Chapter 1

Quickstart

Microchip PIC32s are powerful microcontrollers that can be purchased for less than $10 in quantities of one. The PIC32 combines, in a single chip, a 32-bit central processing unit (CPU), RAM data memory, flash nonvolatile program memory, and many *peripherals* useful for embedded control, such as several channels for analog-to-digital conversion, digital I/O, synchronous and asynchronous serial communication, pulse-width modulated output, etc. The peripherals are what distinguish a microcontroller, like the PIC32, from a microprocessor.

The "32" in PIC32 refers to the 32-bit CPU architecture: the CPU operates on 32-bit instructions and registers, and the instruction and data buses are 32 bits wide. This means that 32 bits can be fetched from RAM simultaneously, for example. While 8-bit and 16-bit microcontrollers continue to be popular, a primary advantage of 32-bit microcontrollers is the greater computational horsepower they offer.

There are many different models of PIC32, currently arranged in four major families: the PIC32MX1xx/2xx family, the PIC32MX3xx/4xx family, the PIC32MX5xx/6xx/7xx family, and the most recent PIC32MZ family. Representatives of the four families include the PIC32MX250F128D, the PIC32MX460F512L, the PIC32MX795F512L, and the PIC32MZ2048ECM144. The three MX families all use the MIPS32 M4K processor (up to 80 MHz) as the CPU while the MZ family uses the MIPS32 microAptiv microprocessor unit (up to 200 MHz) as the CPU. "MIPS32" is a CPU architecture with an associated assembly language instruction set licensed by Microchip from Imagination Technologies.

Each family consists of a number of different models, differing in the number of pins, the amount of RAM and flash available, and the number and type of peripherals available. The chips come with anywhere from 28 to 144 pins, and while most models are available only in surface mount packages, some in the MX1xx/2xx family come as "dual inline packages" (DIPs) that can be plugged directly into a solderless breadboard. The MX3xx/4xx family is the original PIC32 family; the MX5xx/6xx/7xx family offers additional support for CAN bus and ethernet communication; and the more recent MX1xx/2xx family, while generally having fewer peripherals available, offers interfaces for connecting to audio devices and capacitive-based touch sensors. The MX1xx/2xx family also offers the most flexible mapping of different pins to different functions (Peripheral Pin Select). The MZ family is a significantly new design, incorporating the faster microAptiv CPU and many of the best features of the MX families, including Peripheral Pin Select.

While the four families share many features, and most of this book applies to all families, where there are differences we focus on the MX5xx/6xx/7xx family. Also, we will often find it convenient to cite specific numbers, such as the amount of RAM and flash memory available, and in these cases we will use the PIC32MX795F512L as our model. This is the PIC32 used on the NU32 development board as well as the Microchip PIC32 USB Starter Kit II and PIC32 Ethernet Starter Kit.

The PIC32MX795F512L features a max clock frequency of 80 MHz, 512 KB program memory (flash), and 128 KB data memory (RAM). It also features 16 10-bit analog-to-digital input lines (multiplexed to a single analog-to-digital converter, or ADC), many digital I/O channels, USB 2.0, Ethernet, two CAN modules, five I^2C and four SPI synchronous serial communication modules, six UARTs for RS-232 or RS-485 asynchronous serial communication, five 16-bit counter/timers (configurable to give two 32-bit timers and one 16-bit timer), five pulse-width modulation outputs, and a number of pins that can generate interrupts based on external signals, among other features.

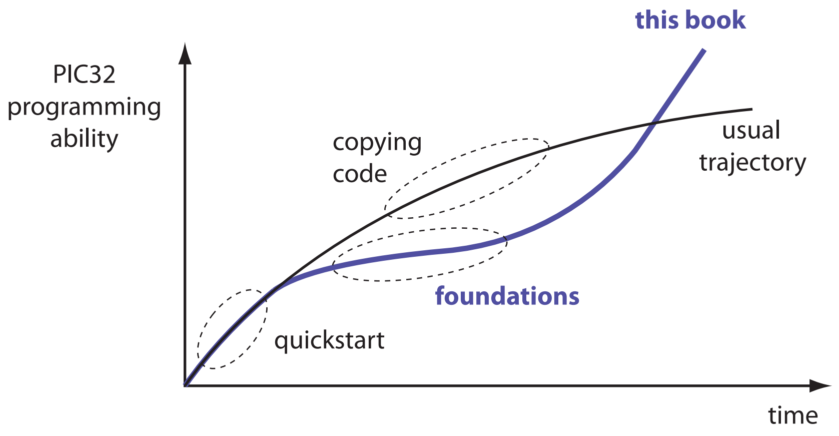


Figure 1.1: The trajectory of PIC32 programming ability vs. time for the usual "copy and modify" approach vs. the foundational approach in this book. The crossover should occur at only a few weeks!

The purpose of this chapter is to make sure you have everything you need to run your first simple programs. A deeper exploration of the hardware and software is left to the following chapters.

1.1 Philosophy

When starting out with the PIC32, the Microchip documentation can be confusing to navigate. A primary goal of this book is to start you off in a more organized way that enables you to make effective use of the Microchip documentation. The philosophy of this book is summed up in Figure 1.1. While most programmers new to the PIC32 can achieve some basic functional ability quickly by modifying existing code, the lack of foundational understanding encouraged by the "copy and modify" approach can lead to an early plateau in the ability to use the PIC32, making it difficult to create and debug more complex projects. This book delays some initial gratification in exchange for a more solid foundation for continued exploration of the PIC32, even if you have no previous background in microcontrollers. Chapters 2–6 focus on example-driven foundations, and with this as background, you can move more quickly through later chapters on peripherals.

1.2 Reference Reading

In this chapter we start quickly, so no need to read any other reference material now. But throughout the book you will learn to rely on the Microchip documentation, so it a good idea to download it for later use.

- **The relevant PIC32 Family Data Sheet.** We will focus on the PIC32MX5xx/6xx/7xx Family Data Sheet, but the other families share many common features.
- **The individual sections of the PIC32 Reference Manual.** Search for "Microchip Reference Manual," and on the Microchip page, search for the sections with "PIC32" in the title. As of this writing, there are approximately 40 sections (with section numbers up to 52) totaling over 1600 pages. These sections generally apply to all four families, so they can sometimes be confusing in their generality. Some of the sections, particularly the later ones, focus on the PIC32MZ family and are not relevant to the PIC32MX795F512L.
- **(Optional) The Microchip MPLAB XC32 C Compiler User's Guide and Assembler, Linker, and Utilities User's Guide.** These come with your XC32 C compiler installation (Section 1.3), so no need to download separately.
- **(Optional) MPLAB Harmony Help.** Harmony is a set of software libraries and drivers provided by Microchip to simplify programming of the PIC32. Since its purpose is to abstract away from the

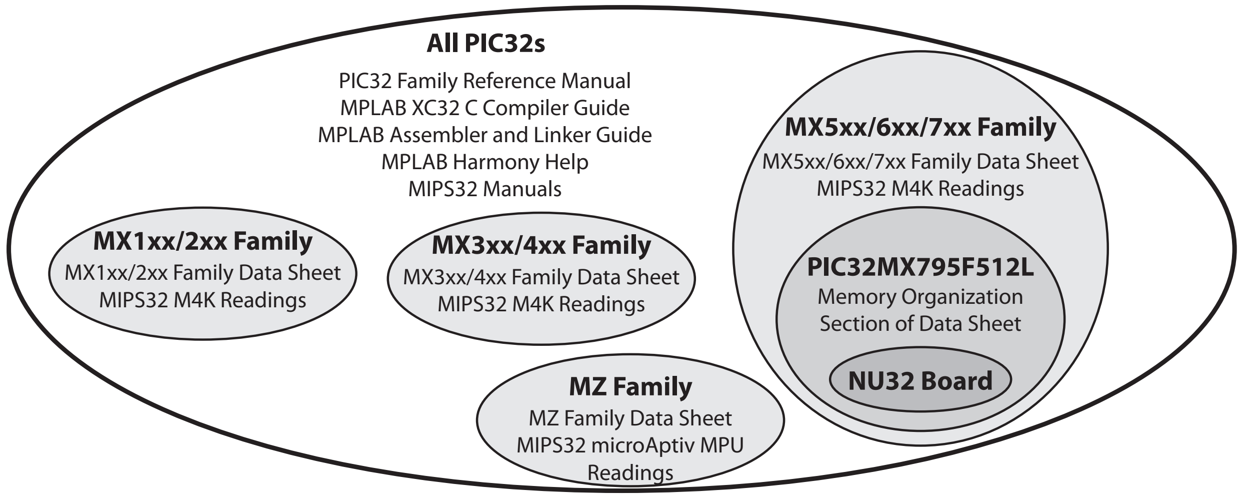


Figure 1.2: Reference reading for the PIC32.

specifics of the particular PIC32 model, and since this book is heavily driven by understanding the hardware, we will have limited use for Harmony.

- **(Optional) MIPS32 Architecture for Programmers manuals and other MIPS32 documentation.** If you are curious about the MIPS32 CPU architectures and the assembly language instruction set, you can find references online. These are not necessary, however.

The scope of the various reference readings is summarized in Figure 1.2. The MPLAB C compiler and assembler/linker user's guides are software reference manuals. The PIC32 Family Reference Manual is the best resource for understanding how the PIC32 hardware works. However, since it is written to apply to all PIC32s, some material is not relevant for all models. For instance, not all PIC32s have all the *special function registers* (SFRs, covered beginning in Chapter 2) mentioned in each Reference Manual section.

The Data Sheets provide details specific to the PIC32 families. In addition, for each particular PIC32 model in a family, such as the PIC32MX795F512L in the MX5xx/6xx/7xx series, the Memory Organization section of the Data Sheet clarifies which SFRs are present on that model, and therefore which Reference Manual functions are available for that model.

Flipping back and forth between sections of the Reference Manual, your particular Data Sheet, and the software user's guides is not a good way to get started programming the PIC32. But neither is simply copying and modifying code. This book provides a more organized introduction to programming the PIC32 and prepares you for future exploration using Microchip documentation.

In many places the material in this book is general to all PIC32s. At others the material is specific to the MX5xx/6xx/7xx family; at others it is specific to the PIC32MX795F512L; and at still others it is specific to the PIC32MX795F512L as it is used on the NU32 development board. To prevent the presentation from being hopelessly filled with caveats, exceptions, and generalities, in most of the book we assume you have a PIC32MX795F512L on an NU32 development board.

1.3 What You Need

1.3.1 Hardware

In this chapter we start with the PIC32 by getting some simple code up and running quickly. To do this, you need the following three things:

1. **A host computer with a USB port.** The host computer is used to create PIC32 programs. Any operating system is fine.

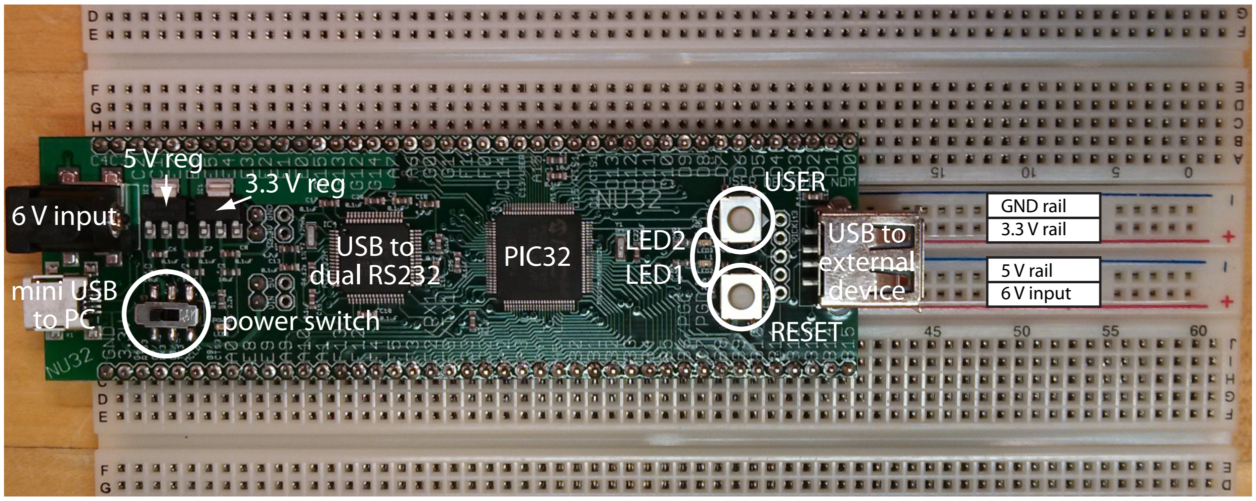


Figure 1.3: A photo of the NU32 development board.

2. **A PIC32 board (and possibly a separate programmer device).** The PIC32 needs some external electronics to function, and these are typically provided by a small printed circuit board on which the PIC32 is mounted. These boards often have buttons and LEDs for simple input and output. Example boards include:

   (a) **Starter kits.** Microchip's various PIC32 Starter Kits, such as the PIC32 Starter Kit, PIC32 USB Starter Kit II, PIC32 Ethernet Starter Kit, Microstick II, and PIC32MX1/MX2 Starter Kit consist of boards with the PIC32, some I/O devices for user interface, and an onboard **programmer** that programs the flash memory of the PIC32 using a USB connection to the host computer.

   (b) **Development boards.** A development board is similar to a starter kit, except that there is no programmer onboard. The PIC32's flash memory is programmed by a separate external programmer device, such as Microchip's ICD 3 or PICkit 3, which connects to a USB port of the host computer. Examples of development boards include Microchip's PIC32 plug-in modules coupled with the Explorer 16 Development Board, as well as the NU32 and UBW32 development boards.

   (c) **Development boards with a PIC32 with an installed "bootloader."** If the PIC32 on the development board already has a **bootloader** installed in program memory, the PIC32 can be programmed without an external programmer device. When the PIC32 is powered on, the bootloader runs, and (typically) depending on whether or not the user is pushing a button, it either (1) jumps to a "user" program that has already been installed somewhere else in program memory, or (2) receives a new user program from the host computer (e.g., via a USB connection) and writes the new program to program memory. You can use the bootloader to change the user program as often as you want. There is no need for an external programmer device, only a "bootloader communication utility" on the host computer that talks to the bootloader. Examples of development boards with preloaded bootloaders include the NU32 and the UBW32.

   Most of the examples in this book use the NU32 development board with an installed bootloader (Figure 1.3). Most code is easily adapted to most PIC32s, however.

1.3.2 Software

Regardless of the particular PIC32 development board, you need the following free Microchip software:

- **The Microchip XC32 compiler, which includes Microchip's C software library.** The XC32 compiler (also called the XC32++ compiler) is used to turn your C programs into executables that can be installed directly into PIC32 program memory.
- **The Microchip MPLAB X IDE.** The MPLAB X Integrated Development Environment provides a front end to the XC32 compiler.

For the examples in this book, you also need the following software:

- **The FTDI Virtual COM Port (VCP) Driver.** Download and install the driver appropriate to your operating system from FTDI Chip's website. This driver allows you to use a USB port as a "virtual COM port" to talk to the NU32 board.
- **PIC32 starter code.** We have packaged together code you need for the examples in this chapter in a single `.zip` file called `PIC32quickstart.zip`. After unzipping, you get the files
  - `nu32utility-win.c`, `nu32utility-mac.c`: Programs for your computer to communicate with the PIC32 over a USB cable. Use the first one if you are using Windows, the second if you are using a Mac.
  - `makefile-win`, `makefile-mac`: Instructions for building executables. Keep the one appropriate for your OS and rename it simply as `makefile`.
  - `simplePIC.c`: The first program you will compile, to install using a bootloader.
  - `NU32bootloaded.ld`: A linker script for programs installed using a bootloader.
  - `simplePIC_standalone.c`: A version of `simplePIC.c` to be installed using a programmer, not a bootloader.
  - `talkingPIC.c`: A program that allows the PIC32 to accept keyboard input from your computer and to send characters back to your computer.
  - `NU32.c`: A library of convenient functions for the NU32 board.
  - `NU32.h`: A header file for the NU32 library.
- **An RS-232 terminal emulator program.** An RS-232 terminal emulator provides a simple I/O interface to a virtual COM port on your computer. Characters you type on your keyboard can be sent by the COM port to the PIC32, and output from the PIC32 can be received by the COM port and displayed on your screen. Terminal emulators include CoolTerm, PuTTY (Windows), and the built-in `screen` emulator for Linux and Unix (Mac OS). There are many others to choose from.

The following software is optional:

- **MPLAB Harmony.** Harmony is a collection of libraries and drivers that comes independent of the XC32 distribution. Using Harmony libraries simplifies the task of writing code that can operate on many different PIC32 models, but we won't use it much.

1.4 Building a Bootloaded Executable

If your NU32 already has a bootloader installed on it, you can install a new executable using a communication utility to talk to the PIC32 using a USB cable. You do not need an external programmer device.

First you need to create the bootloader communication utility executable on your computer. In Windows, do

```
gcc nu32utility-win.c -o nu32utility
```

and in Mac OS, do

```
gcc nu32utility-mac.c -o nu32utility
```

Then move the executable `nu32utility` to an appropriate directory on your computer where you can always find it.

To determine which communication (COM) port you will use to program the PIC32, do the following at the command line:

- **Mac**: Type `ls /dev/tty.*`. Take note of the list of names of COM ports.
- **Windows**: Type `mode`. Take note of the list of names of COM ports.

Now provide power to the NU32 board, turn the power switch on, and verify that the red power LED comes on. Connect the USB cable from the NU32's mini B USB jack (next to the power jack) to a USB port on the host computer. Now do the same steps as above, and note that there are now two new COM ports. For the Mac, they will look something like

```
/dev/tty.usbserial-00001024A      /dev/tty.usbserial-00001024B
```

while for Windows they will look something like

```
COM4   COM5
```

For the Mac, the bootloader COM port is the one labeled B, while for Windows it is the one with the larger number.

To put the NU32 into program receive mode, first locate the RESET button and the USER button on the NU32 board (Figure 1.3). The RESET button is the one adjacent to the pins labeled B8, B9, B10, on the left bottom side of the board if the power jack is at the top, and the USER button is at the bottom right of the board, next to the pins labeled D5, D6, and D7. Press both buttons, then release the RESET button, then the USER button. When the PIC32 wakes up after the RESET and sees that you are pressing the USER button, the bootloader knows that you are planning to program the PIC32. You should see LED1 flashing a "heartbeat" to show the bootloader is ready for your program.

Now you have two options for building the executable that will be loaded by `nu32utility`: (1) using the command line or (2) using the MPLAB X IDE. The command line is simple and convenient, while the IDE provides some helpful tools for understanding and debugging your code.

1.4.1 Using the Command Line

Create a new directory with the files

- `simplePIC.c`. The C source code.
- `NU32bootloaded.ld`. This is a custom "linker script" that specifies where the executable program should be placed in the PIC32's program memory. We will learn more about linker scripts in Chapter 3.
- `makefile`. This makefile is one of `makefile-win-bootloader` or `makefile-mac-bootloader`, depending on your operating system. Make sure you rename it simply as `makefile`.

The `makefile` contains the rules for building your executable. Find the three lines

```
CC = /Applications/microchip/xc32/v1.30/bin/xc32-gcc
HX = /Applications/microchip/xc32/v1.30/bin/xc32-bin2hex
WRITE = ~/NU32/nu32utility
```

They will look somewhat different in Windows. Notably, the directory separator character is \. Edit these three lines so that they contain the proper paths to your `xc32-gcc`, `xc32-bin2hex`, and `nu32utility`.

You also need to edit the line defining the COM PORT. On the Mac, this line looks something like

```
PORT = /dev/tty.usbserial-00001024B
```

and on Windows it looks something like

```
PORT = \\\.\COM23
```

Make sure you replace these with the name of the appropriate port, as discovered above.

At the command line in this directory, type

```
make out.hex
```

You should now see `out.hex`, as well as `simplePIC.o`, in your directory. The file `out.hex` is the executable.

Assuming your NU32 is already in program receive mode with its "heartbeat" flashing, as described above, you can now type

```
make write
```

and the `nu32utility` sends the program to the PIC32. After it has completed, the new program `out.hex` runs. The two LEDs of the NU32 alternate on and off unless you press the USER button, which freezes them. The program listing is given below in Code Sample 1.1. We will learn more about the operation of this program in Chapter 3.

Code Sample 1.1. `simplePIC.c`. Blinking lights on the NU32, unless the USER button is pressed.

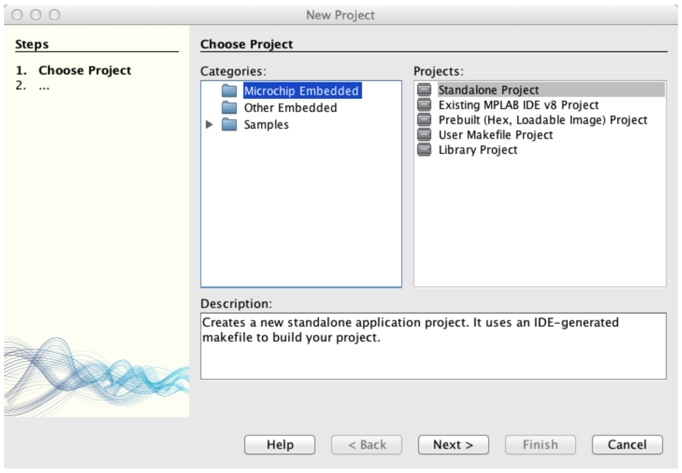
```
#include <plib.h>

void delay(void);

int main(void) {
  DDPCONbits.JTAGEN = 0; // Disable JTAG, make pins 4 and 5 of Port A available.
  TRISA = 0xFFCF;        // Pins 4 and 5 of Port A are LED1 and LED2.  Clear
                         // bits 4/5 to zero, for output.  Others are inputs.
  LATAbits.LATA4 = 0;    // Turn LED1 on and LED2 off.  These pins sink ...
  LATAbits.LATA5 = 1;    // ... current on NU32, so "high" = "off."

  while(1) {
    delay();
    LATAINV = 0x0030;    // toggle the two lights
  }
  return 0;
}

void delay(void) {
  int j;
  for (j=0; j<1000000; j++) { // number is 1 million
    while(!PORTDbits.RD13);   // Pin D13 is the USER switch, low if pressed.
  }
}
```

Now that you can build and load the program, try changing `simplePIC.c`, saving, rebuilding, and reloading. For example, you could change the number of cycles in the `delay` loop to 2 million. Remember always to put the NU32 in receive mode (RESET while holding the USER button) before typing `make write`. It suffices to do `make write`, which builds and loads the executable; it is not necessary to type `make out.hex` first.

1.4.2 Using the MPLAB X IDE

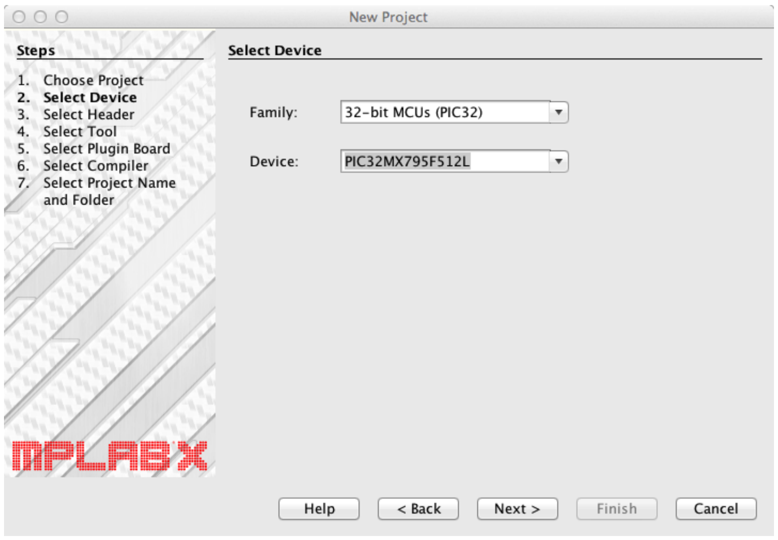
For this project you need two files, `simplePIC.c` and `NU32bootloaded.ld`. You will use the MPLAB X IDE to create a "project" folder to hold these files as well as other files, such as the final executable.

Step 1. Launch MPLAB and choose **File** > **New Project...** Select the Category **Microchip Embedded** and the type of project as **Standalone Project** and click **Next**. (In this book we use the word "standalone" to mean a program that is loaded using a programmer, not a bootloader, which is different than the usage in MPLAB.)



Creates a new standalone application project. It uses an IDE-generated makefile to build your project.

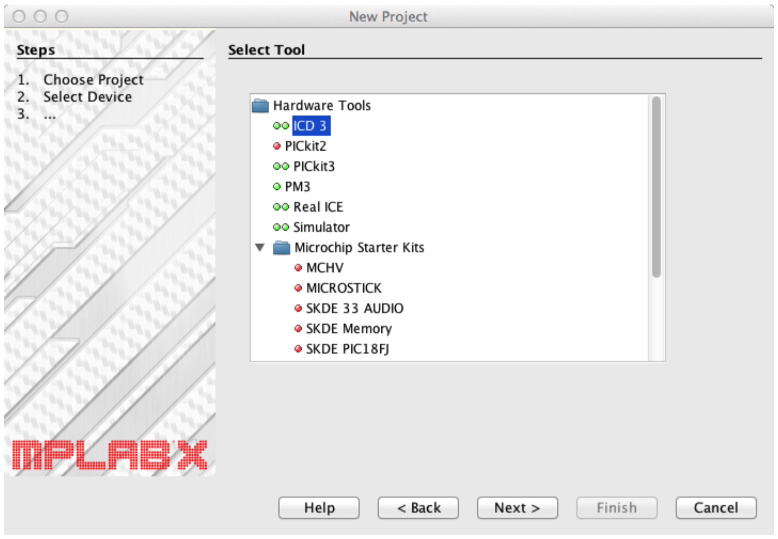
Step 2. Choose the PIC32 family and the device PIC32MX795F512L.



Family: 32-bit MCUs (PIC32)

Device: PIC32MX795F512L

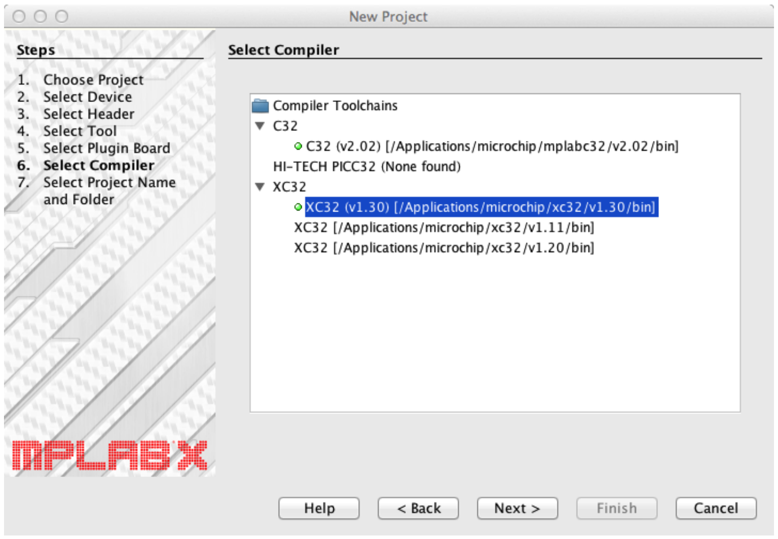
Step 3. Since we will not use a programmer device, the Hardware Tool does not matter. Leave the default.



Hardware Tools
- ICD 3
- PICkit2
- PICkit3
- PM3
- Real ICE
- Simulator

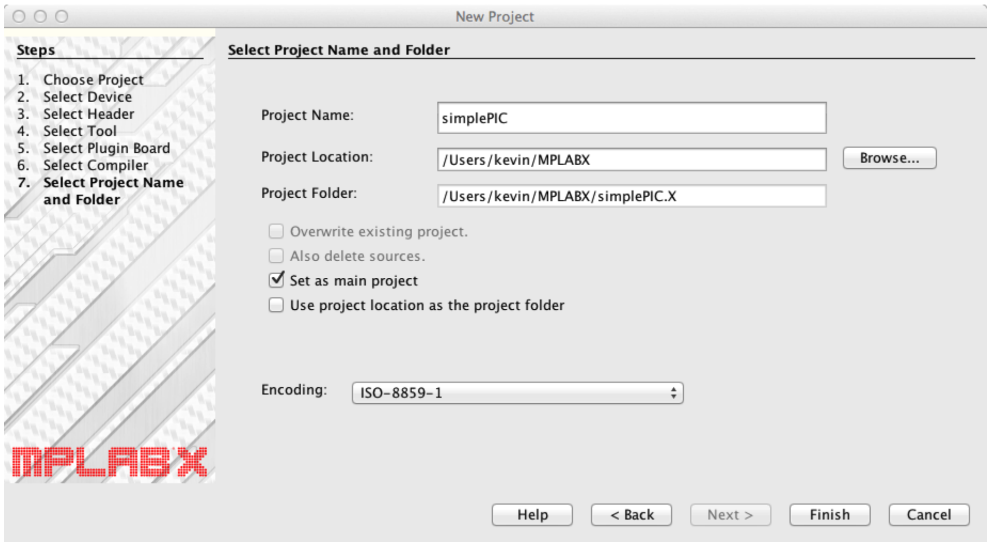
Microchip Starter Kits
- MCHV
- MICROSTICK
- SKDE 33 AUDIO
- SKDE Memory
- SKDE PIC18FJ

Step 4. Choose the XC32 compiler. If you have multiple versions of the XC32 compiler, choose the most recent one.



Compiler Toolchains
- C32
  - C32 (v2.02) [/Applications/microchip/mplabc32/v2.02/bin]
- HI-TECH PICC32 (None found)
- XC32
  - XC32 (v1.30) [/Applications/microchip/xc32/v1.30/bin]
  - XC32 [/Applications/microchip/xc32/v1.11/bin]
  - XC32 [/Applications/microchip/xc32/v1.20/bin]

Step 5. Type `simplePIC` as the Project Name and choose a Project Location, a directory that will hold all of your PIC32 projects. This automatically creates a Project Folder named `simplePIC.X` in your Project Location. In this example, the Project Location is called `MPLABX`. You should also check the **Set as main project** box. Click **Finish**.


New Project

Steps
1. Choose Project
2. Select Device
3. Select Header
4. Select Tool
5. Select Plugin Board
6. Select Compiler
7. Select Project Name and Folder

Select Project Name and Folder

Project Name: simplePIC

Project Location: /Users/kevin/MPLABX

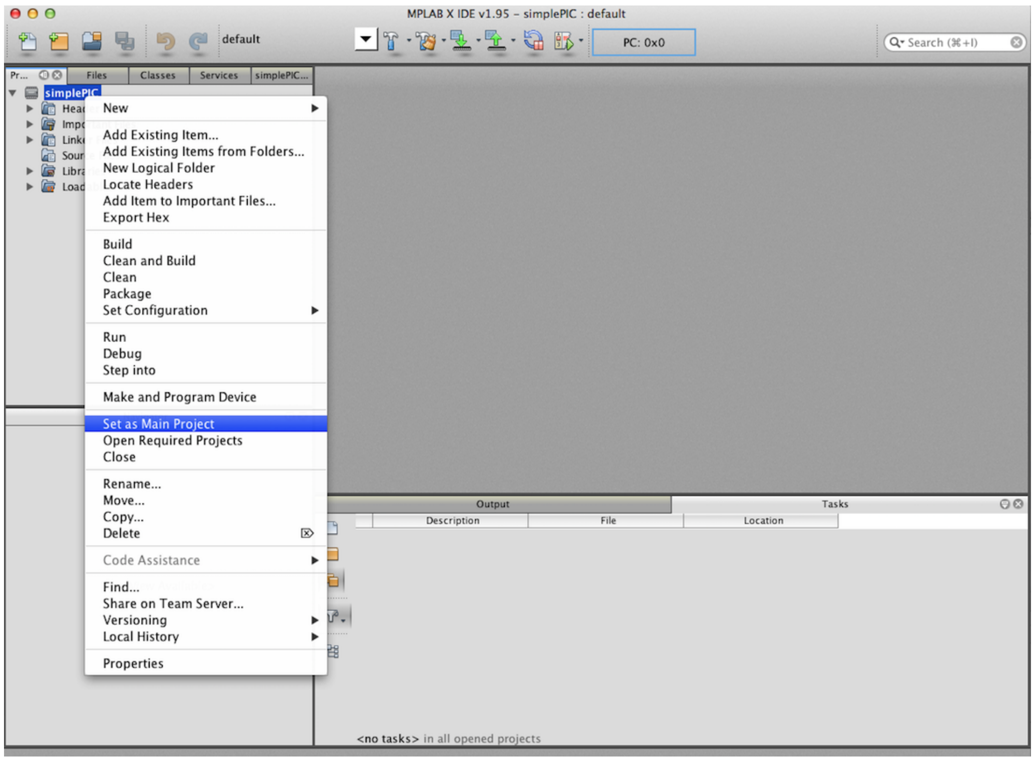
Project Folder: /Users/kevin/MPLABX/simplePIC.X

[ ] Overwrite existing project.
[ ] Also delete sources.
[✓] Set as main project
[ ] Use project location as the project folder

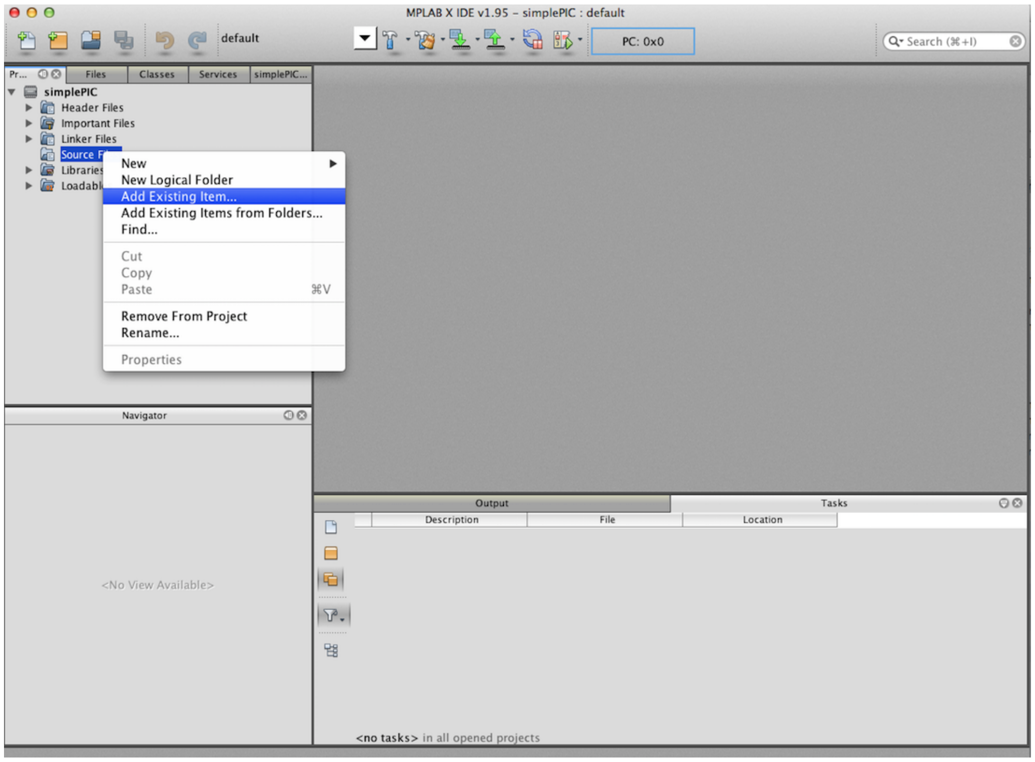
Encoding: ISO-8859-1

Step 6. Copy the files `simplePIC.c` and `NU32bootloaded.ld` to your new `simplePIC.X` directory.

Step 7. If this is the first time you've used the IDE, your project `simplePIC` will be the only project in the left column. If there are other projects, make certain that `simplePIC` is selected as the main project by right-clicking (or on a Mac, ctrl-clicking) on the project name `simplePIC` and selecting **Set as Main Project**. (In future, when you have multiple projects in the left column, double-check that the one you are working on is selected as the **Main Project**. Otherwise the project you build may be different from the one you think you are building.)



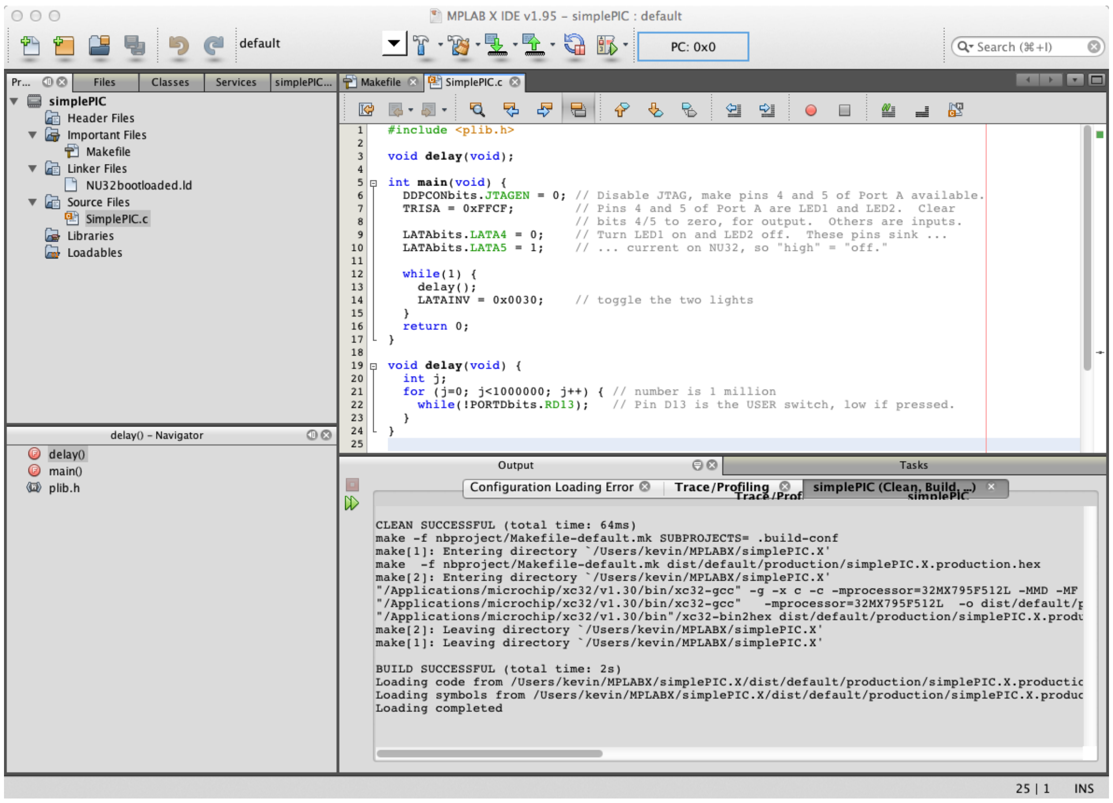
Step 8. Right-click on **Source Files** and choose **Add Existing Item...** Then choose `simplePIC.c`. Right-click on **Linker Files** and choose **Add Existing Item...** Select `NU32bootloaded.ld`. If you ever want to get rid of a file from a project, just right-click on it and choose **Remove From Project**.



Step 9. If you expand the **Header Files**, **Important Files**, **Linker Files**, **Source Files**, **Libraries**, and **Loadables** categories, you should find them all empty except for `NU32bootloaded.ld` in **Linker Files**, `simplePIC.c` in **Source Files**, and `Makefile` in **Important Files**. `Makefile` is a file generated by the IDE that is used to build the project.

Double-click on `simplePIC.c` to open it in the IDE editor. You can use this editor to create and edit your C files. (If you edit a file, remember to choose **File** > **Save** before building your project!)

You are now ready to build your project. Click on the hammer and broom icon, or equivalently, choose **Run** > **Clean and Build Main Project**. (The "clean" part simply removes any previously generated files by the compiler. It is not strictly necessary, but it is occasionally useful and it doesn't cost anything.) You should then see output similar to the figure below, indicating that the project has built successfully.



```
#include <plib.h>

void delay(void);

int main(void) {
  DDPCONbits.JTAGEN = 0; // Disable JTAG, make pins 4 and 5 of Port A available.
  TRISA = 0xFFCF;        // Pins 4 and 5 of Port A are LED1 and LED2.  Clear
                         // bits 4/5 to zero, for output.  Others are inputs.
  LATAbits.LATA4 = 0;    // Turn LED1 on and LED2 off.  These pins sink ...
  LATAbits.LATA5 = 1;    // ... current on NU32, so "high" = "off."

  while(1) {
    delay();
    LATAINV = 0x0030;    // toggle the two lights
  }
  return 0;
}

void delay(void) {
  int j;
  for (j=0; j<1000000; j++) { // number is 1 million
    while(!PORTDbits.RD13);   // Pin D13 is the USER switch, low if pressed.
  }
}
```

```
CLEAN SUCCESSFUL (total time: 64ms)
make -f nbproject/Makefile-default.mk SUBPROJECTS= .build-conf
make[1]: Entering directory `/Users/kevin/MPLABX/simplePIC.X'
make  -f nbproject/Makefile-default.mk dist/default/production/simplePIC.X.production.hex
make[2]: Entering directory `/Users/kevin/MPLABX/simplePIC.X'
"/Applications/microchip/xc32/v1.30/bin/xc32-gcc" -g -x c -c -mprocessor=32MX795F512L -MMD -MF
"/Applications/microchip/xc32/v1.30/bin/xc32-gcc"   -mprocessor=32MX795F512L  -o dist/default/p
"/Applications/microchip/xc32/v1.30/bin"/xc32-bin2hex dist/default/production/simplePIC.X.produ
make[2]: Leaving directory `/Users/kevin/MPLABX/simplePIC.X'
make[1]: Leaving directory `/Users/kevin/MPLABX/simplePIC.X'

BUILD SUCCESSFUL (total time: 2s)
Loading code from /Users/kevin/MPLABX/simplePIC.X/dist/default/production/simplePIC.X.productic
Loading symbols from /Users/kevin/MPLABX/simplePIC.X/dist/default/production/simplePIC.X.produc
Loading completed
```

Step 10. Now go to the directory `MPLABX` to see what the IDE has done. It has created a bunch of subdirectories, and buried under one called `dist` you should find a file with a `.hex` extension. This is your final executable.

To load the executable, put the NU32 in program receive mode by pressing and releasing RESET while holding the USER button. To install the program, type

```
nu32utility PORTNAME filename.hex
```

where `nu32utility` includes its full path, `PORTNAME` is the port discovered above, and `filename.hex` includes the full path.

1.5 Building a Standalone Executable

If you have an external programmer like the ICD 3 or PICkit 3, you can program the NU32 without a bootloader (or, if there is a bootloader already loaded in the PIC32 flash memory, you can overwrite it). This is called a *standalone* program. For this project you need only one file: `simplePIC_standalone.c`.

Start with an unpowered NU32 board with no cables connected to it. Next, connect five of the six pins of the PICkit 3 to the NU32 as shown in Figure 1.4. The five holes closest to the triangle on the PICkit 3 have header pins inserted, and these header pins lean against the five plated holes of the NU32. The triangle next to the plated hole near the D4 pin is aligned with the triangle on the PICkit 3. Now plug the USB cable of the PICkit 3 into your host computer, powering the PICkit 3. Do not connect any other USB cables to the NU32. Plug the NU32 into the wall, turn the power switch on, and verify that the red power LED comes on.

Now follow the same steps as in Section 1.4.2, with the following exceptions. In **Step 3**, you should choose the Hardware Tool PICkit 3 (specifically, the serial number of the PICkit 3). In **Step 5**, call your project `simplePIC_standalone`. In **Step 6**, only copy the file `simplePIC_standalone.c` to your new `simplePIC_standalone.X` directory. In **Step 8**, add only one item to the project, the Source File `simplePIC_standalone.c`. You do not need a custom linker script, as the default linker script suffices for a standalone program. In **Step 10**, you use the IDE to load the program onto the PIC32, not the NU32

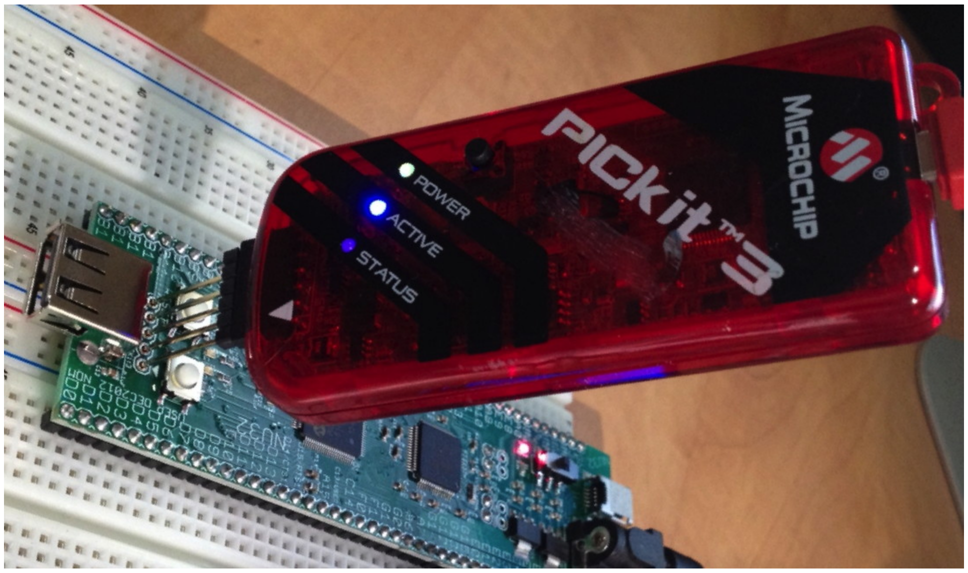
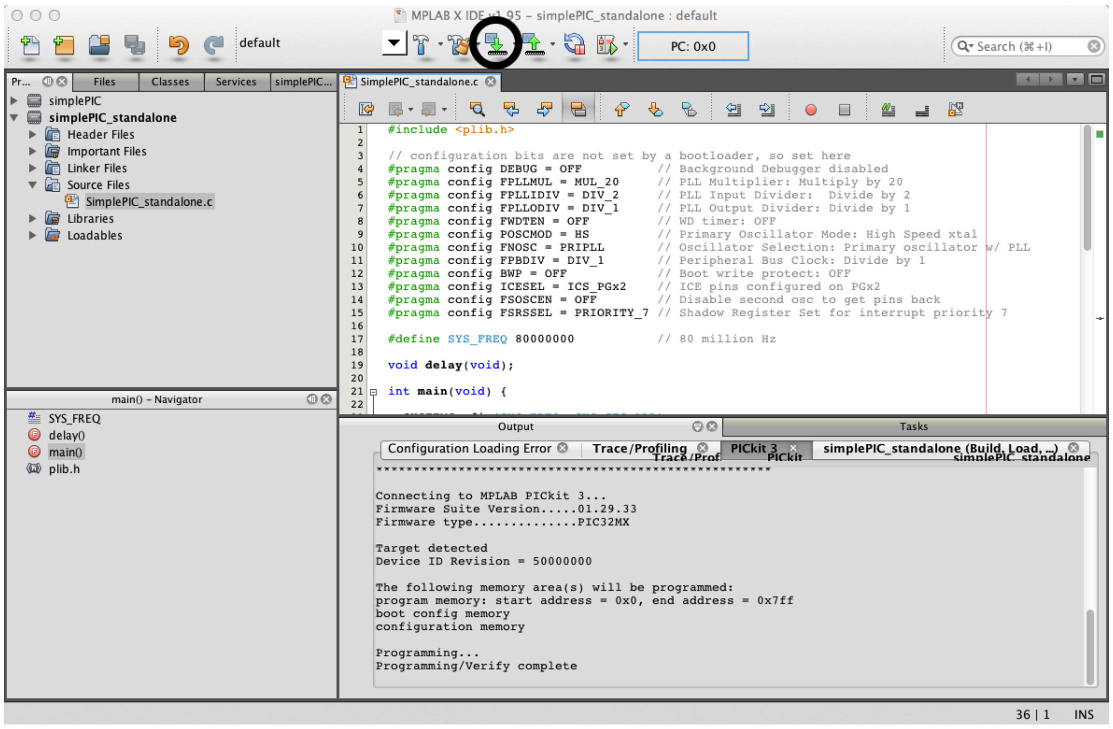


Figure 1.4: Attaching the PICkit 3 programmer to the NU32 board.

communication utility, which is only for bootloaded programs. To load the program from the IDE, click the **Make and Program Device Main Project** icon, circled in the next image. You should see something like the following indicating that the PIC32 has been successfully programmed:



```
#include <plib.h>

// configuration bits are not set by a bootloader, so set here
#pragma config DEBUG = OFF          // Background Debugger disabled
#pragma config FPLLMUL = MUL_20     // PLL Multiplier: Multiply by 20
#pragma config FPLLIDIV = DIV_2     // PLL Input Divider:  Divide by 2
#pragma config FPLLODIV = DIV_1     // PLL Output Divider: Divide by 1
#pragma config FWDTEN = OFF         // WD timer: OFF
#pragma config POSCMOD = HS         // Primary Oscillator Mode: High Speed xtal
#pragma config FNOSC = PRIPLL       // Oscillator Selection: Primary oscillator w/ PLL
#pragma config FPBDIV = DIV_1       // Peripheral Bus Clock: Divide by 1
#pragma config BWP = OFF            // Boot write protect: OFF
#pragma config ICESEL = ICS_PGx2    // ICE pins configured on PGx2
#pragma config FSOSCEN = OFF        // Disable second osc to get pins back
#pragma config FSRSSEL = PRIORITY_7 // Shadow Register Set for interrupt priority 7

#define SYS_FREQ 80000000           // 80 million Hz

void delay(void);

int main(void) {
```

```
****************************************************

Connecting to MPLAB PICkit 3...
Firmware Suite Version.....01.29.33
Firmware type..............PIC32MX

Target detected
Device ID Revision = 50000000

The following memory area(s) will be programmed:
program memory: start address = 0x0, end address = 0x7ff
boot config memory
configuration memory

Programming...
Programming/Verify complete
```

The LEDs should alternate on and off unless you press the USER button, which freezes them. This program will always run after any reset of the PIC32 until it is overwritten.

The listing of `simplePIC_standalone.c` is given below. The differences between the standalone version and the bootloaded version are discussed in Chapters 3 and 4.

Code Sample 1.2. `simplePIC_standalone.c`. Standalone version of bootloaded `simplePIC.c`.

```
#include <plib.h>

// configuration bits are not set by a bootloader, so set here
#pragma config DEBUG = OFF          // Background Debugger disabled
#pragma config FPLLMUL = MUL_20     // PLL Multiplier: Multiply by 20
#pragma config FPLLIDIV = DIV_2     // PLL Input Divider:  Divide by 2
#pragma config FPLLODIV = DIV_1     // PLL Output Divider: Divide by 1
#pragma config FWDTEN = OFF         // WD timer: OFF
#pragma config POSCMOD = HS         // Primary Oscillator Mode: High Speed xtal
#pragma config FNOSC = PRIPLL       // Oscillator Selection: Primary oscillator w/ PLL
#pragma config FPBDIV = DIV_1       // Peripheral Bus Clock: Divide by 1
#pragma config BWP = OFF            // Boot write protect: OFF
#pragma config ICESEL = ICS_PGx2    // ICE pins configured on PGx2
#pragma config FSOSCEN = OFF        // Disable second osc to get pins back
#pragma config FSRSSEL = PRIORITY_7 // Shadow Register Set for interrupt priority 7

#define SYS_FREQ 80000000           // 80 million Hz

void delay(void);

int main(void) {

  SYSTEMConfig(SYS_FREQ, SYS_CFG_ALL); // cache on, PBCLK setup, min flash wait
  DDPCONbits.JTAGEN = 0; // Disable JTAG, make pins 4 and 5 of Port A available.
  TRISA = 0xFFCF;        // Pins 4 and 5 of Port A are LED1 and LED2.  Clear
                         // bits 4/5 to zero, for output.  Others are inputs.
  LATAbits.LATA4 = 0;    // Turn LED1 on and LED2 off.  These pins sink ...
  LATAbits.LATA5 = 1;    // ... current on NU32, so "high" = "off."

  while(1) {
    delay();
    LATAINV = 0x0030;    // toggle the two lights
  }
  return 0;
}

void delay(void) {
  int j;
  for (j=0; j<1000000; j++) { // number is 1 million
    while(!PORTDbits.RD13);   // Pin D13 is the USER switch, low if pressed.
  }
}
```

1.6 A Second Program: Communicating with the Host

You are now ready for a second program, `talkingPIC.c`, that provides keyboard input to the NU32 and prints output to the host computer screen. These capabilities are quite useful for user interaction and debugging. To gain access to these capabilities, we will compile `talkingPIC.c` with the NU32 library, `NU32.c` and `NU32.h`. This library will be discussed in Chapter 4.

1.6.1 Bootloaded Version

This project uses the source files `talkingPIC.c`, `NU32.c`, and `NU32.h`, as well as the custom linker script `NU32bootloaded.ld`. If you are using the MPLAB X IDE, create a project `talkingPIC` and add `talkingPIC.c` and `NU32.c` as Source Files, `NU32.h` as a Header File, and `NU32bootloaded.ld` as a Linker File. Build and load the executable as in Section 1.4.2. If you are building and loading at the command line, put the four files in the same directory, along with the appropriate `makefile`, and type `make write` as in Section 1.4.1.

Once the program is running on the PIC32, to talk to the PIC32 you must connect through any RS-232 terminal emulator configured to communicate at 230,400 baud (bits per second), eight data bits, one stop bit, no parity, and hardware flow control (CTS/Clear To Send and RTS/Request To Send). On Windows, the COM port for communication is one less than the COM port number used by the bootloader communication utility. On a Mac, the port name ends in A instead of B.

Once connected, text you type will be sent to the PIC32 and then echoed back to you. For example, the PIC32 prompts you with the question

```
Hello?  What do you want to tell me?
```

and if you respond `Nothing!`, the PIC32 will receive it, write it back to the terminal emulator, and ask you again for input.

Examples of terminal emulators include CoolTerm or PuTTY (on Windows), or on Mac OSX or Linux you can use the simple `screen` command from the command line (e.g., the Terminal application on the Mac). For example, the command

```
screen /dev/tty.usbserial-00001024A 230400
```

connects to the COM port named `tty.usbserial-00001024A` at 230,400 bits per second. Eight data bits, one stop bit, and no parity are assumed, or you can be explicit using

```
screen /dev/tty.usbserial-00001024A 230400,cs8,-cstopb,-parenb
```

Once connected using `screen`, the PIC32 will write to the terminal and accept keyboard input. Use ctrl-a k to quit.

The `talkingPIC.c` program listing is given below.

Code Sample 1.3. `talkingPIC.c`. The PIC32 echoes any messages sent to it from the host keyboard back to the host screen.

```
//#define NU32_STANDALONE  // uncomment if program is standalone, not bootloaded
#include "NU32.h"          // plib.h, config bits, constants, funcs for startup and UART
#define MAX_MESSAGE_LENGTH 200

int main(void) {
  char message[MAX_MESSAGE_LENGTH];

  NU32_Startup(); // cache on, min flash wait, interrupts on, LED/button init, UART init
  sprintf(message, "Hello World!\r\n");
  NU32_WriteUART1(message);
  while (1) {
    sprintf(message, "Hello?  What do you want to tell me?  ");
    NU32_WriteUART1(message);
    NU32_ReadUART1(message, MAX_MESSAGE_LENGTH);
    NU32_WriteUART1(message);
    NU32_WriteUART1("\r\n");
    NU32LED1 = !NU32LED1; // toggle the LEDs
    NU32LED2 = !NU32LED2;
  }
  return 0;
}
```

1.6.2 Standalone Version

The standalone version of `talkingPIC.c` requires the same files as the bootloaded version in Section 1.6.1, with the exception that the custom `NU32bootloaded.ld` linker script should not be used, since the default linker script is appropriate for standalone programs. Also, you must uncomment the first line of `talkingPIC.c`, so that it now reads

```
#define NU32_STANDALONE  // uncomment if program is standalone, not bootloaded
```

This tells `NU32.h` that the program is being compiled as a standalone version.

Create a project `talkingPIC_standalone` in the MPLAB X IDE with `NU32.c` and `talkingPIC.c` as source files and `NU32.h` as a header file. Build and load the executable as in Section 1.5. Once the program is running, to talk to the PIC32 you must connect through an RS-232 terminal emulator. Once connected, the NU32 echoes whatever you type back to your own computer screen, demonstrating two-way communication.

1.7 Chapter Summary

- A microcontroller, like the PIC32, differs from a microprocessor in that it has peripherals for sensing, communication, and control.
- A PIC32 can be programmed using a separate programming device (standalone program) or by using a bootloader. A bootloader is a program installed in the PIC32 flash memory that can communicate with a host computer (via a USB cable, for example), accept a new user program, and write it to program flash. Typically the bootloader knows to enter this "write new program" mode if a button is being pressed when the PIC32 resets. Otherwise, the bootloader simply calls a program that has already been installed in memory.
- To program a PIC32, you need a PIC32 development board or starter kit and a host computer with Microchip's XC32 compiler and MPLAB X IDE. To use the NU32 development board, you also need the FTDI virtual COM port driver, which allows your USB ports to talk to the PIC32 over a USB cable. If you are bootloading your programs, you also need the bootloader communication utility.
- PIC32 executables can be created at the command line or in the MPLAB X IDE.
- Loading an executable on a PIC32 can be done with a bootloader communication utility (if the PIC32 has a bootloader program) or using an external programmer controlled by the MPLAB X IDE.
- Bootloaded programs for the NU32 use the custom `NU32bootloaded.ld` linker script, while the default linker script suffices for standalone projects.

1.8 Exercises

1. Create the executables for `simplePIC.c` (or `simplePIC_standalone.c`) and `talkingPIC.c`, load them, and verify that they work. Try modifying them and verify that you see the expected behavior.