

Lab 1 – Installing and Using the Arduino IDE for chipKIT Development

Attribution

This lab was developed by Jacob Christ with help from former Mt.SAC students the chipKIT and Arduino community. Notable help from Brian Dobrovodsky (content), John Tsai (content), Michael Skoczen (content and testing), Brian Schmalz (installing drivers), Rick Anderson (review and installing chipKIT-core) and Mark Christensen (testing).

Equipment Needed

Windows XP (or better) Computer or MACOSX (Lion or better) Computer

Arduino Uno

chipKIT development boards (the more you have the better off you are)

USB Cables (type depends on the boards you have)

Part 1 - Arduino IDE

The Arduino IDE is a software tool that is used to develop C and C++ programs for the Arduino and chipKIT development boards we will be using. IDE is an acronym for Integrated Development Environment. The acronym stems from the traditional combination of several programs being required for embedded development being *integrated* into a single environment. Among other things, the Arduino IDE is a combination of the following technologies:

Text Editor: used for editing of text files, also known as the source code.

Compiler: used to change source code into object code.

Linker: used to convert one or more object code files into a single machine code file.

Programmer: used to transfer the machine code file from our development computer to our target board.

Serial or Plotting Monitor: used to interact with our target board to test our program.

This set of tools is used in an iterative process to develop programs. This process is shown in the side bar and mirrors the order the tools are presented above.

A screenshot of the Arduino IDE interface. The window title is "Blink | Arduino 1.6.7". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". The toolbar shows icons for file operations and execution. The main text area displays the "Blink" sketch code, which includes comments and C++ code for setting up and looping through digital pin 13 to turn an LED on and off. The code is as follows:

```
/*
 * Blink
 * Turns on an LED on for one second, then off for one second, repeatedly.
 *
 * Most Arduinos have an on-board LED you can control. On the Uno and
 * Leonardo, it is attached to digital pin 13. If you're unsure what
 * pin the on-board LED is connected to on your Arduino model, check
 * the documentation at http://www.arduino.cc
 *
 * This example code is in the public domain.
 *
 * modified 8 May 2014
 * by Scott Fitzgerald
 */

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);           // wait for a second
  digitalWrite(13, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);           // wait for a second
}
```

The Development Process

(as opposed to the creative process)

1. Enter source code in a text editor.
2. Compile source code into object code.
3. Link object code together into executable machine code.
4. Upload machine code to the target platform.
5. Test machine code in target application.

Windows

The Arduino IDE is compiled to run on a Windows PC, MAC OSX and Linux (as well as possible others). This section of the lab is written explicitly for Windows, since these are the computers available in the computer lab at the school I teach at. This lab was written using the Windows 10 operating system, and the computers in the classroom typically have Windows XP. There is a slight difference that will be addressed in class.

For instructions on installing Arduino IDE on a Mac computer visit the following site:

<https://www.arduino.cc/en/Guide/MacOSX>

For instructions on installing Arduino IDE Linux computer visit the following site:

<http://playground.arduino.cc/Learning/Linux>

Where to find the Arduino IDE

These labs were tested using versions 1.6.7 and 1.84 of the Arduino IDE but they should work with any version of the IDE that works with chipKIT-core. Download the zip file from this page by clicking on the link text “Windows ZIP file for non admin install” found on the following page:

<https://www.arduino.cc/en/Main/Software>



ARDUINO 1.8.5
The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board. Refer to the [Getting Started](#) page for Installation instructions.

- Windows** Installer
- Windows** ZIP file for non admin install
- Windows app** Requires Win 8.1 or 10
- Mac OS X** 10.7 Lion or newer
- Linux** 32 bits
- Linux** 64 bits
- Linux** ARM

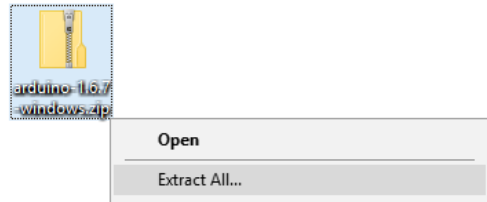
[Release Notes](#)
[Source Code](#)
[Checksums \(sha512\)](#)

The Arduino IDE is a fast changing open source tool and there are many versions to choose from. If version listed at the top of the page is not the version you are looking for then you should be able to find it on a link somewhere on this page that contains a list of previous released versions of the software.

Save the zip file to the desktop of your computer.

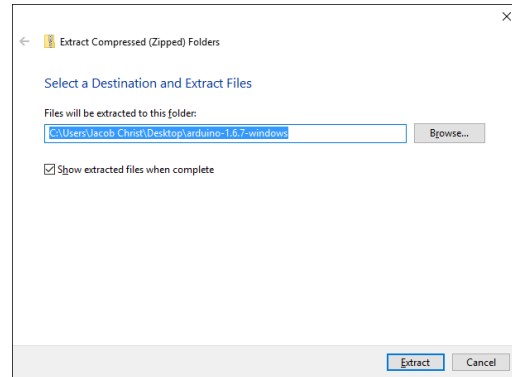
Unzipping the file in Windows

The file you have downloaded, regardless of the operating system, is a compressed file. In Windows this file can be extracted by right clicking on the downloaded file and picking the "Extract All..." item on the pop-up menu.



A new dialog will appear that asks for the location to extract the files too. This location should be your desktop.

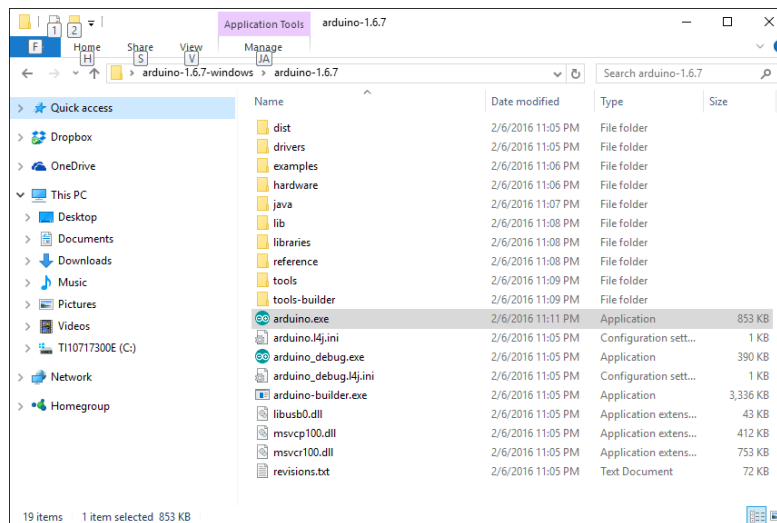
Click the "Extract" button to extract the Arduino IDE. This may take a few minutes due to the large size of the program. When complete this will result in a new folder on your desktop called "arduino-1.6.7-windows" which will contain the extracted development tools.



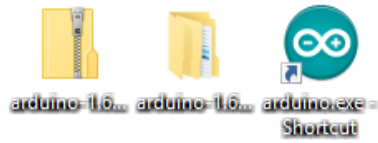
The Arduino IDE should work from any folder but for consistency in this manual it will be assumed that the Arduino IDE will be in a folder onto the Desktop.

Creating a Shortcut

You could easily run Arduino IDE by just double clicking on arduino.exe file in the unzipped folder, however for the class we will make a shortcut to the executable by right clicking on arduino.exe and selecting copy then right clicking on the desktop and pasting as a shortcut.

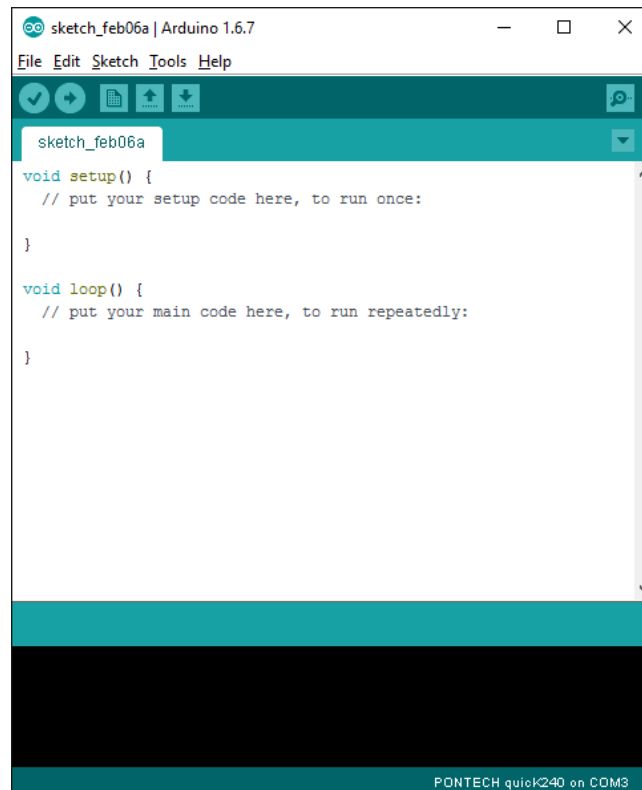


Your desktop will now look something like this:



How to run Arduino IDE

Double click on "arduino.exe - Shortcut" icon on your desktop and you should get a window that looks something like the one pictured below.



Check off

Part 1 of the lab is complete, call the instructor over to verify your installation.

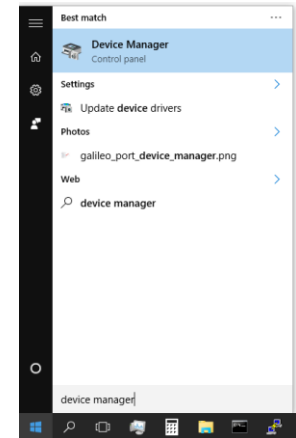
Part 2 Arduino Device Drivers and the Windows Device Manager

Plug in an Arduino Uno board into your PC and wait for Windows to begin the driver installation process. If you are on a Mac, Linux or Windows 10 the board should be found automatically. If you are using an older version of windows then after a few moments, the process will fail, despite its best efforts.

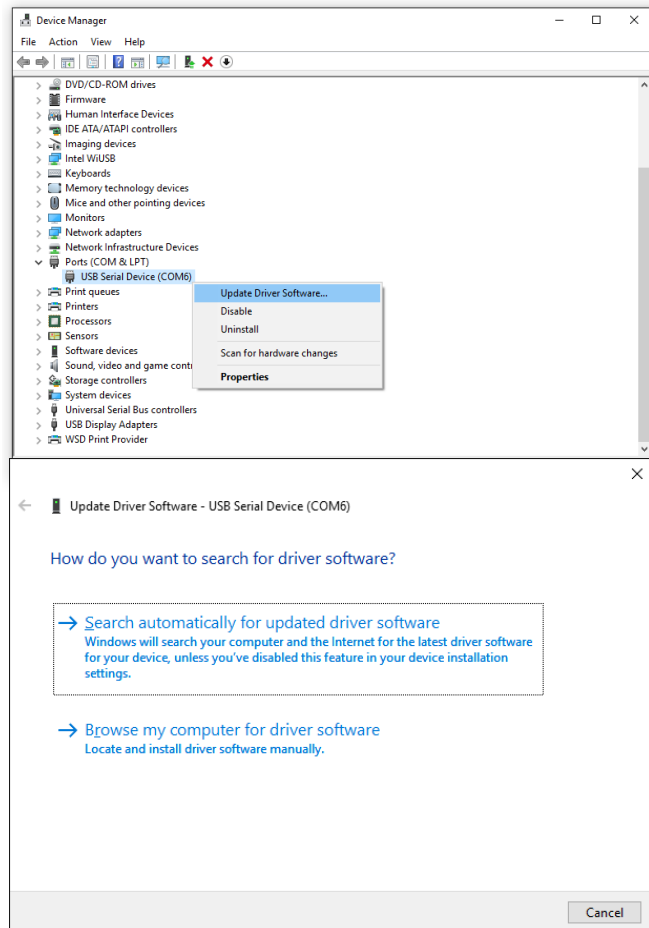
In Windows XP Click on the Start Menu, and open up the Control Panel. While in the Control Panel, navigate to System and Security. Next, click on System. Once the System window is up, open the Device Manager.

In Window 7, 8 you need only search for the "Device Manager"

In the Device Manager Look under Ports (COM & LPT). You should see an open port named "Arduino UNO (COMxx)" or USB Serial Device (COMxx). If there is no COM & LPT section, look under "Other Devices" for "Unknown Device". You may have to plug and unplug the board a couple of time to figure out which entry in the device manager corresponds to the board you have. You can tell which entry is yours because it will disappear when you unplug your board.

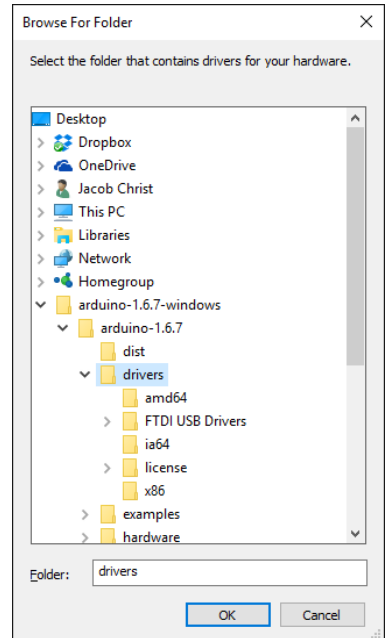
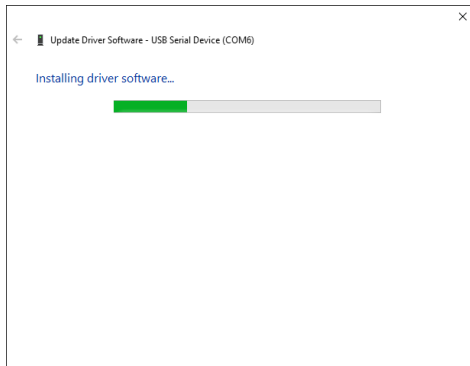


Right click on the "Arduino UNO (COMxx)" port and choose the "Update Driver Software" option.

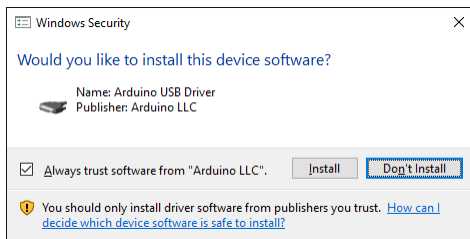


Next, choose the "Browse my computer for Driver software" option.

Navigate to and select the “drivers” folder in the Arduino IDE install folder on your desktop. Once selected, click okay and windows will search for the driver to install.

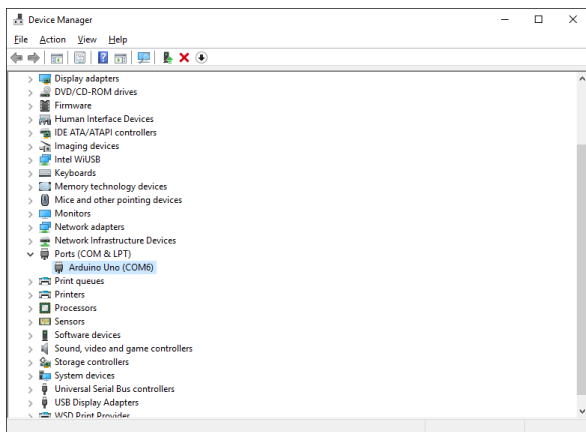
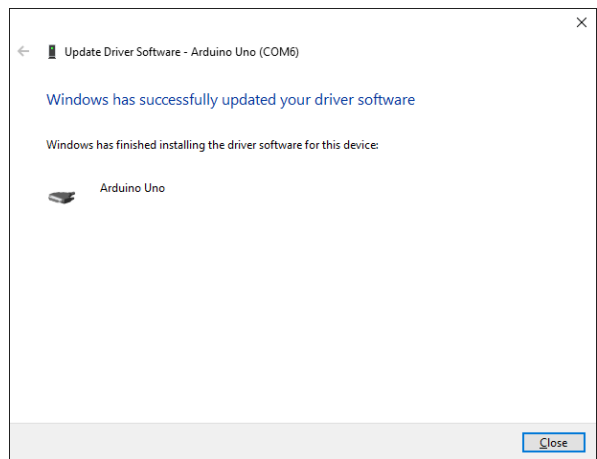


A Windows Security message may show up like is shown below. Click install to install the driver.



Once the driver has been installed you will get a dialog box as shown to the right.

The Device Manager entry will now update and show a new device under the Ports (COM & LPT) hierarchy for the Arduino Uno on (COMxx).



Finally, make note of the assigned COM port number. This will be useful later when trying to upload programs to our board.

Check off

Part 2 of the lab is complete, call the instructor over to verify your installation.

Part 3 Get Blink Sketch to run on Arduino Uno

The Arduino IDE

If not already running, double click on "arduino.exe - Shortcut" icon on your desktop and you should get a window that looks something like the one pictured to the right. Some notable items on the IDE window are pointed out below.

Menus

Icon Bar short cuts

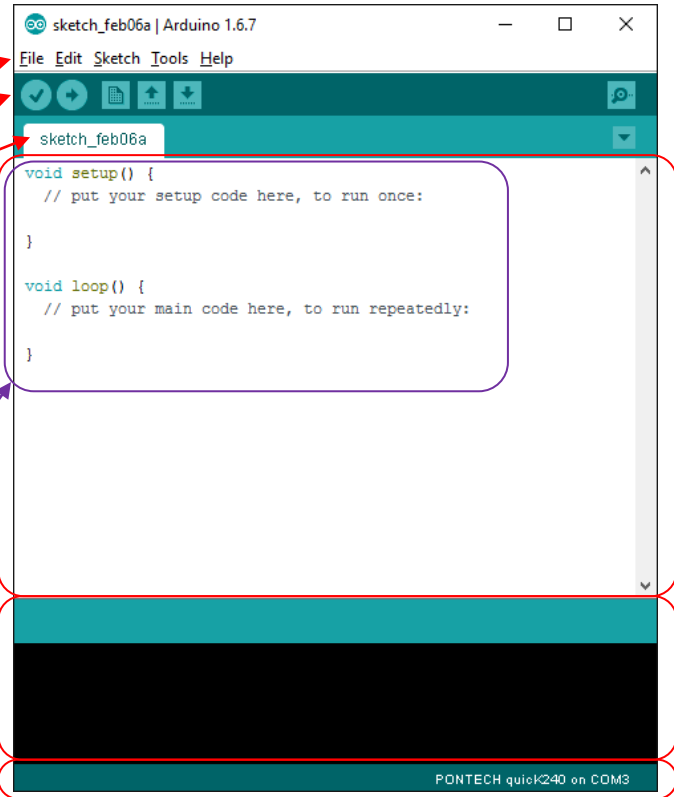
Filename

Text editor where we edit our program

Code (and comments) rendered in colorful text

Compiler and Program Transfer Status

Selected target board and COM port Status



The Icon Bar

The most frequently used buttons when you are trying to get a program working are on the Icon Bar. A brief description of each button is given here:



Verify: compiles your code. Any errors are shown in the black status box at the bottom of the IDE.



Upload: Compiles your code then attempts to write the compiled machine code to a connected board.



New: Creates a new sketch.



Open: Opens an existing sketch.



Save: Saves your work.



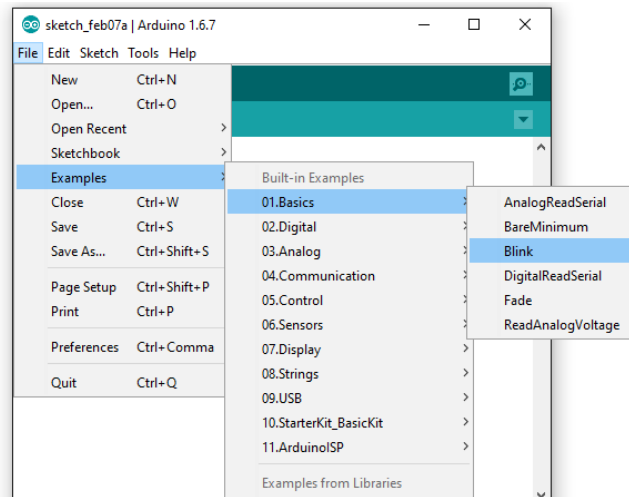
Serial Monitor: Open a serial terminal to interact with your running program.

Sketches

All programs written using the Arduino IDE are called sketches. The idea behind the sketch is that Arduino IDE makes writing a programming as simple as an artist picking up a paper and pencil to draw a sketch and hence the name.

Example Sketches

One way to learn how to sketch is by looking at example sketches created by others. The Arduino IDE comes with many example sketches to help you understand how to write programs for Arduino and Wiring compatible boards such as chipKIT. These sketches are accessible from the File menu by mousing over Examples. As you can see from the image to the right in this installation there are many examples to choose from. We will start at the beginning and try a "01.Basic" sketch called "Blink". Mouse to and click on the "Blink" sketch to open the example file.



Blink Sketch

When you select the "Blink" example a new IDE window will open leaving your unused sketch behind the new example sketch. Notice that you can resize the window so that you can see the whole sketch on the screen in a single glance.

Two things should look different in this new window.

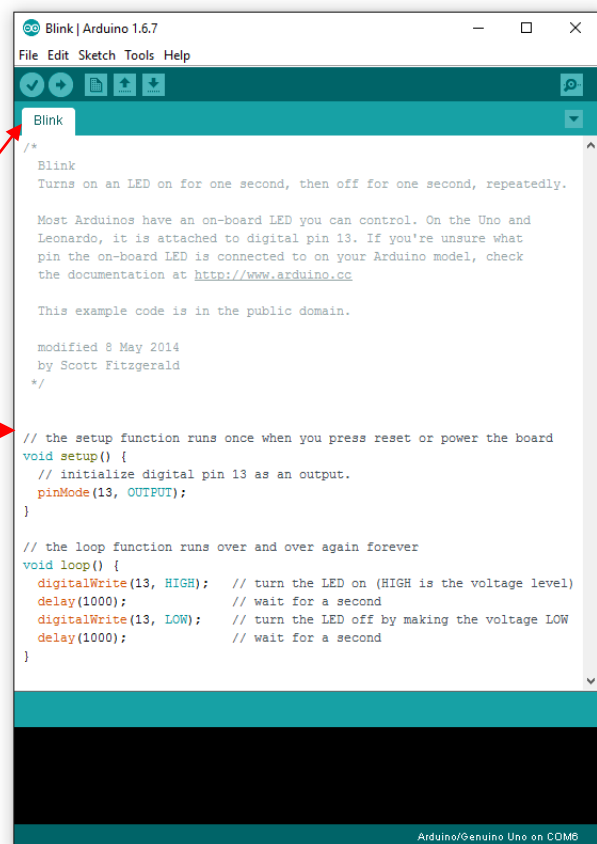
The tab will be named "Blink"

The text shown in the window is now filled with the code for this loaded sketch.

Notice also that the IDE editor has color coded the "source code" to make it easy to distinguish the parts of the code. The color code is listed roughly below.

Gray: User comments, orange: function calls, olive: function definitions, blue: qualifiers and constants.

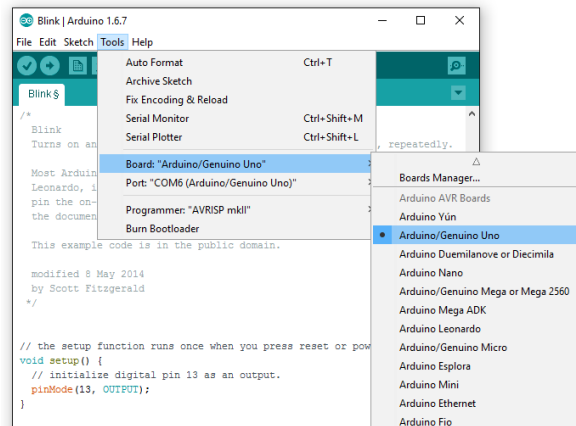
The "out of the box" functionality of the "Blink"



sketch is to blink a LED connected to the board at a rate of 1/2Hz (one second on then one second off).

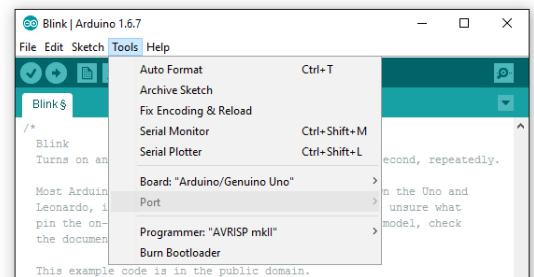
Selecting the Board

Now that we have a useful program loaded into the IDE we must next select the board we wish to run our program on. This is done from the Tools->Board menu. In this lab we will be trying to run the "Blink" sketch on several different boards Arduino and chipKIT boards. But for now we can only test on the Arduino Uno since we have not yet loaded the chipKIT core into the IDE. The board we will be testing on is the Arduino Uno. So select the "Arduino/Genuino Uno"

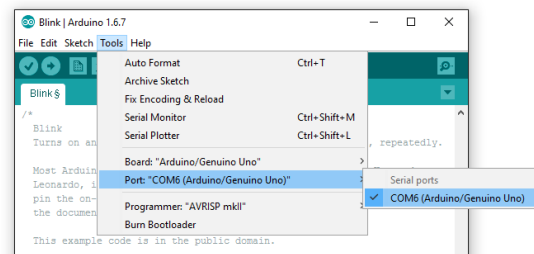


Selecting the COM Port

After selecting the target board, we must next select the COM Port the board is mapped to on our computer. This is done through the Tools->Port menu. Sometimes the board or COM port we are looking for is not present in the list of ports. When this happens the "Ports" menu item will be grayed out as shown to the right. This can happen when the driver is not installed correctly or the board is not plugged in to our computer.



If the COM Port for our board is present, select it so that we can test our program.

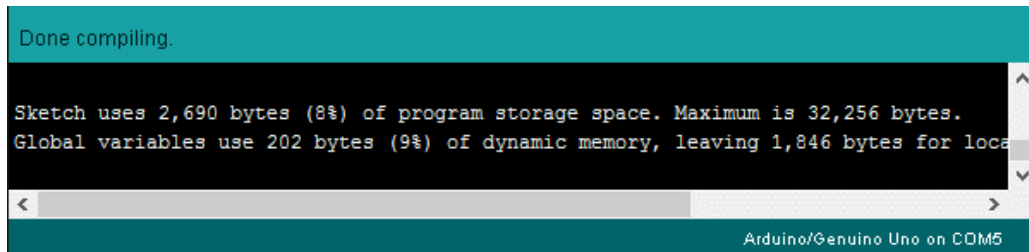


Compiling the Sketch

Before we actually try to program our board, we need to verify that our program is syntactically correct. As mentioned previously the "Verify" button will compile the program. When you press it the bottom of the Arduino IDE interface will indicate that the program is compiling by displaying "Compiling sketch..." and showing a progress bar.



When finished compiling the stats will change to "Done compiling." and if everything was syntactically correct the size of the sketch will be displayed.



```
Done compiling.  
Sketch uses 2,690 bytes (8%) of program storage space. Maximum is 32,256 bytes.  
Global variables use 202 bytes (9%) of dynamic memory, leaving 1,846 bytes for local variables.  
Arduino/Genuino Uno on COM5
```

Upload and Test



Next press the upload icon button and observe the LED blinking on the board.

Check off

Part 3 of the lab is complete, call the instructor over and demonstrate uploading the program to the board.

Part 4 Installing chipKIT-core into Arduino IDE and chipKIT drivers

Arduino IDE as a Multiplatform Tool

The Arduino IDE, as of version 1.6.x, is a multiplatform Arduino Compatible Integrated Development Environment. "Multiplatform" implies that it works with more than one platform. A platform in the embedded development world usually implies a family of microprocessors chips that have a common microprocessor architecture. You can think of different platforms kind of like different construction toys. For example, Lego bricks and Mega Bloks bricks. Both can be used to construct sculptures but they may not be interchangeable. That is to say they are different platforms for construction. In the case of embedded development, the tools used to convert your program source code to machine code are usually for a specific platform or architecture of microprocessor.

chipKIT-core

Platform tools for the Arduino IDE are implemented in what are called cores. The Arduino IDE comes with the 8-bit Atmel AVR core pre-installed. This core works with the chips on the original Arduino Uno and related boards. In other words, the 8-bit Atmel AVR platform. Besides the Arduino 8-bit Atmel AVR core there is also an Arduino 32-bit ARM core that does not come pre-installed as well as many others. The core we are going to be using is called the chipKIT-core and it targets the 32-bit Microchip PIC32 microcontrollers (which are based on the MIPS 4000 microprocessor). Besides the listed cores in the IDE and the chipKIT-core there are cores made by others and probably more on the way. The reason the cores for all these different platforms are not included by default is that they are quite large. If they had been included, it would mean you would need extra hard drive storage and more importantly extra time to download cores you are not planning on using. What a waste!

The Arduino IDE with the chipKIT-core installed utilizes the gcc open source C++ cross compilers. A cross compiler is a tool that lets you generate machine code for one computer architecture on another. The chipKIT-core cross compiler has similar functionality that you would find in a C++ compiler for writing applications for a PC but utilizes libraries specifically written for chipKIT embedded hardware. Programming is done utilizing the C++ language but automatically included libraries allow a beginner to ignore (at least in the beginning) the complexity associated with C++ so that they can dive in and get started creating fast.

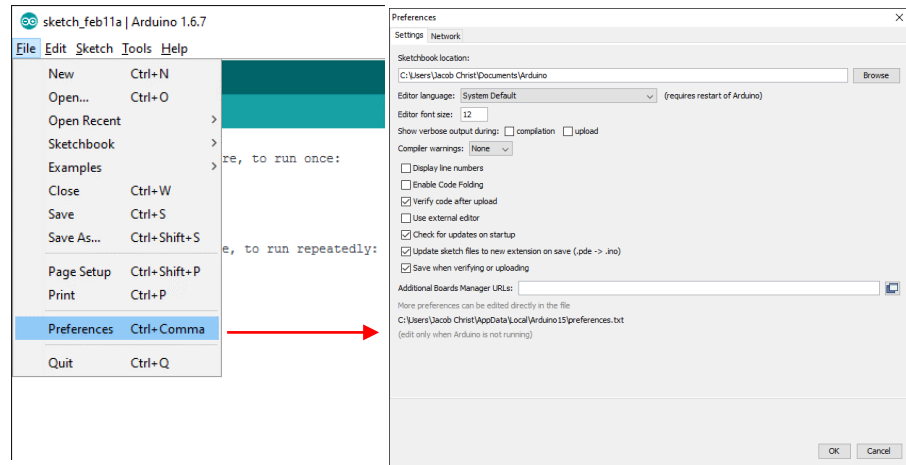
To use an additional core the Arduino team created a mechanism within the IDE to allow you to simply install a core by copying and pasting a URL in the IDE and then go to a "Board Manager" to download selected versions of additional cores.

Installing chipKIT-core

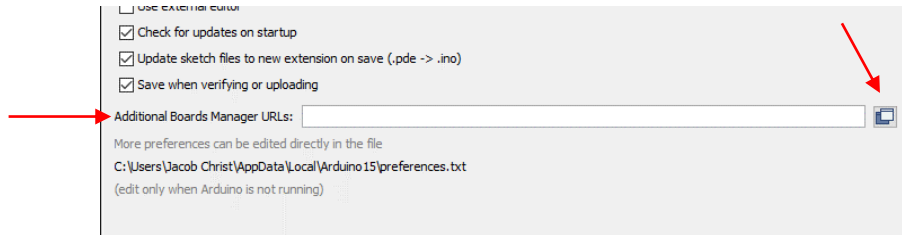
From within the Arduino IDE, click on the following menu items:

File->Preferences (for Windows) or Arduino->Preferences (for a MAC)

This will open the preferences dialog box.

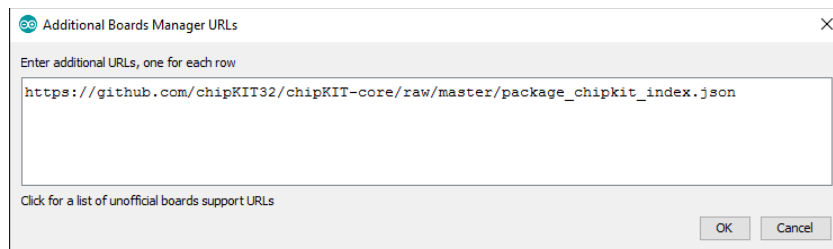


Within the preferences dialog box look for the text entry field called "Additional Boards Manager URLs:". Click the icon to the right of the text field to open a new dialog box to allow you to edit all additional board manager URLs.



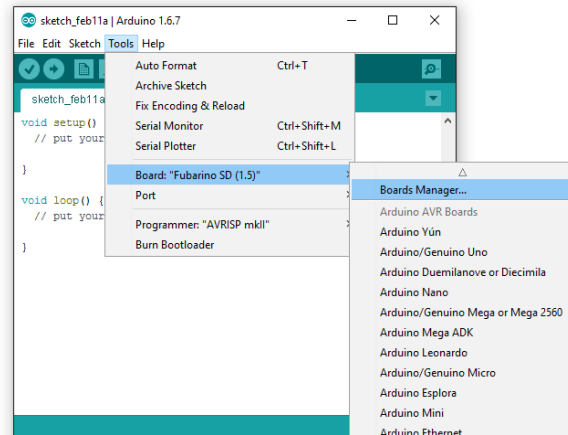
Each URL in the "Additional Boards Manager URLs" dialog must be on a line by itself. With this dialog box open copy and paste onto a blank line the following URL to enable downloading of the chipKIT-core:

https://github.com/chipKIT32/chipKIT-core/raw/master/package_chipkit_index.json

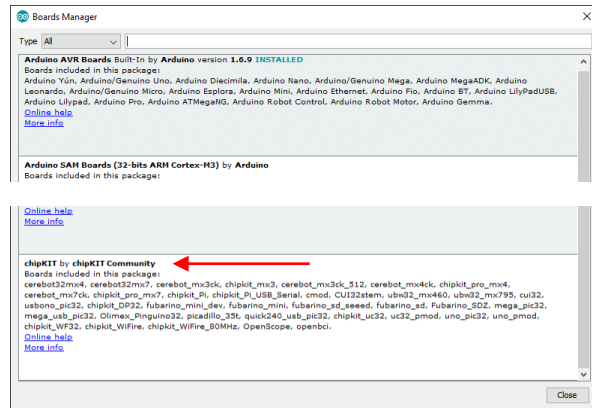


The Arduino IDE lets you have many different cores loaded into the IDE as long as each URL is on a separate line. Click OK to close the Additional Boards Manager URLs dialog box and then click OK again to close the Preferences dialog box.

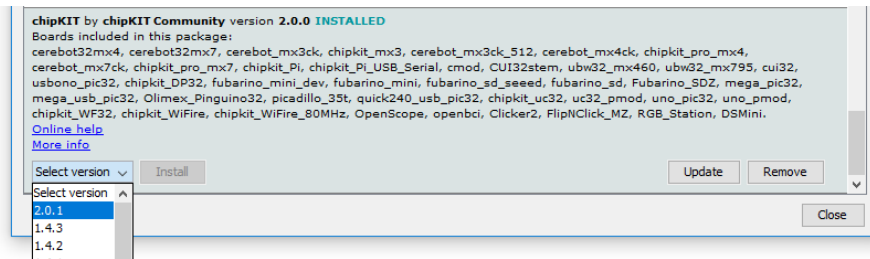
Now select the Tools->Board->Board Manager menu from the Arduino IDE, and it will open up the Boards Manager window. Once open you will see several other cores that are not installed by default. It may take a few seconds to retrieve the chipKIT information from the internet, but eventually the chipKIT-core will show up and be presented in the list of cores.



Scroll down until you see the "chipKIT by chipKIT Community" board listing. Click once on any of the text in this section. Once you click a dropdown menu and an Install button will appear.



Select version 2.0.1 then press the Install button. It will take some time to download all of the chipKIT components and install them, but when it's done, you can click the Close button to close the Board Manager window.



Once complete verify the chipKIT-core is installed by looking in the boards menu for the chipKIT boards. Do this from the Tools->Board menu and scroll down until you see the chipKIT boards.

As new versions of the chipKIT-core files are released, you will be able to update your chipKIT-core files from inside the Arduino IDE. During this class, refrain from upgrading to be assure that all your labs work.

This section of the lab manual was largely based on documentation on the chipKIT wiki. In addition to this method there are also other ways to install a chipKIT core which are documented here:

http://chipkit.net/wiki/index.php?title=ChipKIT_core

USB Drivers for chipKIT Boards

A device driver, or often simply referred to as a driver, is a piece of software that is either pre-installed or installed after the fact in an operating system to allow the operating system to interact with a piece of hardware connected to the computer. The driver needed depends on the piece of hardware you are connecting to your computer. Most, and maybe all, of the chipKIT boards need one or both of the following drivers. An FTDI (Future Technology Devices International, ftdichip.com) VCP (Virtual COM Port) driver or a chipKIT USB CDC-ACM driver. As of this writing the latest versions of Windows (10), MAC OSX and Linux have these drivers built into the operating system. If you are using an older version of Windows, you may need to install these drivers yourself.

FTDI

FTDI VCP drivers are needed for boards that have a very popular USB to Serial converter chip made by FTDI. Some examples of these boards are the Uno32, the Max32. The FTDI VCP driver can be downloaded from their web page:

<http://www.ftdichip.com/Drivers/VCP.htm>

Look for the operating system you are using to download the latest version of the driver. In windows you will download a zip file. Save this file to your desktop and right click on it to extract the files.

Run the executable and the driver will install within a few seconds.

chipKIT

Later chipKIT boards used the built in USB interface that is available on some PIC32 microcontrollers. Programming in the bootloader of the chip allows the USB port connected to these chips to emulate a virtual serial port, often referred to as a CDC-ACM USB device. These drivers are included in the chipKIT-core but can be difficult to find on your hard drive. Alternatively, the drivers located in a zip file can be downloaded from the following location:

https://github.com/chipKIT32/chipKIT-drivers/releases/download/v1.0/drivers_windows_v1.zip

Like the FTDI driver above, save the zip file to your desktop, extract the files and run the executable. Follow the instructions given by the executable to install the drivers.

Check off

For the check off of this section of the lab, demonstrate that the chipKIT core is installed within the Arduino IDE and that the operating system can connect to a chipKIT board. The easiest way to demonstrate connection is to plug in and unplug your board and verify that the COM port for the board is added then removed from the Tools->Port menu in Arduino IDE.

Part 4 of the lab is complete, call the instructor over and demonstrate.

Part 5-8 - Get a Blink sketch working on 4 different chipKIT boards

The Blink sketch for the Arduino Uno specifically blinks the LED connected to Pin 13. The developers of chipKIT realized that not all boards would have LED's connected to the same pin. Because of this they added a macro to the board definition files that automatically will select the correct pin to use for the LED on the selected chipKIT board. Open the File->Examples->01.Basics->Blink sketch.

The LED_BUILTIN Macro

On the lines of the sketch with the `pinMode()` function and the `pinWrite()` function the macro `LED_BUILTIN` will change the depending on the board you tell the IDE is connected to the computer.

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}
```

The `LED_BUILTIN` macro is the abstracted pin number on the target board that we want to blink. Most target chipKIT boards have these macros defined. Depending on the board you are using you may need to replace `LED_BUILTIN` with the correct pin number to get the desired LED to blink. For the Fubarino SD, `LED_BUILTIN` does not need to be changed. In the chart below you will see a list of boards used in this class and the pins on the boards that have LED's connected to them.

Board	LED1	LED2
Fubarino SD	21	N/A
Uno32	13	
Max32	13	
uC32	13	
UAV100	80	83
Quick-240	37	81
Uno	13	

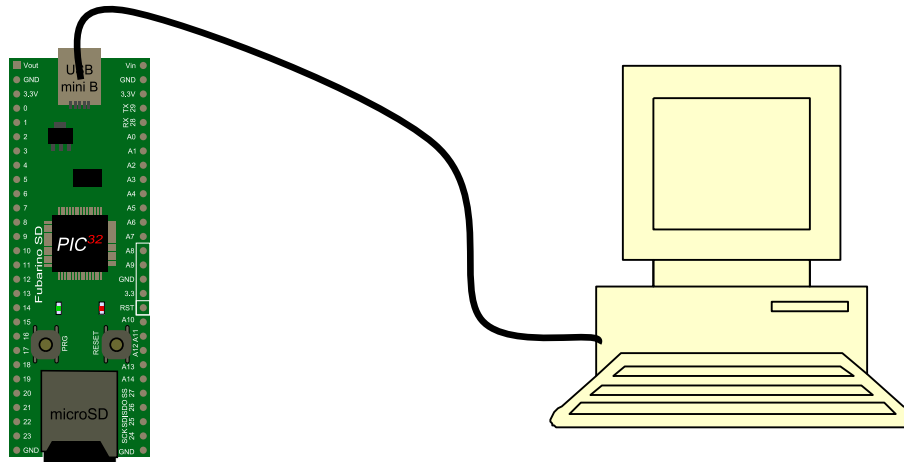
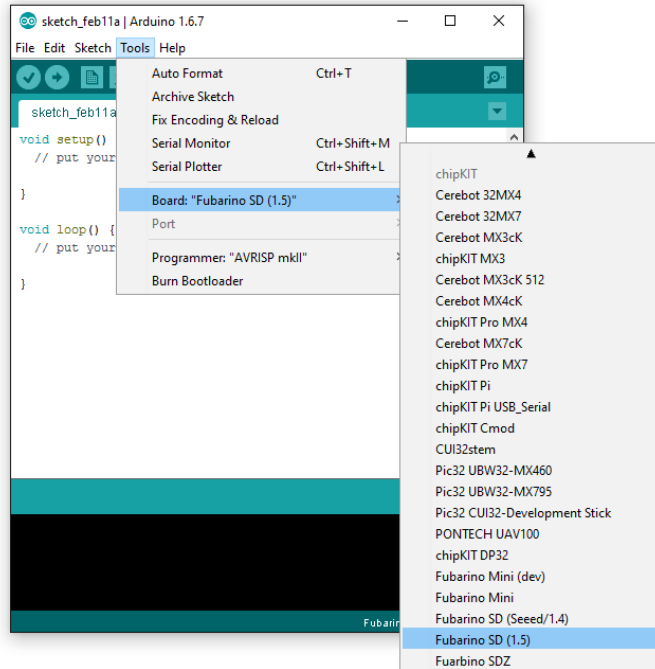
Here's something to try, use a value other than what is expected and see if the LED blinks?

Selecting the board

With the new chipKIT-core installed you can now select a chipKIT board to run our program on. This is done from the "Tools->Board" menu under the chipKIT section.

Connecting the board to the computer

Most chipKIT and Arduino boards will connect to a computer with a USB cable. Additionally, most boards will also be powered by the computer stealing power from the USB port. USB cables have four connections: +5VDC, GND (ground), DATA+, and DATA-. Newer microcontrollers run on +3.3V or lower and boards such as the Fubarino SD, chipKIT Uno32 and PONTECH Quick-240 will have voltage regulators on board that lower the +5VDC from the USB cable to a usable +3.3VDC for the microcontroller.

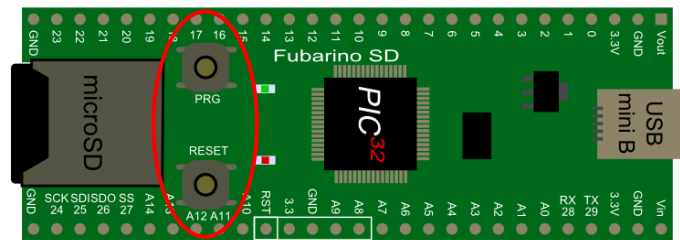


Getting to the Bootloader

A bootloader is a small program that is pre-loaded on to each board by the manufacturer (or yourself if you happen to be the manufacturer) that allows the board to communicate with another computer in order to receive additional programs.

Before you can program the board, it needs to be ready to receive the data and "sitting in the bootloader" the Uno32 and Arduino Uno do this automatically but the others need more steps.

Fubarino SD: The Fubarino SD has two buttons on the top of the board labeled PRG and RESET. The Reset button does just that, it will reset the microcontroller. The PRG button, if held down during a reset will cause the board to enter the bootloader. When in the bootloader the GREEN led will blink rapidly.



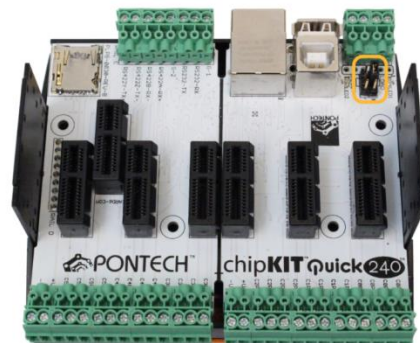
A video showing the procedure to get the Fubarino SD into the bootloader can be found here:

<http://www.youtube.com/watch?v=7Knri2QWEFU>

PONTECH UAV100: Put the jumper in the programming position, then push the button.



PONTECH Quick-240: Similar to the UAV except rotated and you short the two pins next to the jumper instead of using a button



The Arduino Uno, chipKIT Uno32, chipKIT Max32 and chipKIT uC32 have a chip that can control the reset line of the microcontroller allowing the Arduino IDE to force the device into reset prior to programming. Because of this no special reset sequence is needed to program these boards.

Selecting the Serial Port

Each chipKIT or Arduino board connected to your computer may have different COM ports associated with it. This is especially true if you have more than one board connected at the same time. Follow the instructions in part 3 of this lab to select the a serial (COM) port to connect to your board. This is due to the COM port number being chosen by the computer. If there is only a single device that has a COM port on the computer then finding the correct COM port is easy, for it will be the only one in the list. However, if there is more than one COM device connected to the computer you will need to figure out which COM port is for the device you want to program. This is simply done by looking at the list of available COM ports prior to connecting your board and then again after connecting and noting the new COM port that was added after your board was plugged in. If no additional COM port is found after plugging your board this could be due to one of several issues.

- The driver for the board you are using is not installed.
- The device you are using is not in the bootloader.
- The USB Cable is damaged.
- The board itself is damaged.

Check off

Demonstrate uploading the blink sketch to four different chipKIT boards. At least one must be a Uno32 or Max32 and at least one must be the Fubarino SD.

Additional information for the boards used in this lab can be found on these web pages

Fubarino SD

<http://fubarino.org>

Digilent Uno32, uC32 and Max32

<http://www.digilentinc.com>

PONTECH UAV100 and Quick-240

<http://www.pontech.com>

<http://www.quick240.com>

Part 9 - Anatomy of a Sketch and User Sketches

Anatomy of a Sketch

Let's examine each parts of the Blink sketch in detail. The first six lines of gray/green text in the sketch are encapsulated with the delimiters `/*` and `*/`.

```
/*  
  Blink  
  Turns on an LED on for one second, then off for one second, repeatedly.  
  
  This example code is in the public domain.  
*/
```

These delimiters and everything between them are called a multiline comment. Comments in general are notes that the programmer leaves for themselves to remind them what the program does or specifics that need to be adhered to when altering the program. Comments have no affect on how the program runs or what it does.

Next is the setup function.

```
// the setup function runs once when you press reset or power the board  
void setup() {  
  // initialize digital pin LED_BUILTIN as an output.  
  pinMode(LED_BUILTIN, OUTPUT);  
}
```

The setup function is five lines long. The first and last of the function are what define and encapsulate the function. The function consists of four important aspects.

1. The return type. In this case **void** which simply means this function is void of a return type.
2. The function name. In this case: **setup**.
3. The parameter list, a comma separated list encapsulated by parenthesis after the function name. In this case an empty list.
4. The function encapsulating curly braces which are always { (open curly brace) and } (close curly brace). The function definition exists between these two braces.

```
void setup() {  
  
}
```

When you see `setup()` in the text if you were to read the it out loud you would say "setup function" where the parenthesis indicate that it is a function.

The `setup()` is a function that is run only once at the start of the program. Setup as its name implies is usually used to setup the processor for the remainder of the program.

Again, the lines that are in a gray/green font are comments.

```
// the setup function runs once when you press reset or power the board  
  
// the loop function runs over and over again forever
```

However, these comments are single line comments and are delimited by the `//` at the beginning of the comment and a newline at the end.

Finally, a function call to `pinMode(pin, pin-mode);`.

```
pinMode(LED_BUILTIN, OUTPUT);
```

This function configures **pin** of the control board (pin 21) to be of **pin-mode** type **OUTPUT**. The values `LED_BUILTIN` and `OUTPUT` are said to be the parameters that are passed to the function. The parameters for the function are encapsulated by parenthesis and separated by commas. Finally the call to `pinMode()` ends with a semicolon.

The second function is called `loop()`.

```
// the loop function runs over and over again forever  
void loop() {  
    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)  
    delay(1000); // wait for a second  
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW  
    delay(1000); // wait for a second  
}
```

The `loop` function has four function calls and four comments. The `digitalWrite(pin,state);` is used to set the selected **pin** to the selected **state**. In our case we are setting pin 21 to either **HIGH** or **LOW**. The `delay(ms);` causes the program to wait **ms** milliseconds.

As you might have guessed the `loop()` is a function that is run over and over. Loop is the location where you put the portion of your code you want to repeat, to perform the main task of the device.

So the `loop()` causes pin 21 to go HIGH, waits for 1 second then causes pin 21 to go LOW and waits an additional second. The program then repeats until the board is reset or until power is lost.

User Sketches

You cannot modify and save an example sketch. If you modify the program and want to save it, it will then become a User Sketch. They are accessible from the Open Icon and File->Open->Sketchbook

The default save location in Windows is C:\Users\\Documents\Arduino

The default save location on a MAC is /Users/<username>/Documents/Arduino

Where <username> is the name of the computer user that you are logged on to the computer as.

Modified Blink

Replace the `loop()` function in your code with the one shown below. Save it as a user sketch, compile and upload it to a board.

```
void loop() {
  digitalWrite(PIN_LED1, HIGH); // set the LED on
  delay(1000); // wait for a second
  digitalWrite(PIN_LED1, LOW); // set the LED off
  delay(1000); // wait for a second
  digitalWrite(PIN_LED1, HIGH); // set the LED on
  delay(500); // wait for half a second
  digitalWrite(PIN_LED1, LOW); // set the LED off
  delay(500); // wait for half a second
}
```

Check off

Call the instructor over for a check off. You will need to demonstrate the uploading of the program to the board.

Part 10 - More advanced Blink

Alter the loop so that you will get a pattern of one long, two short, one long, then one short blink.

Check off

Call the instructor over for a check off. You will need to demonstrate the uploading of the program to the board.

Lab 1 Homework

Get Arduino IDE with chipKIT core working on your own

Get Arduino IDE working on your own computer or a computer you have access to outside of class. Get the blink sketch working and uploading to a board on your own.

Investigate chipKIT on the internet

chipKIT Web Page

chipkit.net

chipKIT Forum

The chipKIT forum (<http://chipkit.net/forum/>) is a place people can go to get help with a problem they are having in using chipKIT compatible boards, get code other people have written for their own projects, or share code they think will be helpful to others.

Investigate Arduino language reference

Web Site: <https://www.arduino.cc/reference/en/>

Investigate Fubarino SD

Web Site: fubarino.org

Development Repository: <https://github.com/fubarino/fubarino.github.com>

Investigate Ardublock

<https://learn.sparkfun.com/tutorials/digital-sandbox-experiment-guide/setting-up-arduino-and-ardublock>

Investigate Open Source

Many of the boards you are going to be using are Open Source. Open Source is an idea where a Product or Software has its schematics and source code available for other people to use. This can help users of the product to get the most out of it because if they don't like a part of it they have the information needed to change it. Open Source also allows people to make new products based off old ones without having to start from scratch.

Be prepared to discuss advantages and disadvantages in next class.

Investigate Wiring Pin Abstractions

<http://wiring.org.co/>

Most brands of microcontrollers have different ways that the software uses to control the output pins of the part. This compiler uses Pin Abstractions to prevent this from being a problem, it knows what kind of chip is being programmed and uses the correct method. They also allow the designer of the board to use pins that are not adjacent on the processor as sequential abstracted pins.