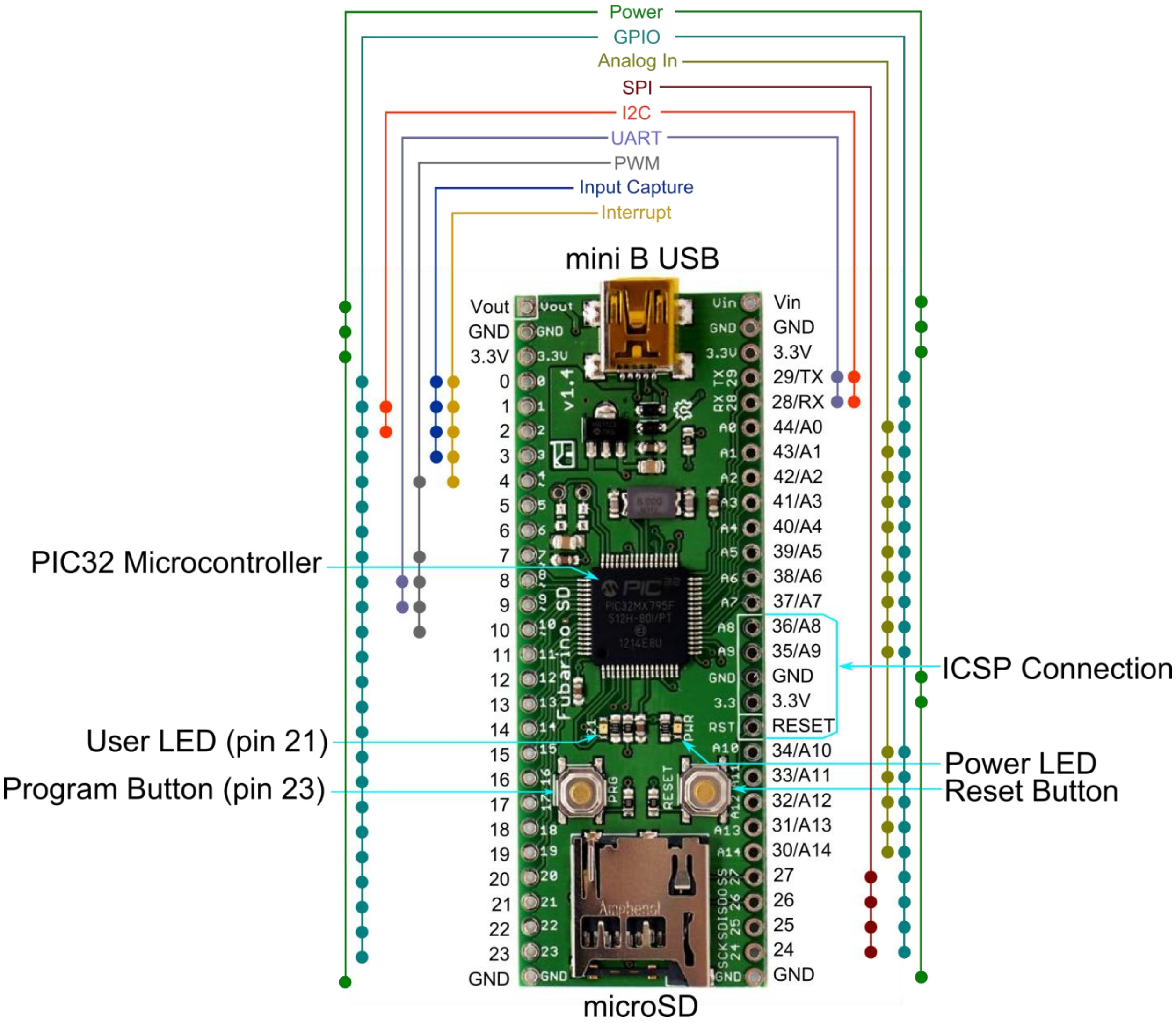
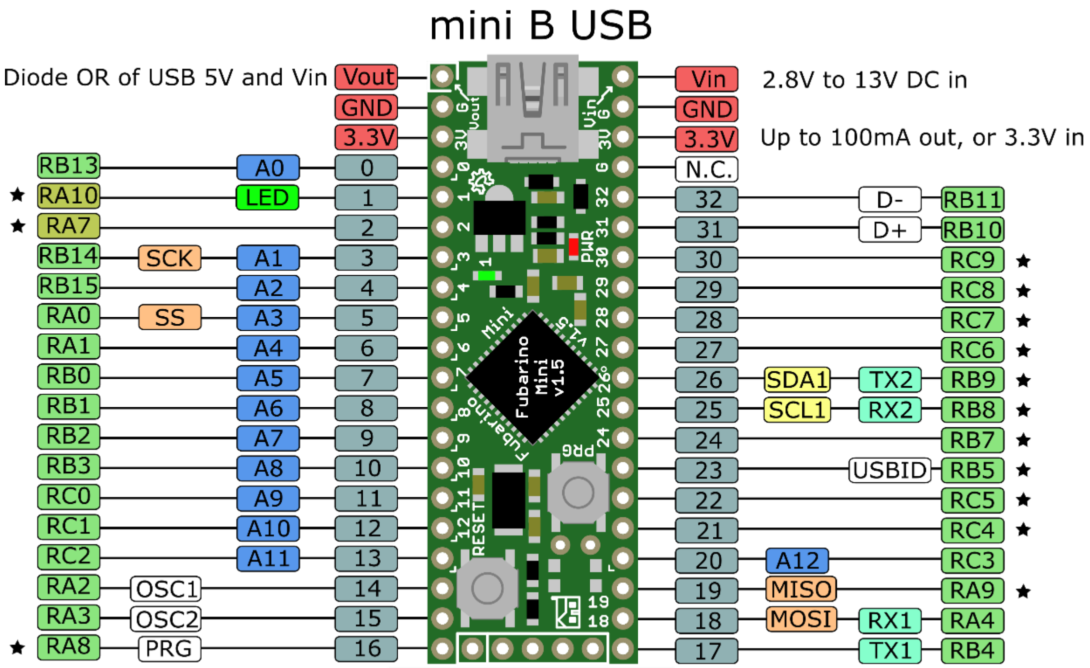
Lab 2 - Powering the Fubarino, Intro to Serial, Functions and Variables

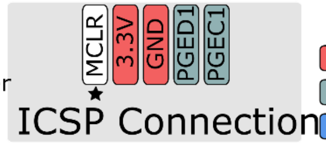
Part 1 - Powering the Fubarino SD / Fubarino Mini

The Fubarino SD is a 56 pin device. The Fubarino SD is a 40 pin device + 5 pin ICSP header at the bottom of the board. Each pin on a chipKIT device falls broadly into one of 9 categories: Power, GPIO, Analog In, SPI, I2C, UART, PWM, Input Capture or Interrupt. The difference between these categories will become clearer as the class progresses. Broadly because many pins can fall into more than one of these categories. The image below highlights in general what pins fall where and will be useful in future labs to help identify different pins on the device.



Fubarino Mini v1.5 pins



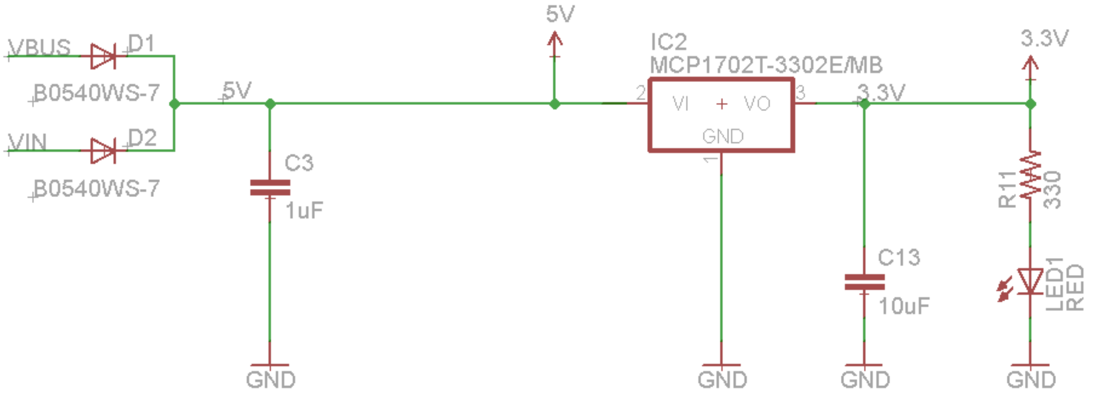


- Press and hold PRG while pressing and releasing RESET to enter bootloader mode over USB.
- Green LED : Pin 1
- Red LED is on 3.3V power
- PRG button : Pin 16
- USB : `Serial`
- TX1/RX1 : `Serial0`
- TX2/RX2 : `Serial1`
- PPS = Peripheral Pin Select
  This includes SPI/UART/INT/OC/PWM/etc.
  Note that each periperhal can not be mapped to any PPS pin, but only 8 of the possible 31.
- Power input through USB, Vin, or 3.3V pins

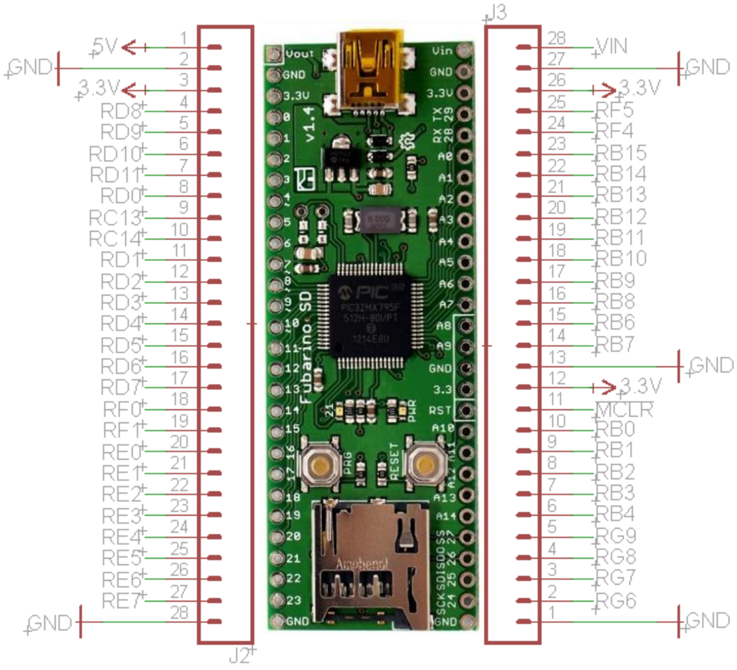
★ 5V Tolerant
🟥 Power/Ground
▨ Arduino Digital Pin
🟦 Analog input
🟩 PIC32 Pin with PPS
🟧 Serial (SPI)
🟨 Serial (I2C)
▧ Serial (UART)
▩ PIC32 Pin no PPS
□ Default configuration uses these pins (for USB, crystal, PRG button, etc.) and they are not available for general purpose GPIO unless the confugration is changed.

Schematic Analysis

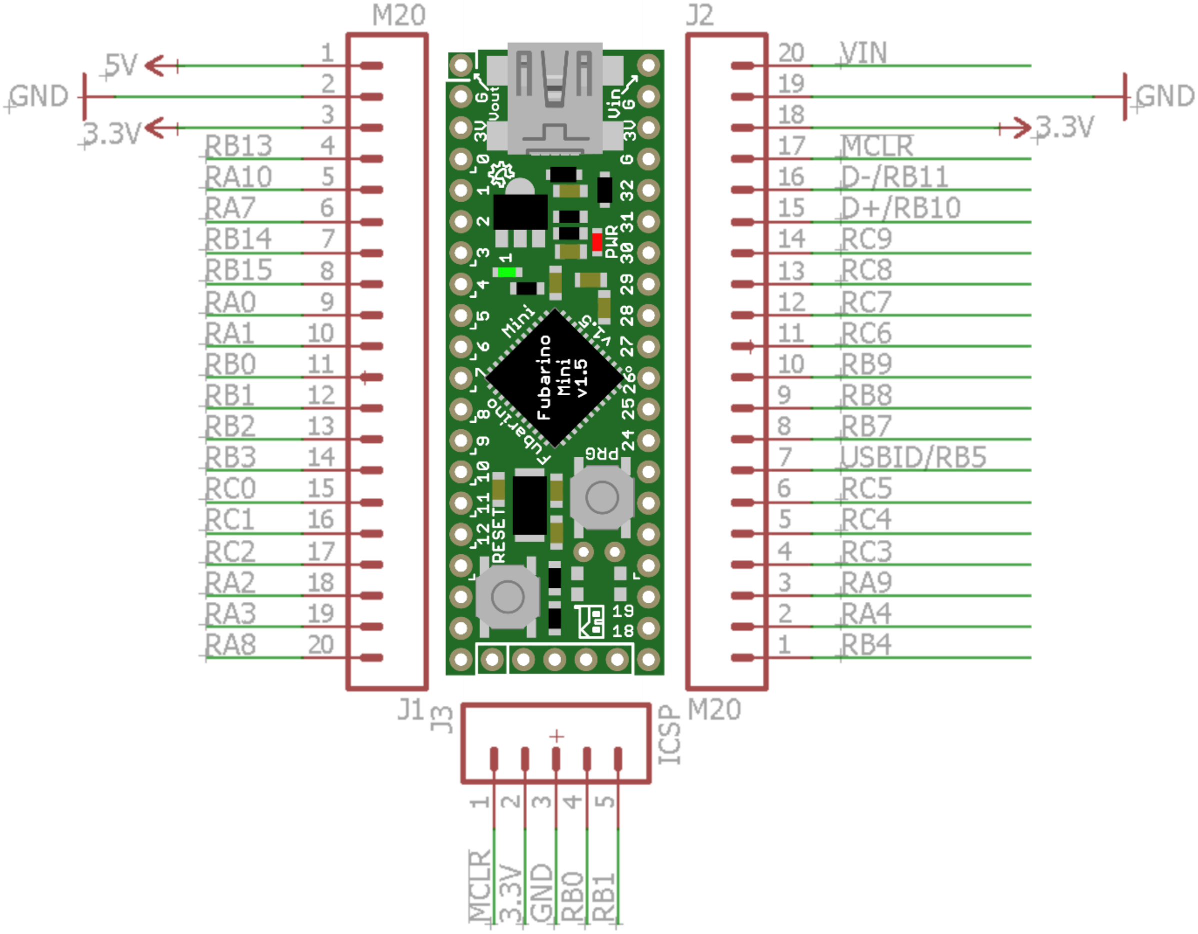
When you look at the pinout of the Fubarino SD or the Fubarino Mini you will notice that there are power pins made up of four nets: Vout, Vin, 3.3V and GND. The complete power supply circuit for the Fubarino SD and Fubarino Mini is shown below and all four of the afore mentioned nets are shown including two additional nets: Vbus and 5V (Note: the 5V net in the schematic is connected to the Vout pin of the Fubarino SD/Mini).



In this instance Vbus is an input 5V signal supplied from a PC's (host devices) USB port. Although Vbus is 5V it should not be confused with the 5V net in Fubarino SD schematic. Shown below is the pin connections of the Fubarino SD from the schematic overlaid with the Fubarino SD. You will notice that where the board is labeled Vout the actual net is 5V.

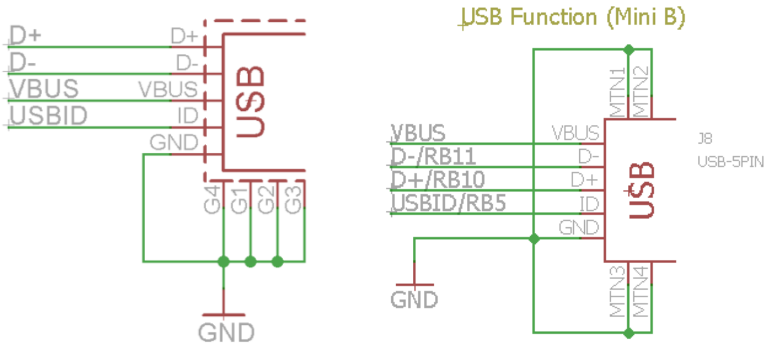


Fubarino SD with Schematic Overlay



Fubarino Mini with Schematic Overlay

Shown below is the USB connector from the Fubarino schematic, the Vbus signal is clearly marked and in this case it is a power input to the board.



Fubarino SD / Fubarino Mini

In lab one we have already seen that connecting a PC USB cable to the Fubarino will power the board. In analyzing the schematic we can see the current path from the PC to the PIC32 microcontroller is as follows. Power (5VDC) comes in the USB port on Vbus which makes its way to D1 anode. It then drops 0.7V across D1 with 4.3VDC ending up at the VI input to IC2 (3.3V linear regulator) which in turn produces 3.3VDC that is used to power the LED1 (a red power LED) and provide power to the microcontroller.

Powering Fubarino using an external power supply.

Create a new blink sketch that blinks the LED on for 1 second then follows with four 0.250 second blinks. Program the board as described in Lab 1. Next remove the USB cable and power the board using an external power supply with Vin connected to +5VDC and one of the many GND pins connected to the ground of the power supply.

Check off

Part 1 of the lab is complete, call the instructor over to demonstrate your completed lab.

Be prepared to demonstrate the blink sequence and answer the following questions:

What voltage does the PIC32 run at on the Fubarino board?

What roll do the D1 and D2 diodes play in the power supply circuit?

What voltage will you see on the Vout pin of the Fubarino if it is powered by the USB cable?

Part 2 - Serial Communications and Aruduino IDE Serial Monitor

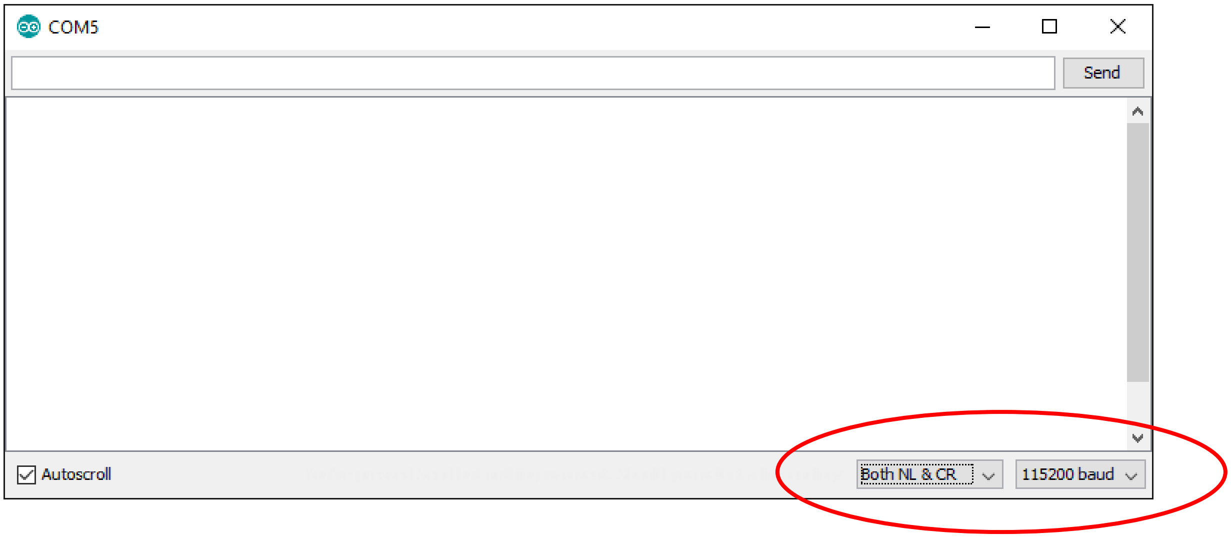
When writing programs for chipKIT or Arduino boards one of the most powerful tools at our disposal for insight into how our program is running is the ability to use serial communications to transmit data out of the serial port. In this section we introduce serial communications and the built in serial terminal of Arduino.

Launch the Arduino IDE and copy and paste the following code into the editor.

```
/*
  Communications Test and Conversion Chart
 */

void setup()
{
  // Open the Fubarino SD USB port as a Serial Terminal
  Serial.begin(115200);
  // Print something out of the serial port asap
  Serial.println("Pre-delay communications");
  // Give us a chance to open the serial terminal
  delay(3000);
  // prints title with ending line break
  Serial.println("3 seconds have gone by since the sketch started");
}

void loop()
{
  // create an 8-bit unsigned static variable in the loop function
  static unsigned char count = 0;

  Serial.print("base 10 = ");
  Serial.print(count, DEC);    // DEC constant implies base 10
  Serial.print(", base 16 = ");
  Serial.print(count, HEX);    // HEX constant implies base 16
  Serial.print(", base 8: ");
  Serial.print(count, OCT);    // OCT constant implies base 8
  Serial.print(", base 2: ");
  Serial.print(count, BIN);    // BIN constant implies base 2
  Serial.println(".");         // Print . and new line
  delay(250);
  count = count + 1;           // Increment the count variable
}
```

Upload the program to your board, when the upload is complete open the serial monitor by clicking on the following icon:

Once the serial terminal is open the drop down boxes in the bottom right corner should be set to "Both NL & CR" and "115200 baud" as shown below.



☑ Autoscroll

Both NL & CR

115200 baud

Analysis:

The setup() has four function calls. The delay() we have seen before but the Serial functions are new.

```
  Serial.begin(115200);
```

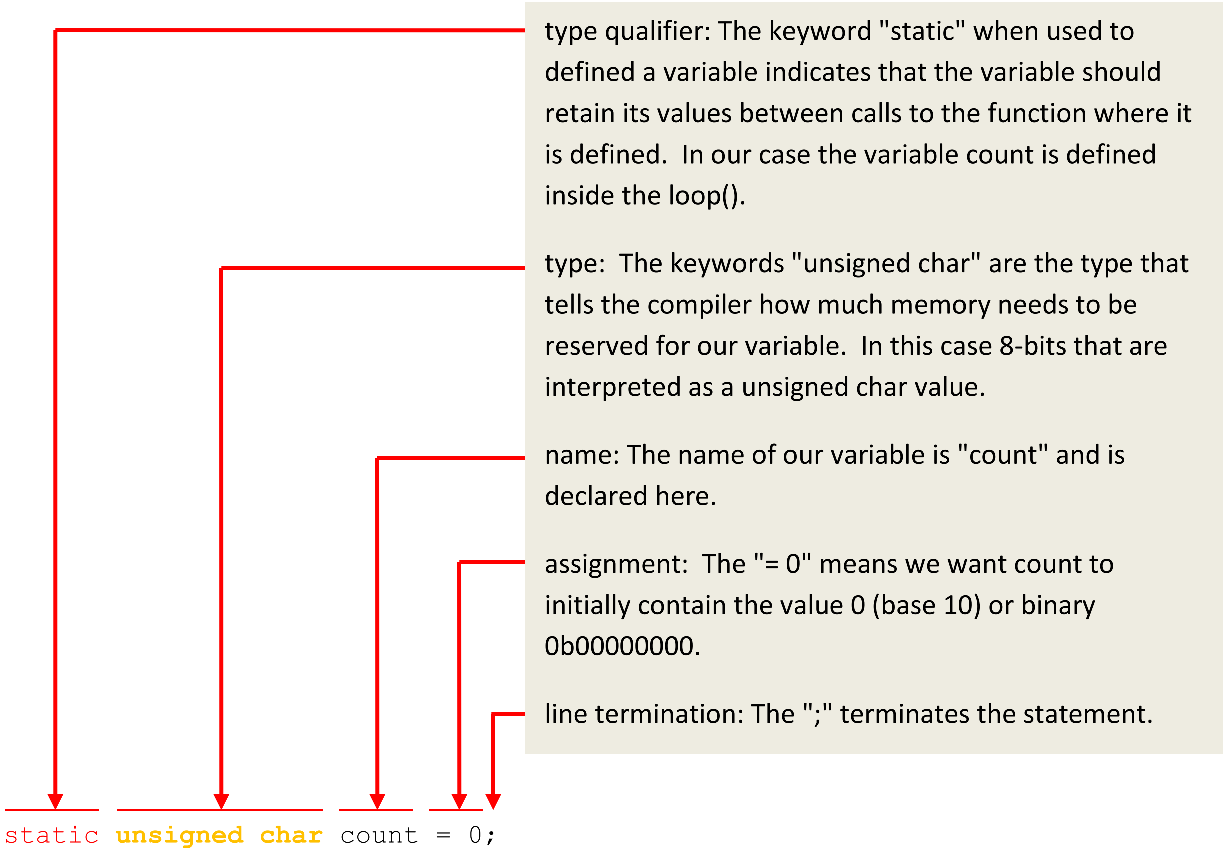
Opens the serial port on your board for transmission and reception of 115,200 bits per second (or 115,200 baud). The baud rate is only important if your board has a serial to USB conversion chip (typically manufactured by FTDI). If the board speaks native USB then this parameter will be ignored and it will communicate at the fastest speed available. The following chart shows which boards use the serial to USB conversion chip.

Board	Com Port
Fubarino SD	Native USB
Uno32	FTDI
Max32	FTDI
uC32	FTDI
UAV100	Native USB
Quick-240	Native USB

```
  Serial.println("Pre-delay communications");
  Serial.println("3 seconds have gone by since the sketch started");
```

The two Serial.println() send data from our program out the serial port and to the serial monitor of the Arduino IDE.

In loop() we see for the first time the creation of a variable called count. There are several notable things about what is expressed in this single statement.



type qualifier: The keyword "static" when used to defined a variable indicates that the variable should retain its values between calls to the function where it is defined. In our case the variable count is defined inside the loop().

type: The keywords "unsigned char" are the type that tells the compiler how much memory needs to be reserved for our variable. In this case 8-bits that are interpreted as a unsigned char value.

name: The name of our variable is "count" and is declared here.

assignment: The "= 0" means we want count to initially contain the value 0 (base 10) or binary 0b00000000.

line termination: The ";" terminates the statement.

```
static unsigned char count = 0;
```

The Serial.print() is equivalent to the Serial.println() with the exception that the terminal does not advance to the next line when printing.

```
  Serial.print("base 10 = ");
```

In the above statement the string of characters between the parenthesis is called a "literal string" and represents what will be sent to the serial terminal not including the leading and trailing quotation marks. This literal string is said to be the parameter that is passed to the function. The next statement we will look at has two parameters. Function parameters are separated by commas.

```
  Serial.print(count, DEC);     // DEC constant implies base 10
```

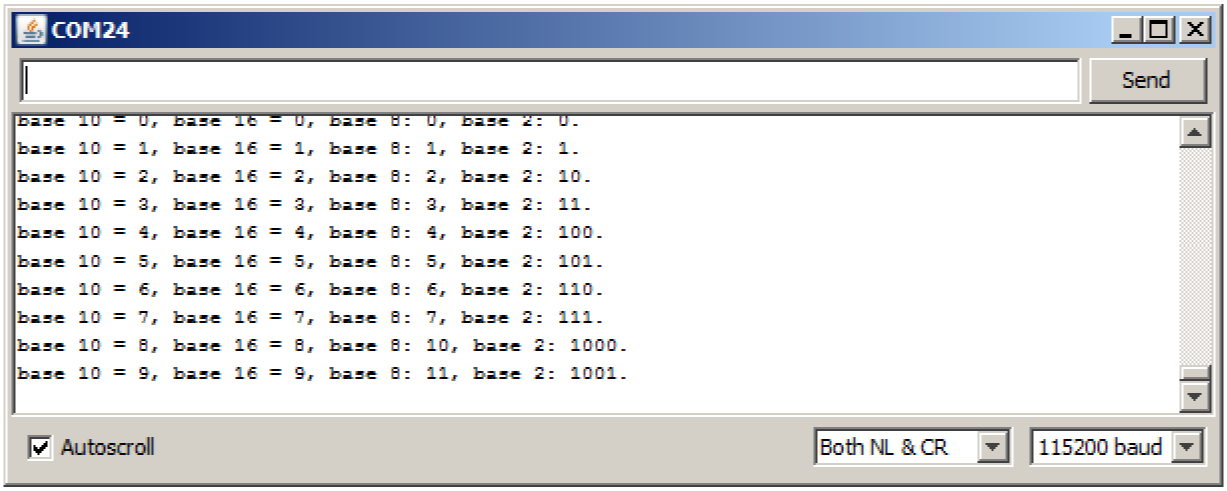
The two parameters in the above statement are the variable "count" and the constant "DEC". This statement will cause the value stored in the variable count to be sent to the serial terminal. The DEC literal tells the function to convert the binary value stored in the memory of the computer represented by the count variable to a decimal number. The interpretation for the statements that follow this one is the same with the exception that the constants HEX, OCT and BIN cause the binary values stored in count to be rendered in hexadecimal, octal and binary respectively.

The final statement in our program that we have not seen before is:

```
  count = count + 1;              // Increment the count variable
```

This causes the value that is stored in the memory location "count" to be assigned the value of "count" plus one more (+1). We say that count is being incremented.

The output of this program should look something like what is shown below:



```
base 10 = 0, base 16 = 0, base 8: 0, base 2: 0.
base 10 = 1, base 16 = 1, base 8: 1, base 2: 1.
base 10 = 2, base 16 = 2, base 8: 2, base 2: 10.
base 10 = 3, base 16 = 3, base 8: 3, base 2: 11.
base 10 = 4, base 16 = 4, base 8: 4, base 2: 100.
base 10 = 5, base 16 = 5, base 8: 5, base 2: 101.
base 10 = 6, base 16 = 6, base 8: 6, base 2: 110.
base 10 = 7, base 16 = 7, base 8: 7, base 2: 111.
base 10 = 8, base 16 = 8, base 8: 10, base 2: 1000.
base 10 = 9, base 16 = 9, base 8: 11, base 2: 1001.
```

☑ Autoscroll   Both NL & CR   115200 baud

Check off

Modify the above program so that the loop() only prints out the base 16 value of count.

Verify and upload the new program.

Call the instructor over for check off when the program is working.

Be prepared to answer the following questions:

What happens if you remove the static keyword from the declaration of count? (hint: try it)

```
  unsigned char count = 0;
```

What is the maximum value that count achieves?

What is implied about the number of bits of storage for a char variable type based on the maximum value of count?

Part 3 – C Variable Databus Size

The creators of the C programing language made a decision that seemed reasonable and appropriate at the time. They made the amount of memory allocated for a specific variable type based on the size of the data bus of the computer they were using. That is to say the original size (number of bits) for an int type was dependent on the computer and compiler that was being used for compilation. This creates a problem when we write a program for one computer then try to move them to another computer that has a different data bus size. Namely, the programs may behave differently. Try the following program on an Arduino and a chipKIT board:

```
void setup()
{
  Serial.begin(115200);  // Open the Serial Port
  delay(3000);  // Give us a chance to open the serial terminal
}
void loop()
{
  static unsigned int count = 1;
  Serial.print("base 10 = ");
  Serial.print(count, DEC);    // DEC constant implies base 10
  Serial.println(".");         // Print . and new line
  delay(1000);
  count = count * 10;           // Increment the count variable
}
```

Check off

Be ready demonstrate the programing running and have available the largest number display on an Arduino and a chipKIT board.

What is the primary issue this program demonstrates with trying to create a program that will run the same on both an Arduino and a chipKIT board?

Part 4 – Resolving C Variable Databus Size

There is a solution to this issue presented in part 3.

In the previous section we saw the creation of variable that behaved differently when executed on two different architectures due to the way the C/C++ programing language was created. This problem is not exclusive to Arduino and chipKIT but comes up often in computing and programming. One of the goals of this material is to give the reader the ability to develop programs on both chipKIT and Arduino platforms. By solving this problem here, we see how this issue might be resolved in other applications. So the reason this problem exists is that an int for an 8-bit Atmel AVR microcontroller (Arduino) indicates 16-bits of storage (which is not even actually the data bus size of this chip) and is not the same as an int for a 32-bit Microchip PIC microcontroller (chipKIT) which is 32-bits of storage. To resolve this issue you can use some macros from a library called stdlib.h that is included automatically when building your program.

#include <stdint.h>

Since this class was originally developed I have discovered a library that is becoming the defaco way to make development cross platform called stdint.h. Previous classes implemented macros to accomplish this same goal. So instead of the uint8_t the notation in the previous class would have been us8. If you run across this notation in this book, please let me know so I can correct it. If you see this notation, the table below shows an equivalent in my old notation vs the stdint.h notation:

Previous Years Type Notation	stdint.h type notation
us8	uint8_t
us16	uint16_t
us32	uint32_t
s8	int8_t
s16	int16_t
s32	int32_t

Implementation

Change the type in the last program from the previous section by updating this line of code from this:

```
static unsigned int count = 1;
```

to this:

```
static uint32_t count = 1;
```

Check off

Verify and upload the new program on both an Arduino and a chipKIT board.

Call the instructor over for check off when the program is working.

What is Be prepared to answer the following questions:

What is the maximum value for the Arduino and chipKIT?

Why is this an improvement over the previous version of the program.

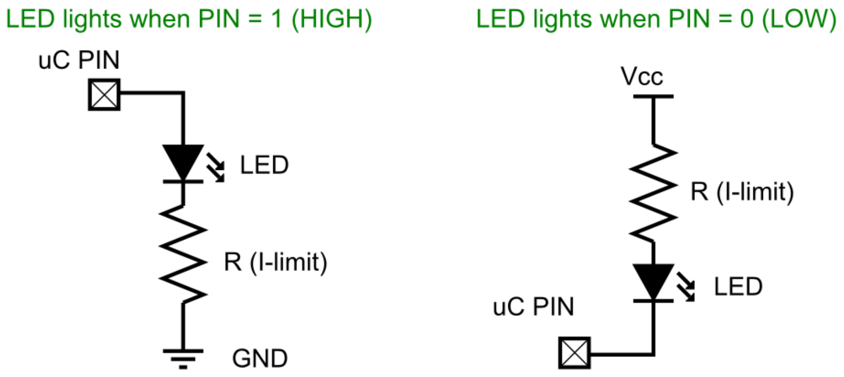
Part 5 - LED Chase Patterns (Knight Rider)

https://www.youtube.com/watch?v=oNyXYPhnUIs&feature=youtu.be&t=4

All chipKIT devices have a selection of pins that can be used for digital input and output (I/O) or GPIO (General Purpose Input Output). Each digital I/O pin has a minimum of two modes they can operate in: INPUT or OUTPUT. When configured as OUTPUT the signal presented can be a logic HIGH or LOW. When configured as INPUT the pin becomes HIGH IMPEDANCE so that digital signals can be read. The Fubarino SD is a 56 pin device of which 45 pins can be used as GPIO. The Fubarino Mini is a 45 pin device of which 35 pins can be used as GPIO. Of these digital I/O pins, 15 (FubSD) / 12 (FubMini) are shared with the analog input module of the PIC32.

In this section we will extend our use of the pinMode() and pinWrite() to utilize some of the other digital I/O pins of the chipKIT board to light additional LED's. To do this we are going to connect four more LED's to pins that we will turn into outputs so that we can display some chase patterns. A lighting chase pattern is one where the lights appear to be chasing each other.

LED's for the most part over a certain voltage will always drop the same voltage. With constant voltage and nothing to limit the current, the power flowing out of the LED will be enough to melt its die. So to prevent our LED's from blowing up we will add current limit resistors. The microcontrollers on the chipKIT and Arduino boards are capable of sourcing and sinking current so this gives us two possible ways in which we can hook our LED's up to make them light. In the two topologies shown below one will cause the LED to light when a logic 1 (HIGH) is presented on the pin and the other will cause the LED to light when a logic 0 (LOW) is presented on the pin.



Typically, we like to think of logic 1 as either TRUE or ON, so let's use the topology that creates this effect.

The maximum source current of an output pin also should be less or equal to 20mA. Using ohms law the required value of the current limiting resistor for the LED can be calculated as below:

$R = (V_S - V_F)/I_f$

V_S is the 3.3V supply voltage, V_F is the Light Emit Diode voltage drop and I_f is the LED forward current; V_F and I_f come from the LED datasheet. If the LED current is 20mA, the resistor will be:

R = (3.3 - 2)V/0.020A = 65 ohms
Since our kit does not include 65 ohms a value close to that will have to be chosen.

Connect the LED's

Any of the GPIO pins of our board can be used as digital output (refer to the board diagrams at the beginning of this lab). In this example we will connect our LED's to pins 0-3 of our board.

Look up the datasheet for the LED you are using to determine how to connect the LED's to your board.

The Program

The following program will create a simple chase pattern.

```
void setup() {
  Serial.begin(115200);
  pinMode(0,OUTPUT);
  pinMode(1,OUTPUT);
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
}

void loop() {
  Serial.println("looping");
  digitalWrite(0,HIGH);
  delay(200);
  digitalWrite(0,LOW);
  digitalWrite(1,HIGH);
  delay(200);
  digitalWrite(1,LOW);
  digitalWrite(2,HIGH);
  delay(200);
  digitalWrite(2,LOW);
  digitalWrite(3,HIGH);
  delay(200);
  digitalWrite(3,LOW);
}
```

Create a new sketch, copy and paste this code into the Arduino IDE editor. Verify and upload the program to the board. After the program has uploaded, open the serial monitor and note the "looping" text each time the patter repeats.

Analysis

We have seen all the functions in this program before but may have glossed over the meaning.

```
  pinMode(0,OUTPUT);
```

The above function pinMode configures a pin on the board for either OUTPUT, INPUT or some other configurations that we will not talk about for now. This takes two parameters. The first parameter specifies the abstracted Wiring pin number of the board and the second is the mode. The macros OUTPUT and INPUT are predefined for us.

```
  digitalWrite(0,HIGH);
  digitalWrite(0,LOW);
```

The above function digitalWrite sets the output voltage to either a logic high or logic low for a pin configured as output. The actual voltage at the pin may vary depending on the voltage that the microcontroller on the board runs at. In this book all boards discussed run at 3.3VDC except the Arudino Uno which runs at 5VDC.

Check off

Modify the program so that instead of always going in the same direction the LED's change direction and return to where they started before repeating (Michael Knight would be proud of you).

What value resistor did you choose?

What will the resultant current be through the LED if the LED drops 2V?

Part 6: Bicolor Blink pattern

Connect up the other color of the LED’s to pins 4-7.

Modify the program so that the newly LED pins are set to output and create a new pattern that is a mix of red and green led’s being lit up.

Check off

Call the instructor over for check off.

Part 7 - LED Binary Counting and Functions

This section we are going to use the connected LED's to display a binary count sequence. To do this we need to introduce functions. The setup() and loop() are called by the core wiring libraries but a user function is created and called by from either the setup(), loop() or some other function we create.

In this section we want to merge the ideas from the last two sections in that we will have four LED's that will display the output of our variable "count" in not only the serial monitor but also on the four LED's that are now connected to our board. This will accomplished by creating a function called nibbleToPins.

JARGON

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.

Create a new sketch and copy and paste the following code into the Arduino IDE text editor.

```
void nibbleToPins(uint8_t in, uint8_t pin3, uint8_t pin2, uint8_t pin1,
uint8_t pin0) {
  // use bitwise & operator to create logic condition that is
  // used to light bits as if in the same nibble.
  digitalWrite(pin0, in & 0b00000001);
  digitalWrite(pin1, in & 0b00000010);
  digitalWrite(pin2, in & 0b00000100);
  digitalWrite(pin3, in & 0b00001000);
}

void setup() {
  Serial.begin(115200);
  pinMode(0,OUTPUT);
  pinMode(1,OUTPUT);
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
}

void count_up() {
  static uint8_t count = 0; // declare count and set its value to 0
  nibbleToPins(count,0,1,2,3); //call a function that sets all 4 pins
  delay(200);
  count = count + 1;
}

void loop() {
  Serial.println("looping");
  count_up();
}
```

Analysis

The loop() has one new concept, it calls the nibbleToPins(). The function takes five parameters: An 8-bit unsigned value that is to be displayed on the output and four pin numbers where the value is to be deconstructed into individual bits.

The nibbleToPins() itself uses the digitalWrite() that we have seen before, but instead of passing literal values to the function, variables that have been passed to the function are used.

The function definition looks like this:

```
void nibbleToPins(uint8_t in,
                  uint8_t pin3, uint8_t pin2,
                  uint8_t pin1, uint8_t pin0) {
}
```

The five parameters that are passed to the function are in, pin3, pin2, pin1, pin0.

in is the value to be deconstructed

pin3 is the pin where the most significant bit is to be displayed.

pin2 is the pin where the second most significant bit is to be displayed.

pin1 is the pin where the third most significant bit is to be displayed.

pin0 is the pin where the least significant bit is to be displayed.

Consult the appendix of this lab for further analysis of the nibbleToPins().

Check off

Call the instructor over for check off.

Part 8 - LED Binary Counting and Functions

Create a new function called count_down. Make it count from 15 down to 0. Modify the loop() to call the count_down() instead of count_up();

Check off

Call the instruction over for check off.

Part 9 - LED Chase Patterns Using Arrays

Add this function to the code from the previous section after the nibbleToPins().

```
void chase(void)
{
  static uint8_t i=0;
  uint8_t pattern1[]={ 0b00000001,
                       0b00000010,
                       0b00000100,
                       0b00001000,
                       0b00000100,
                       0b00000010,
                       0b00000001};

  nibbleToPins(pattern1 [i],0,1,2,3);
  delay(500);
  i = (i + 1) % sizeof(pattern1);
}
```

And change loop to this

```
void loop() {
  Serial.println("looping");
  chase();
}
```

Analysis

This code introduces three new ideas: Arrays, indexing the array and the use of the sizeof function. These concepts are explained in the week 3 lecture notes.

Call the instructor over for check off be prepared to explain how chase works.

Part 10 - Remove the Pause

If you watch the sequence you will notice, there is a slight pause when the pattern restarts. Study the pattern and remove this pause. This is called a one-off error and is extremely common in programming.

Check off

Call the instructor over for check off be prepared to explain how you fixed the issue.

Lab 2 Homework

Create a program that it counts down from 10 to 0 while displaying the value **both** on the LED's and the serial terminal. The program should start with a delay of 1 second between counts and decrease by 100ms with every count so that it speeds up when counting. When it reaches zero all LED's come on for 1 second then shut off. The program should not repeat.

This homework is not required to be turned in, nor is it graded. This assignment is provided as a self-test. If you are unable to complete this programming assignment on your own you are having trouble understanding this material and should seek help. Also, even if you think you understand how to do this and don't think you need to actually attempt it, I suggest you try. It amazing how difficult a seemingly simple task can seem.

Lab 2 - Appendix

Introduction

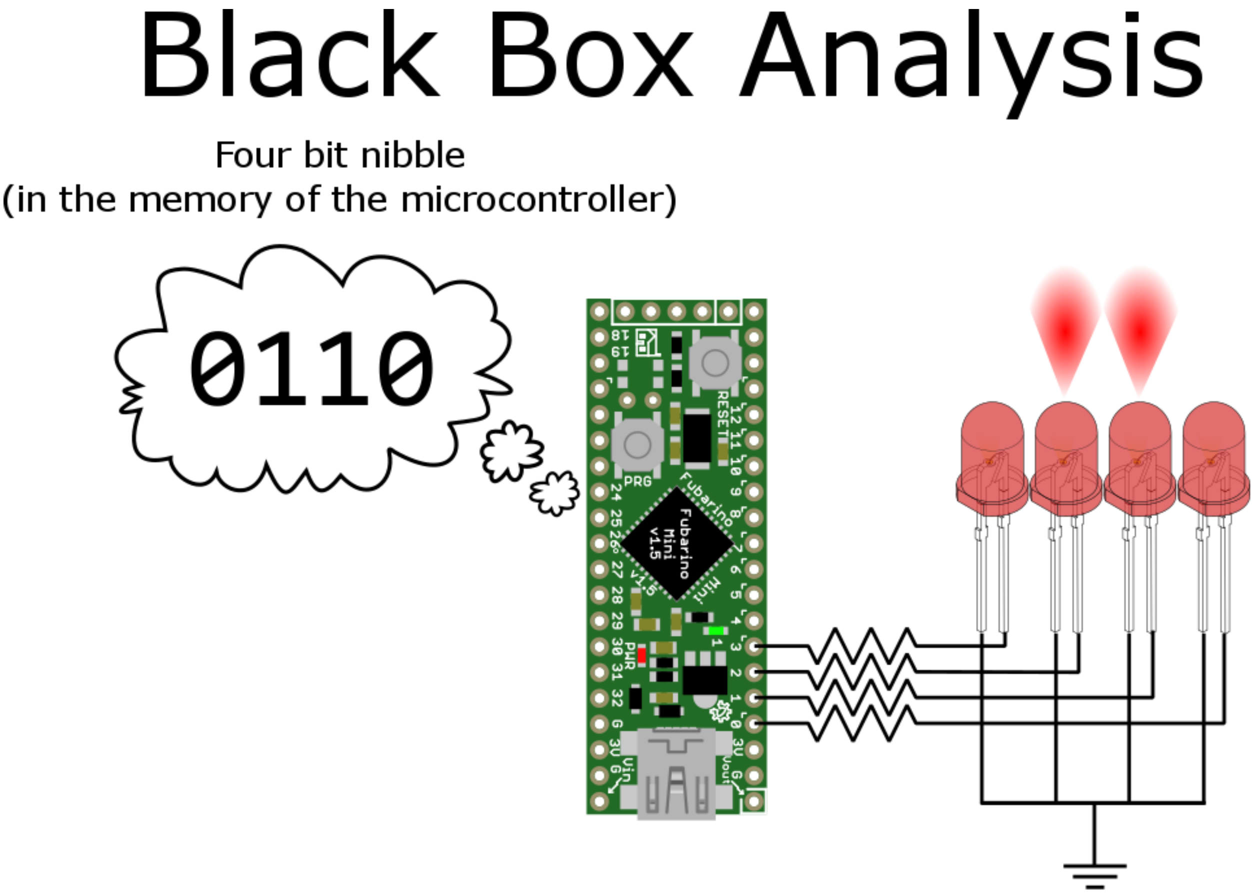
In this video we are going to create a function that will run on an Arduino or Arduino compatible development board, such as at chipKIT Fubarino Mini, that will take a 4-bit number and display it on four independent LED's connected to the board.

Black Box Analysis

In electronics, computer science and other disciplines the term "black box" is often used to refer to some portion of a complex system that the inner workings of the system can be ignored. This is often done to hide the complexity in order to first understand the goal or the system at a "higher level". Sometimes it is sufficient to understand a system at the black box level and never dive in deeper. Other times you need to know both. When writing or analyzing programs it is convenient to think of functions as black boxes that provide some useful functionality.

Analysis of the function nibbleToPins()

To understand the nibbleToPin() we are going to first treat it as a black box to understand the goal of the function. Then we will dive in deep to understand what is going on inside the function.



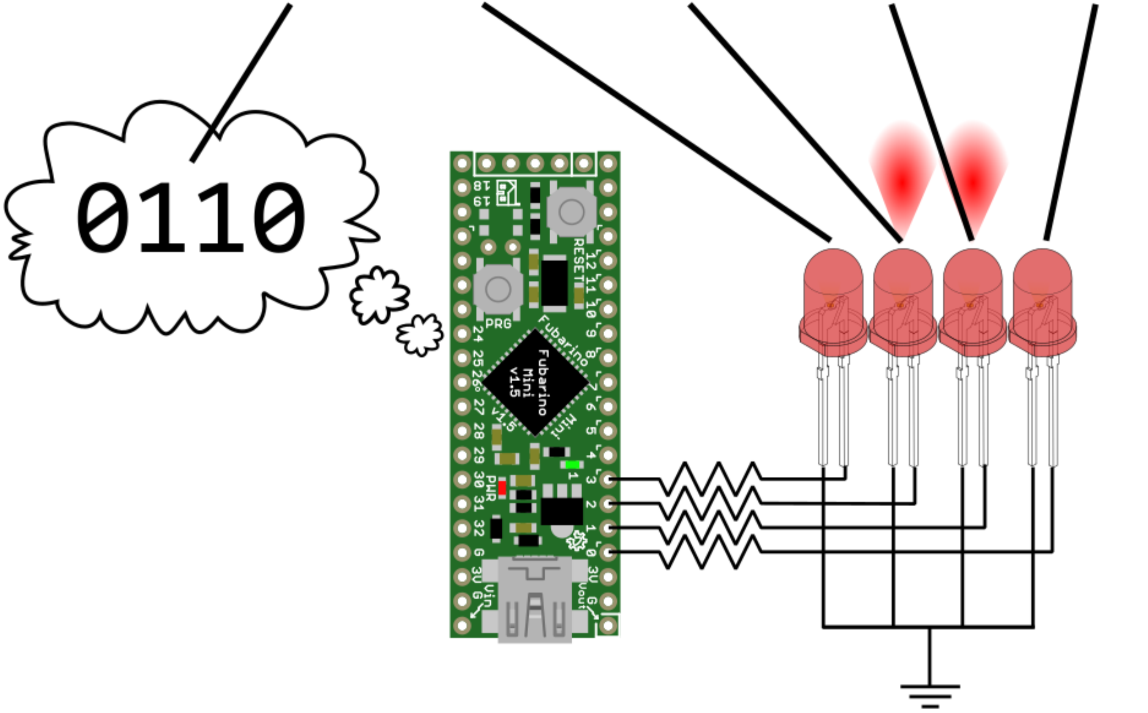
Black Box Analysis

Four bit nibble
(in the memory of the microcontroller)

My goal in creating this function was to take a 4-bit number (a nibble) stored it in an 8-bit variable and output each bit of the nibble onto a single led connected to physical pins of the Fubarino.

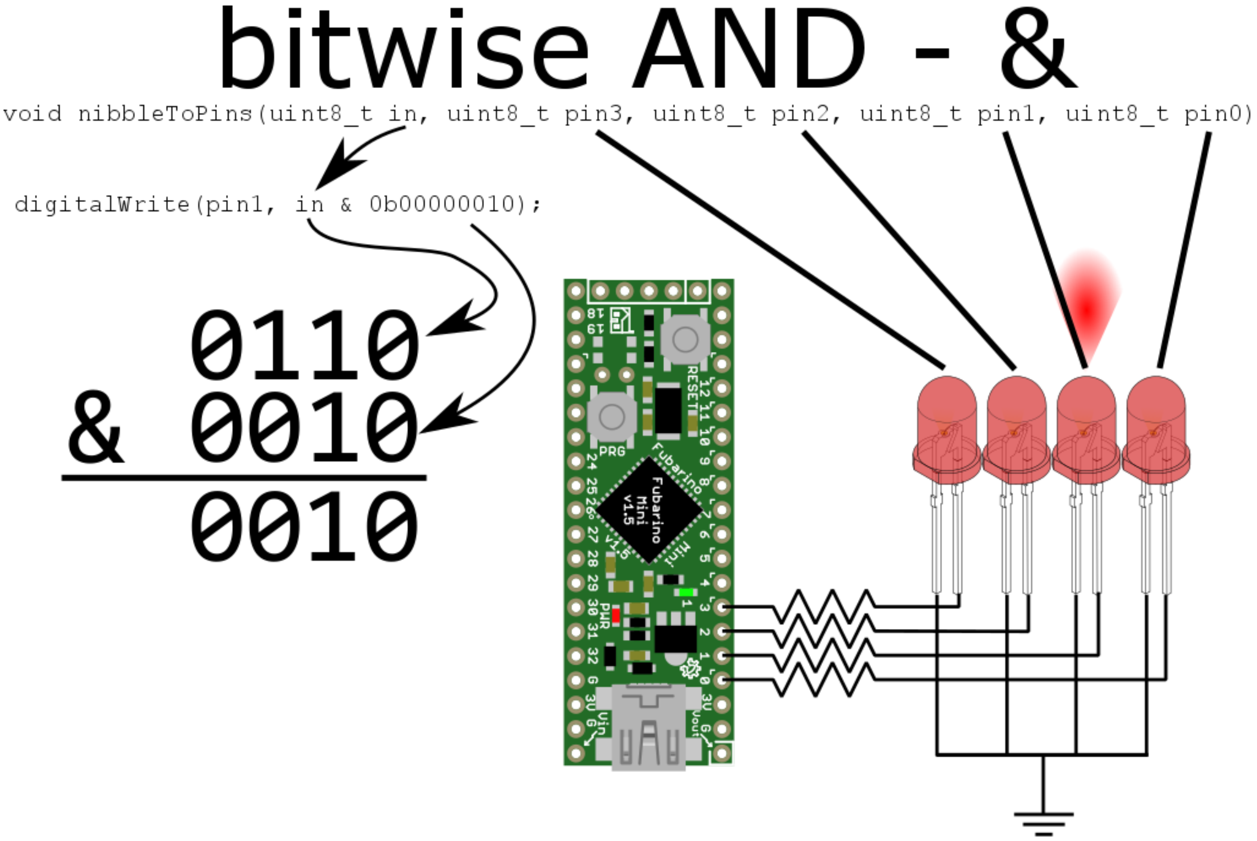
This function will be called nibbleToPins(). I'm going to have the function take as input five parameters. These five parameters will represent the nibble I wish to display and the four pins I wish to display the nibbles on. These are what are the names shown in the parenthesis after the name of the function:

```
void nibbleToPins(uint8_t in, uint8_t pin3, uint8_t pin2, uint8_t pin1, uint8_t pin0)
```



Each of these parameters is of type uint8_t. This just means that the maximum number of bits stored in each variable is 8 and the bits will represent an unsigned integer value (0 through 255). The parameter named “in” will be my 4-bit nibble. The parameters named “pin3”, “pin2”, “pin1” and “pin0” will store the pin number of each specific bit of my nibble to go to.

To extract a single bit from the “in” variable the bitwise AND function represented in the C programming language as an ampersand character ‘&’ is used. To this we need to put a one in the each the bit position we wish to extract. When we do this the result of the bitwise AND operation will be non-zero if there a one in the corresponding bit position for the “in” variable. If this result is then passed to the digital write command this will cause the pin to reflect a one if the values is non-zero and a zero if the value is zero. So, if the value of in = 0x06 (0b00000110) then and we and it with 0x02 (0x00000010) we end up with 0x02 (0b0000010).



```
void nibbleToPins(uint8_t in, uint8_t pin3, uint8_t pin2, uint8_t pin1, uint8_t pin0)
```

```
digitalWrite(pin1, in & 0b00000010);
```

Finally, the resulting function is shown below. Notice that in the function there is one digialWrite() for each of the four pins we wish to write to. Each successive function puts a one in the bit position shifted to the left of the previous call. The result is that the low nibble of the “in” variable is presented on pins pin0 – pin3.

```
void nibbleToPins(uint8_t in, uint8_t pin3, uint8_t pin2, uint8_t pin1,
uint8_t pin0) {
  // use bitwise & operator to create logic condition that is
  // used to light bits as if in the same nibble.
  digitalWrite(pin0, in & 0b00000001);
  digitalWrite(pin1, in & 0b00000010);
  digitalWrite(pin2, in & 0b00000100);
  digitalWrite(pin3, in & 0b00001000);
}
```