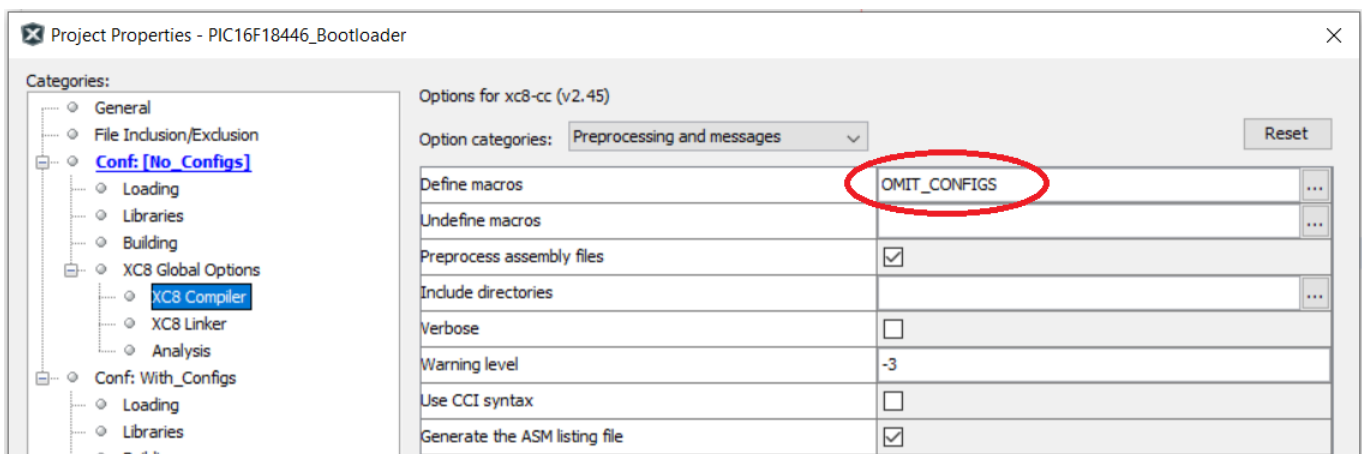
## Overview:

Before we go on, let's fix the config word problem. We'll do this by creating two configurations of the bootloader, one that will build with the config words and one without.

1. Go to file bootloader's `device_config.c` where you commented out the config words in step 18 of lab 3. Uncomment out the config words and add the following code around the config words.
2. Open the Bootloader Project Properties window.

```
#ifndef OMIT_CONFIGS
#warning "*****"
#warning "Configuration bits are not built with the bootloader."
#warning "*****"
#else
#warning "*****"
#warning "Configuration bits are built with the bootloader. If you get a \"(1600) data conflict at address 1000Fh\",  
Select the No_Configs configuration or add #define OMIT_CONFIGS."
#warning "*****"
{MCC.c configuration settings here}
#endif
```

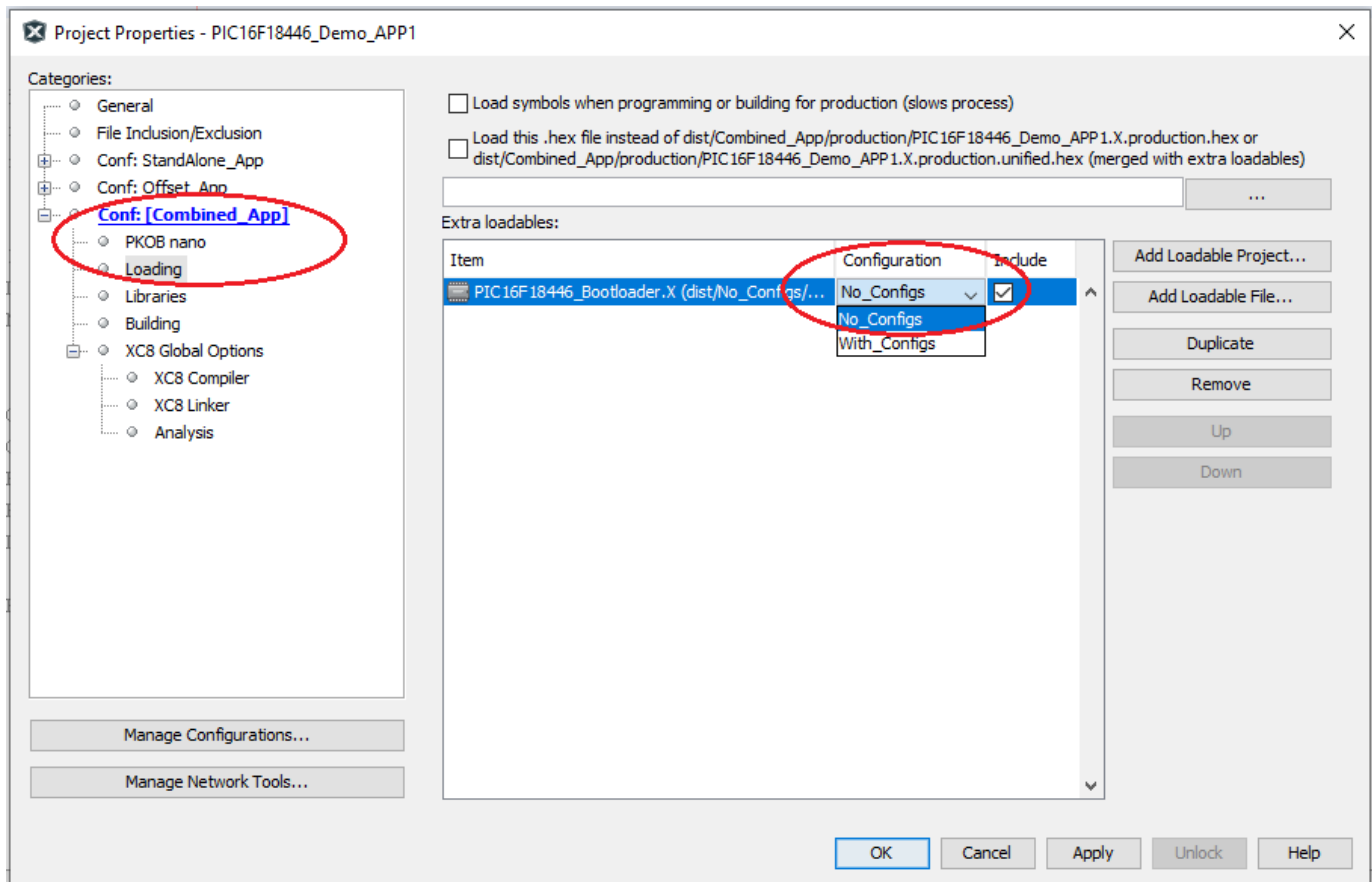
3. Press the Manage Configurations button.
4. Duplicate the default configuration.
5. Rename one configuration to **With\_Configs** and the other to **No\_Configs**. Click on OK.
6. Select the XC8 compiler tab in the No\_Configs configuration.
7. Add a Define macro "**OMIT\_CONFIGS**". Select "Apply". This will trigger the `#ifndef` in step 1 conditionally generate configuration words.



Now we have to go back to the Demo application project properties and tell it to use the bootloader without config bits.

1. Open the application project properties window.
2. Select the Combined\_App Configuration loading tab on the left
3. Select the No\_Configs configuration
4. Apply changes and build. Observe no error 1600 Data conflict at 1000Fh

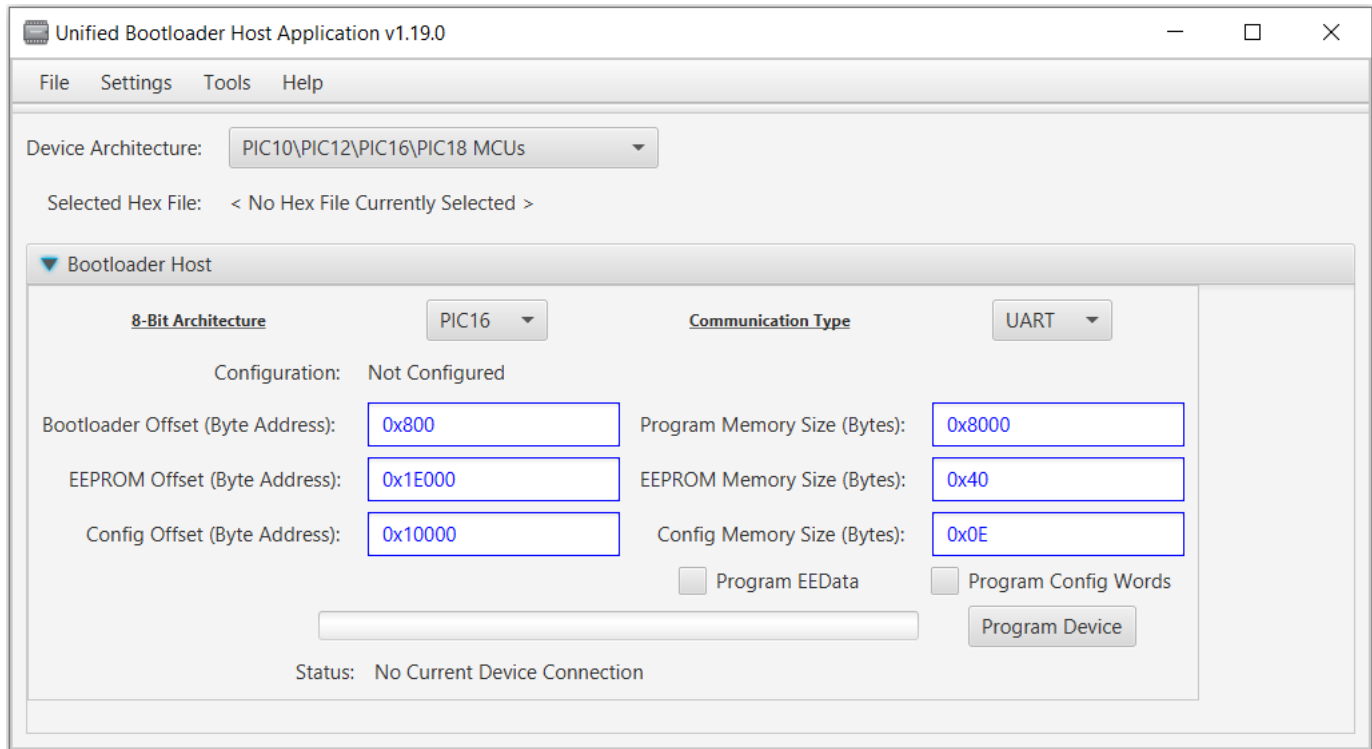
The configuration project is now solved.



# Lab 5: Bootloader Offset Application

## Overview:

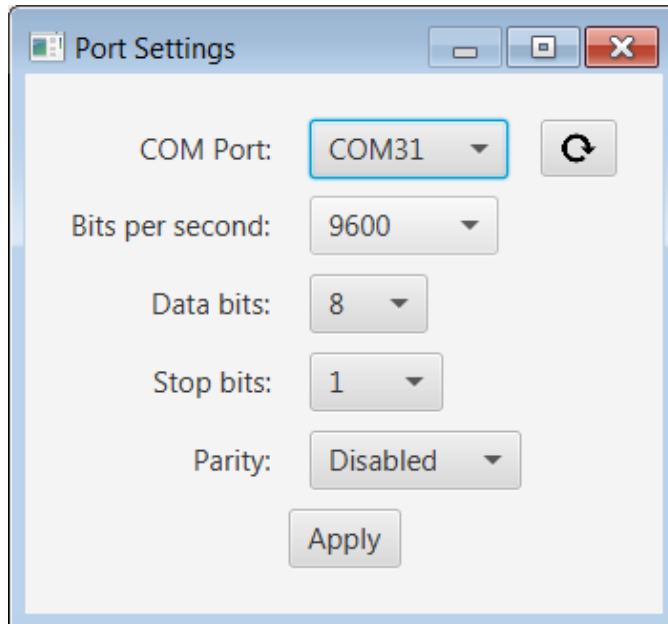
This lab uses the Unified Bootloader host application to load the application.



## Procedure:

1. If the LED is still blinking, press the SW0 button to switch to the bootloader.
2. Start the Unified Bootloader java application. (C:\RTC\BTL101\Bootloader Host\UnifiedHost-1.19.0\UnifiedHost-1.19.0.JAR). The download link as follows.  
<https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/SoftwareTools/UnifiedHost-1.19.0-bin.zip>
3. Select the 8 bit architecture(PIC10\PIC12\PIC16\PIC18 MCUs)

4. In the Settings menu, select Serial, then select correct UART for your CURIOSITY board.
5. Configure the port for 9600, 8 data bits, 1 stop. Parity Disabled. Press “Apply” to Close window.



6. Open console window from the tools pulldown menu.
7. The bootloader host works in byte address & length so we have to convert PIC16 word addresses by multiplying by 2. Set the program Memory Size to 0x8000 and the bootloader offset to 0x800.
8. Select the File Open/Load drop down. Navigate to and select the Offset\_App hex file generated in Lab 2. The path will be **C:\RTC\BTL101\Labs\Demo\_APP\PIC16F18446\_Demo\_APP1.X\dist\Offset\_App\production.hex**.
9. Press the “Program Device” button. The end device is reloaded and transferred control to the end application as in Lab 2.

## Lab Summary

This lab demonstrated reloading a device via the Unified Bootloader Host Application.

# Lab 6: Checksum Validation Method

## Overview:

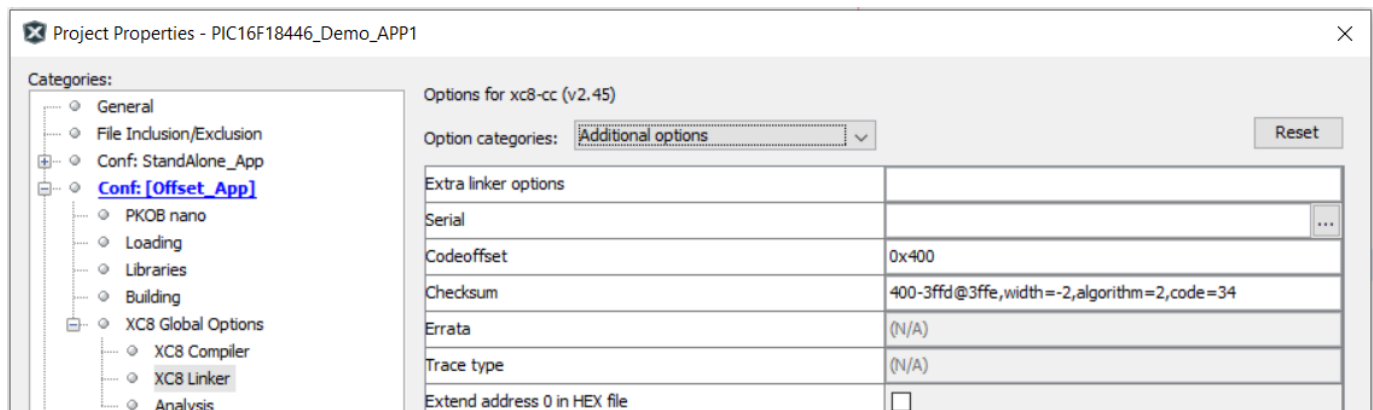
This lab will walk through the steps to embed an application checksum into the hex file and have the bootloader check that on startup.

## Procedure:

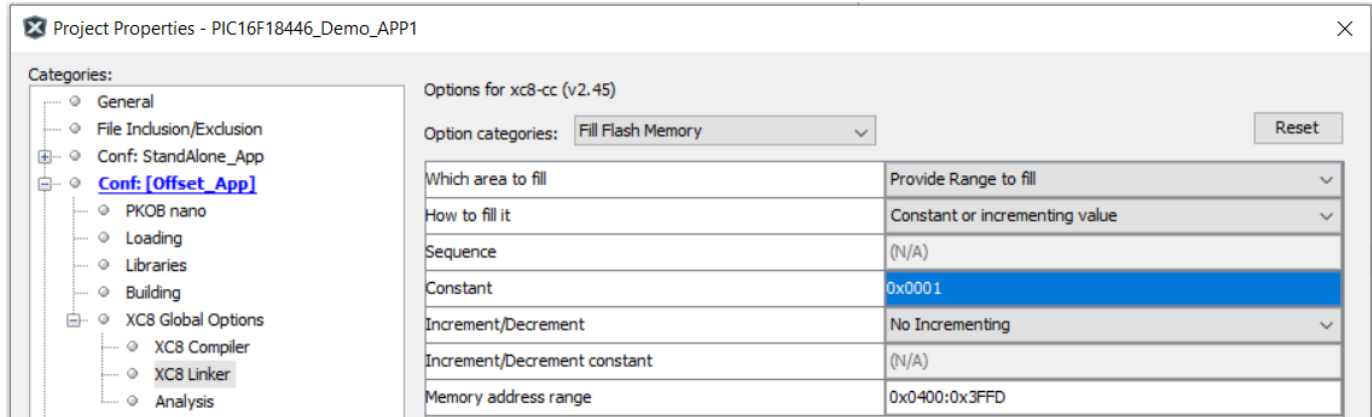
1. Generate a new bootloader as in Lab 2, but this time choose “Checksum” option in the Verification dropdown and make it the main project.
2. Open pic16F1\_bootload.c and inspect the code to verify the checksum method has been generated. Look for the call to Calc\_Checksum around line 240.
3. Build the new bootloader. Navigate to the hex file and drag and drop to the CURIOSITY Disk. The blinking pattern stops and only one LED is on steady indicating the bootloader is running.
4. Switch back to the Demo\_APP1. Comment out `const` line added in Lab 3 step 2.
5. Open the Demo\_APP1 → Project Properties screen. Under the Offset\_App configuration, select the Linker tab and select the Additional Options tab. Add the following to the Checksum line (no spaces):

```
0400-3ffd@3ffe,width=-2,algorithm=2,code=34
```

This line tells Hexmate to generate a checksum over the range of 0x300-0x3FFD and place it at 0x3FFE and 0x3FFF. The checksum is 2 bytes wide, low byte first. The high order bits of the word will be set to 0x34 and algorithm 2 is a 16-bit wordwise checksum.



6. Select Fill Flash Memory tab. For “Which area to fill” select “Provide Range to fill”, then for “How to fill it” select “constant or incrementing value”. Then for “Memory address range” enter 0x0400:0x3FFD. This is needed to generate code for the unused regions of memory so the checksum is calculated correctly. Suggest constant 0x0001. No incrementing. The 0x0001 is a device reset so if the MCU ever gets to a location we didn’t program, the device will reset.



7. Select the Offset configuration and do a clean and build.
8. Use the Bootloader host to load the Demo\_APP1 that now includes a checksum.
- Again, bootloader offset is 0x800 and Memory size = 0x8000.
9. The LED pattern should be blinking every 1sec, indicating the Demo\_APP1 is running.

## Lab Summary

This lab shows how easy it is to generate a checksum in the hex file and have the bootloader calculate it's own checksum and compare to the stored value.

# Lab 7: Deep-Dive to Self-Write Protection

## Overview:

This lab will deep-drive to self-write protection mechanism for bootloader and application project.

## Procedure:

1. Open the Demo\_APP1's device\_config.c. Change the “**Boot Clock Size Selection bit**” to 1024 words, enable the “**Boot Block Enable bit**”, enable the “**Boot Block Write Protection bit**” and enable the “**Configuration Register Write Protection bit**”.

```
71
72 // CONFIG4
73 #pragma config BBSIZE = BB1K // Boot Block Size Selection bits (1024 words boot block size)
74 #pragma config BBEN = ON // Boot Block Enable bit (Boot Block enabled)
75 #pragma config SAFEN = OFF // SAF Enable bit (SAF disabled)
76 #pragma config WRTAPP = OFF // Application Block Write Protection bit (Application Block not write protected)
77 #pragma config WRTB = ON // Boot Block Write Protection bit (Boot Block write protected)
78 #pragma config WRTC = ON // Configuration Register Write Protection bit (Configuration Register write protected)
79 #pragma config WRD = OFF // Data EEPROM write protection bit (Data EEPROM NOT write protected)
80 #pragma config WRTSAF = OFF // Storage Area Flash Write Protection bit (SAF not write protected)
81 #pragma config LVP = ON // Low Voltage Programming Enable bit (Low Voltage programming enabled. MCLR/Vpp pin function is MCLR.)
```

2. Open the Demo\_APP1 → Project Properties screen.
3. Select the **Offset\_App** configuration.
4. Click on **XC8 Linker**.
5. Select the **Memory Model** pulldown tab.
6. Change the **ROM ranges** from 400-3FFF to 400-3FDF (Preserve last page for checksum and this area is not available for App FW). Press “Apply”.

### Hints:

- Erase Page size is 32 words.
- Write Block size is 32 words.

7. Select the **Additional Options** tab. Change the Checksum line (no spaces) as below:

0400-3fdf@3ffe,width=-2,algorithm=2,code=34

8. Click on “Apply” and “OK” to close the Project Properties window.
9. Open the bootloader’s pic16f1\_bootload.c. Look for the call to **Bootload\_Required()** around line 177.
10. Change the data length of Demo\_APP1 project around line 184 as follows,  

```
frame.data_length = (END_FLASH - NEW_RESET_VECTOR - 32) << 1;
```
11. Open Demo\_APP1 project and select the **Combined\_App** configuration and press the “Program Device” button.
12. The LED pattern should be blinking every 1sec, indicating the Demo\_APP1 is running.
13. Press **SW0** button, MCU will jump to Bootloader from Application and then stay in Bootloader due to checksum is invalid now. **LED0** should be on solid indicating the bootloader is running.
14. Use the Bootloader host to load the Demo\_APP2. Again, bootloader offset is 0x800 and Memory size = 0x8000.
13. The LED pattern should be blinking every 0.5sec, indicating the Demo\_APP2 is running.

## Lab Summary

This lab shows how to achieve more robust self-write protection mechanism for both bootloader and application.



# Reference

1. Hexmate User's Guide. The checksum is generated via a program called Hexmate which is distributed with the XC8 compiler. The User's Guide explains each of the options on the Checksum line. The User's guide is section 8.3 of the XC8 Compiler User's Guide.

## 2. Debugging

- The host first gets the version of the end device to check communications.

If this fails:

- Verify the UART has both TX and RX enabled.
- Ensure analog is disabled on the pins.
- Check the PPS settings.

## 3. Bootloader Size

The bootloader used in this class were generated without optimizations enabled (opt.= 0). If this is available (opt.= s), the bootloader may build into the 0x0-0x2FF space used in this manual. To enlarge the bootloader area, make the following changes:

1. Change the bootloader ROM reservation in the linker properties from 0-3FF to 0-2FF.
2. Change the application offset in the application linker properties to 0x300
3. Change the #defines for NEW\_RESET\_VECTOR and NEW\_INTERRUPT\_VECTOR to 0x300 and 0x304 respectively.
4. Change the bootloader offset in the host program to 0x600 (bytes).

4. "CLEAN FAILED" compile error. If the build directory is open in a file explorer, the clean may fail. Either close the file explorer or just use "build" rather than the "Clean and Build".