UNIVERSITY OF ALASKA FAIRBANKS

COLLEGE OF ENGINEERING AND MINES

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

COURSE CODE	EE F102 F01 (CRN: 34544)		
COURSE NAME	INTRODUCTION TO ELECTRICAL AND COMPUTER ENGINEERING		
SEMESTER	SPRING	**YEAR**	2022
LABORATORY LOCATION	ELIF 331 (ELECTRONICS LAB)		
LAB SESSION DATE AND TIME	MONDAY 07 FEB 2022		
TYPE OF SUBMISSION	LABORATORY REPORT	**NUMBER OF SUBMISSION**	3
TITLE OF SUBMISSION	PROGRAMMING THE ARDUINO NANO		
METHOD OF SUBMISSION	ONLINE TO: maher.albadri@alaska.edu		
DUE DATE OF SUBMISSION	FRIDAY 18 FEB 2022	**DUE TIME OF SUBMISSION**	23:59

STUDENT NAME	

MAKE THIS FORM A "COVER PAGE" FOR YOUR REPORT SUBMISSION.

FOR THE TA USE ONLY

REMARKS:

PROGRAMMING THE ARDUINO NANO

Objective:
In this lab we will explore Computer Engineering through programming the Arduino Nano. Computer Engineering resides at the interface between Electrical Engineering and Computer Science. We will learn the basic structure for all code developed with the Arduino programming environment. We will develop two simple programs that will flash LED(s) based on the value of a timer and an analog input. We will explore how to monitor time varying signals with an oscilloscope.

Arduino Overview
The Arduino Nano is a Microcontroller board based on the ATmega328p Microcontroller. It has 22 digital input/output pins and 8 analog inputs. 8 of the digital and analog pins are shared. When the Arduino is connected to the computer the USB cable will fully power the Arduino Nano. When it is not connected to the computer, external power will need to be supplied. The basic pin outs for the Arduino Nano are shown in Figure 1. The schematic for the Arduino Nano is shown in Figure 2.

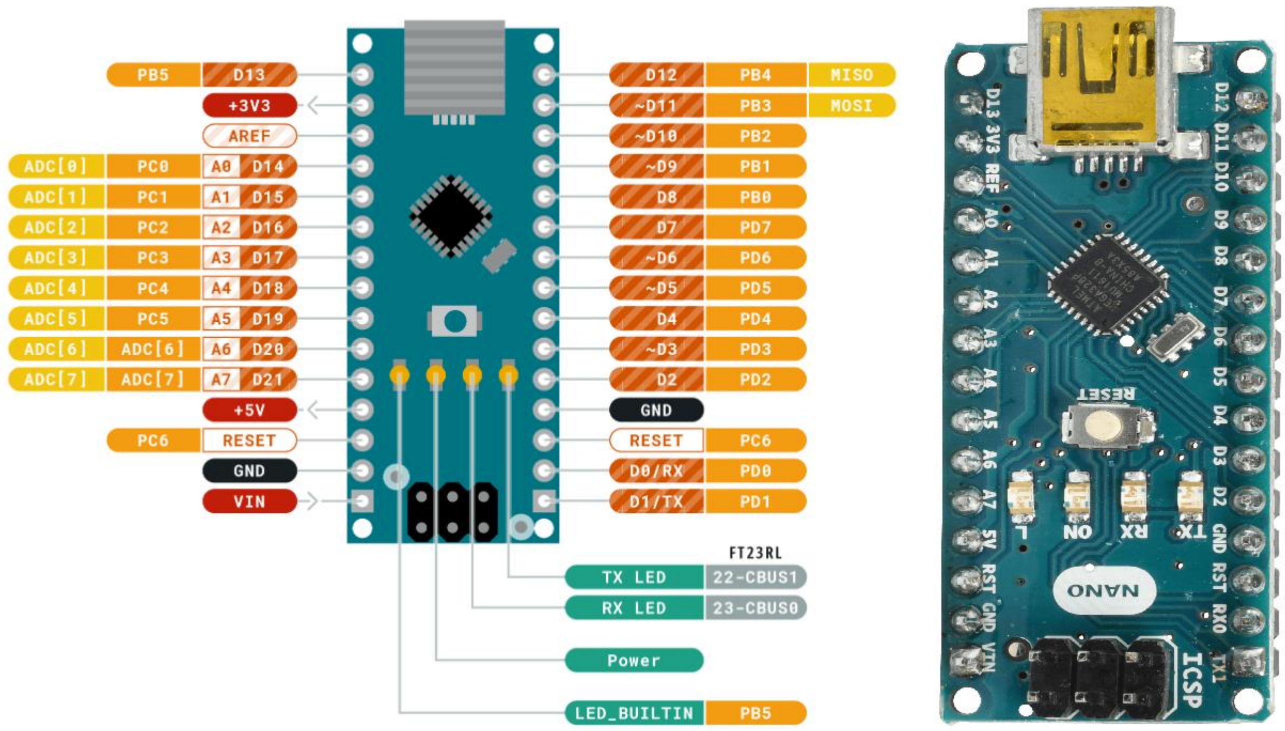


Figure 1. Arduino Nano pin out reference.

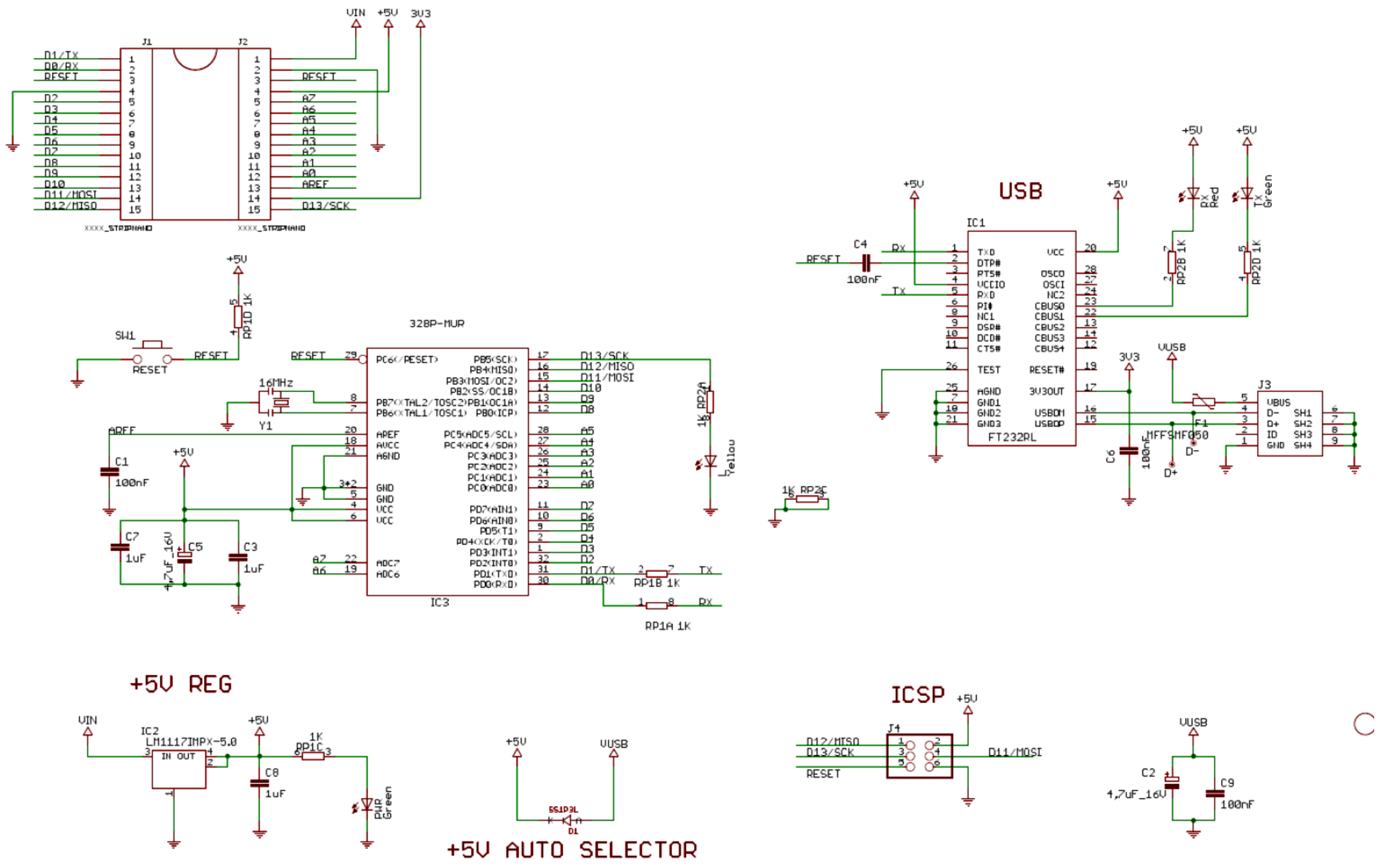
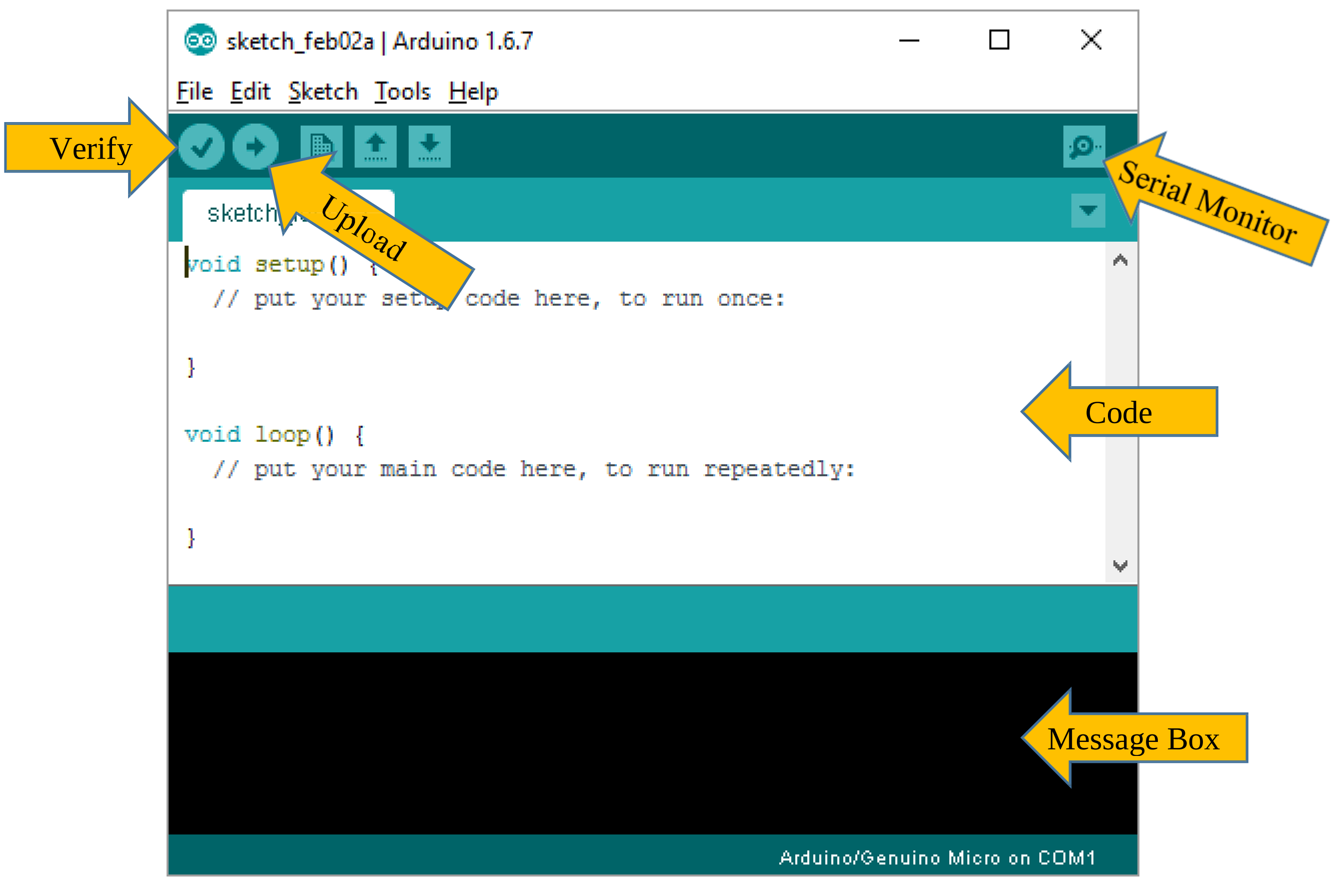


Figure 2. Arduino Nano Schematic.

Programming the Arduino Microcontroller

A Microcontroller is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. In order to program the Arduino, you first need to open the Arduino Integrated Development Environment (IDE), and then configure it to communicate with your specific Arduino. After establishing a connection you can upload programs to the Arduino. You can download the Arduino IDE to your own computer through this link: **Arduino IDE 1.8.19**. (***Note: do not plug in your Arduino Nano until instructed to!***)

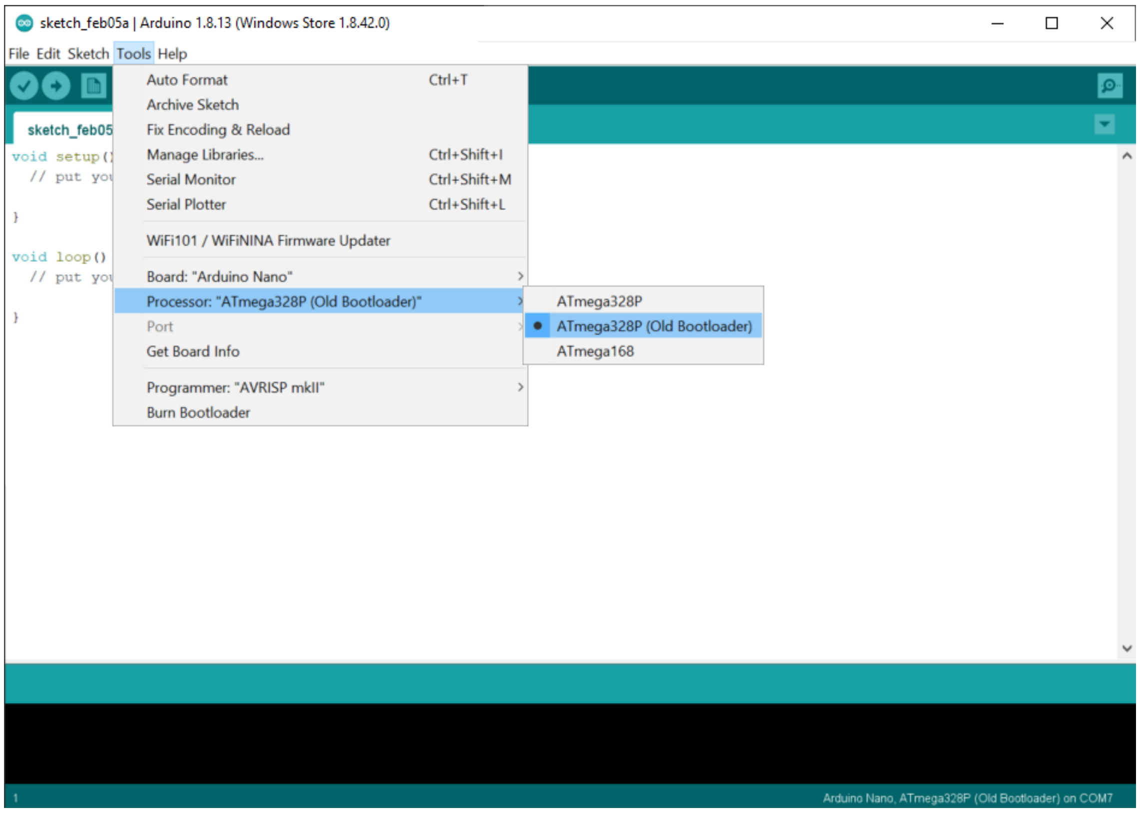
1 Search for arduino.exe. When you find the Arduino symbol, , click on it and the Arduino IDE will open. It should look something like what is shown in Figure 3. Arduino program files are called *Sketches*. When you start the Arduino IDE a new empty sketch appears. The Arduino IDE has an area where you type in code and a message box for the Arduino IDE to provide you with information. You should name the sketch you are writing and save it before compiling.


sketch_feb02a | Arduino 1.6.7
File Edit Sketch Tools Help
Verify
Upload
Serial Monitor
sketch
Code
Message Box
Arduino/Genuino Micro on COM1

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figure 3. Arduino IDE initial sketch.

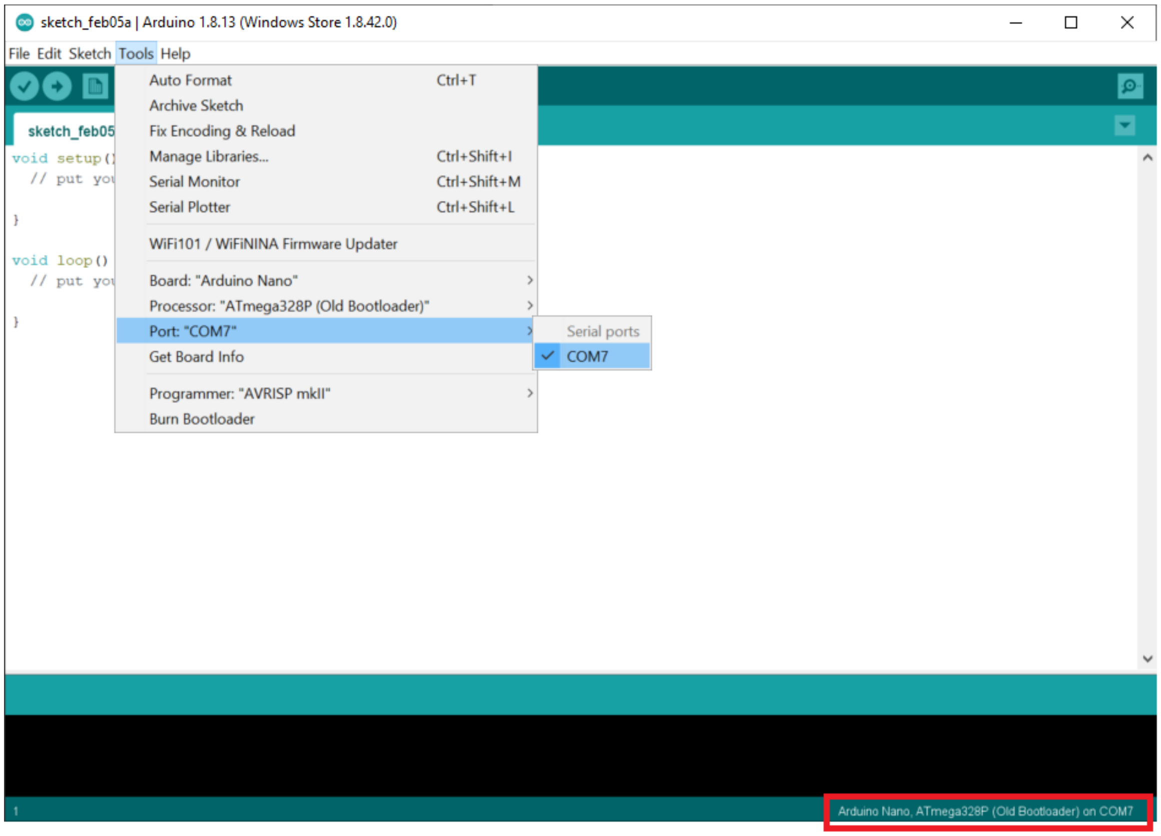
2 Configure the Arduino IDE by:
   (a) Configure the correct board. Select: *Tools->Board->Arduino Nano*
   (b) Configure the correct processor and the bootloader. Select: *Tools->Processor->ATmega328P (Old Bootloader)*. (see Figure 4).
   (c) Configure the correct serial port. The Arduino IDE is initially connected to an arbitrary COM port. Select *Tools->Port*: to view current COM ports. **Connect your Arduino Nano to the computer USB port. Select *Tools->Port->COMx* where x is the COM port that your Arduino is connected to. I have selected COM7. (see Figure 5).**


sketch_feb05a | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Nano"
Processor: "ATmega328P (Old Bootloader)"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
ATmega328P
ATmega328P (Old Bootloader)
ATmega168
sketch_feb05

```
void setup(
  // put yo

}

void loop()
  // put yo

}
```

Arduino Nano, ATmega328P (Old Bootloader) on COM7

Figure 4. Arduino IDE with correctly configured processor and bootloader.

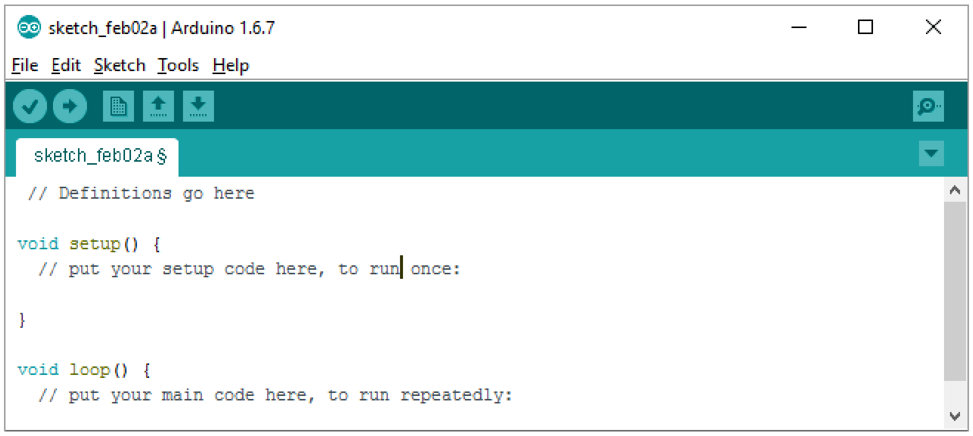

sketch_feb05a | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Nano"
Processor: "ATmega328P (Old Bootloader)"
Port: "COM7"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM7
sketch_feb05

```
void setup(
  // put yo

}

void loop()
  // put yo

}
```

Arduino Nano, ATmega328P (Old Bootloader) on COM7

Figure 5. Arduino IDE with Arduino Nano connected.

3 Examine the “Code” area of the Arduino sketch. There are three main sections of code in an Arduino sketch: *Definitions*, *Void Setup* and *Void Loop* (see Figure 6).

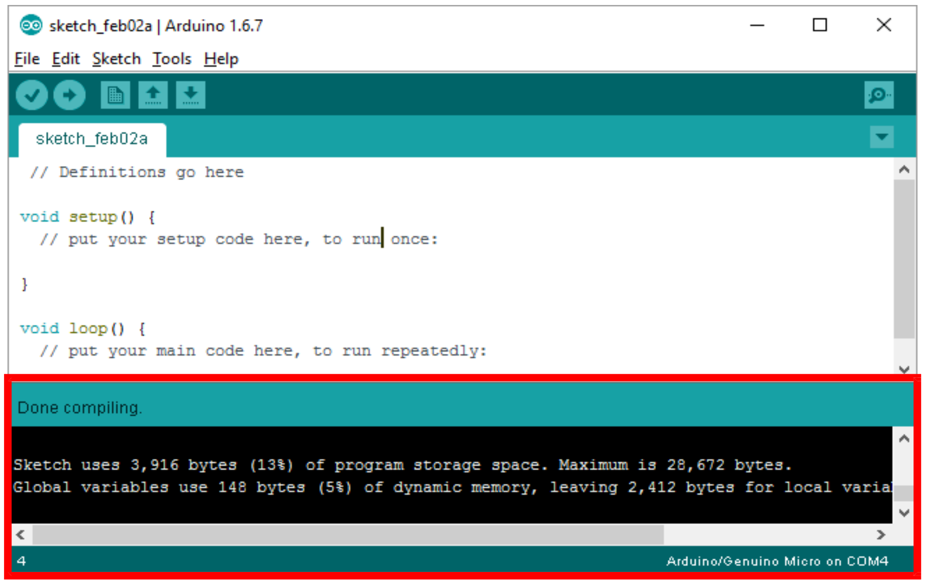


```
 // Definitions go here

void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:
```

Figure 6. Code section of Arduino IDE

(a) Definitions are declared **prior to void setup** and can include pin definitions, libraries to include in the sketch, functions, and global variables. Most programs declare something, however it is not always required.
(b) **void setup** is the first code block in a sketch. It is run only once. **void setup** is used to setup pin modes, communication initialization, and any code we only want to run one time.
(c) **void loop** is the second code block and it continuously repeats itself. This block is for code that needs to repeat such as sampling a sensor every couple of seconds. This is where the primary tasks of the code are carried out.

4 Compile code and check for messages. Even though this Sketch is not doing anything, it has all the necessary ingredients to be compiled and uploaded. Select the **Verify** icon to compile. You should see something like what is shown in Figure 7.


sketch_feb02a | Arduino 1.6.7
File Edit Sketch Tools Help
sketch_feb02a

```
 // Definitions go here

void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:
```

Done compiling.

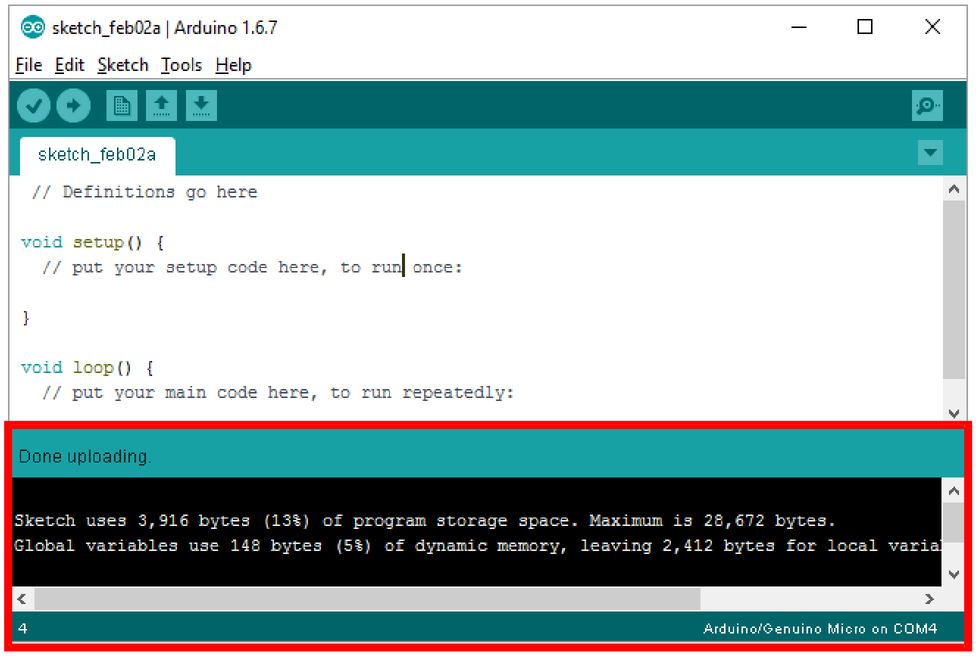
```
Sketch uses 3,916 bytes (13%) of program storage space. Maximum is 28,672 bytes.
Global variables use 148 bytes (5%) of dynamic memory, leaving 2,412 bytes for local varia
```

4
Arduino/Genuino Micro on COM4

Figure 7. Arduino sketch after compiling.

What is happening when you compile? The programming environment checks your code for syntax errors and returns error messages. It converts human-readable code into machine language (i.e., ones and zeros). When you tell the Arduino to upload, it first compiles then uploads (programs) your code (communicating between the computer and the Arduino).

5 Upload your code to the Arduino. Select the **Upload** icon to upload. If successfully uploaded (see Figure 8), you will know the computer can communicate with the Arduino!



```
// Definitions go here

void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:
```

Done uploading.

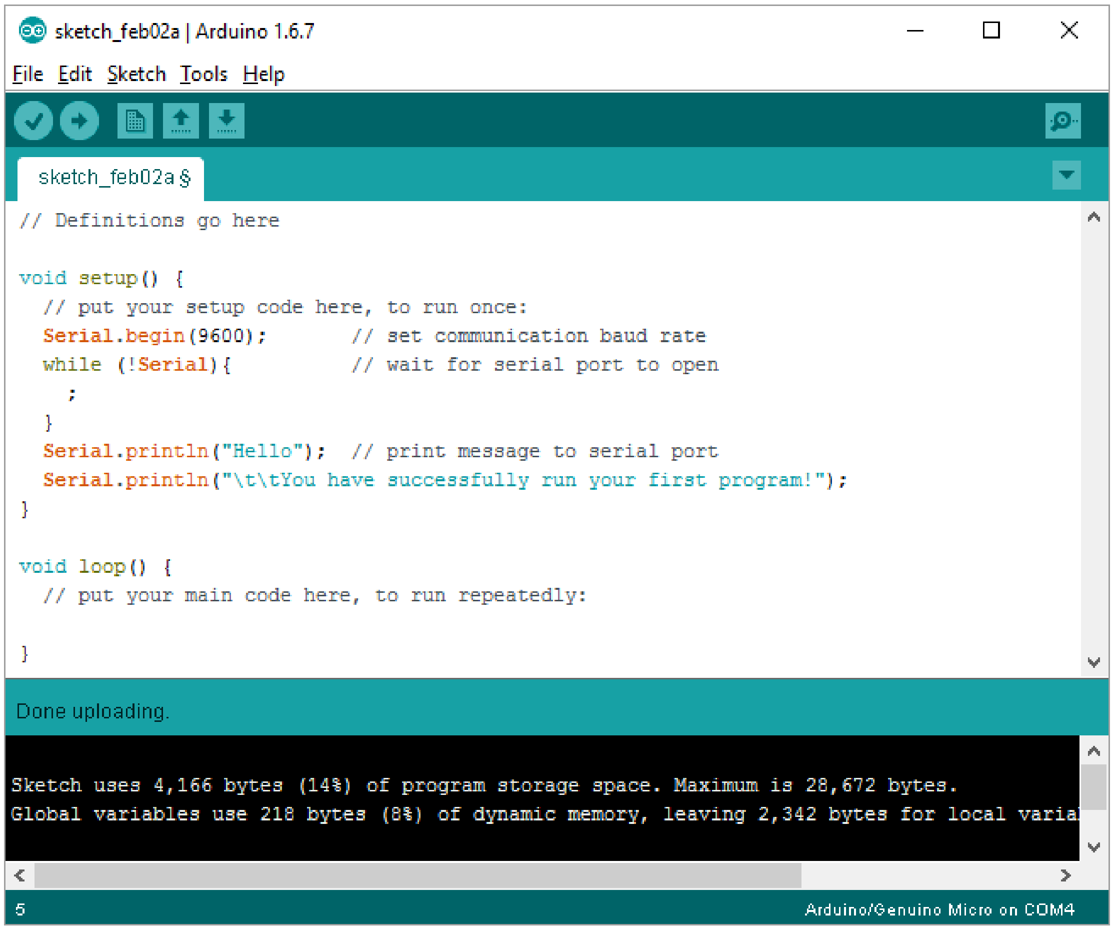
```
Sketch uses 3,916 bytes (13%) of program storage space. Maximum is 28,672 bytes.
Global variables use 148 bytes (5%) of dynamic memory, leaving 2,412 bytes for local varia
```

4 Arduino/Genuino Micro on COM4

Figure 8. Arduino sketch after uploading.

6 Code contains instructions that you want the microcontroller to perform. There are different programming languages in the same way that we have different spoken languages. We say “Hello” or “Bonjour”, Arduino says:
Serial.begin(9600);
Serial.print(“Hello”);
Arduino language is based on C/C++. Just like with spoken human languages, once you know one language, learning others is easy.

7 Modify the sketch to add the following to the **void setup()** (see Figure 9).



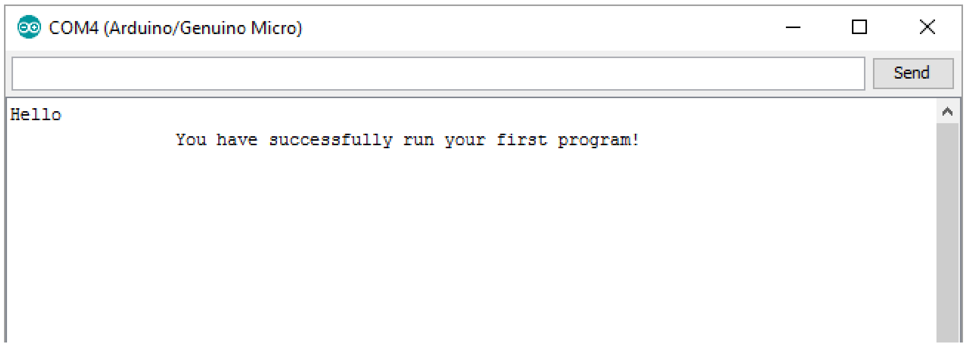
```
// Definitions go here

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);        // set communication baud rate
  while (!Serial){           // wait for serial port to open
    ;
  }
  Serial.println("Hello");  // print message to serial port
  Serial.println("\t\tYou have successfully run your first program!");
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

```
Done uploading.

Sketch uses 4,166 bytes (14%) of program storage space. Maximum is 28,672 bytes.
Global variables use 218 bytes (8%) of dynamic memory, leaving 2,342 bytes for local varia
```

Figure 9. Hello program

(a) **Serial.begin()** needs us to specify a communication rate (baud rate). We use 9600 bits per second, so put 9600 in the parentheses. **Serial.begin()** is in setup because this rate needs to be set only once.
(b) **while (!Serial){}** waits until you open the serial monitor to print.
(c) **Serial.print()** will just print to the monitor. **Serial.println()** will print to the monitor and then go to the next line (essentially pushes "return").

8 Compile the code and check for messages. Upload the code to the Arduino. Open the Serial Monitor._You should see the following on your serial monitor.



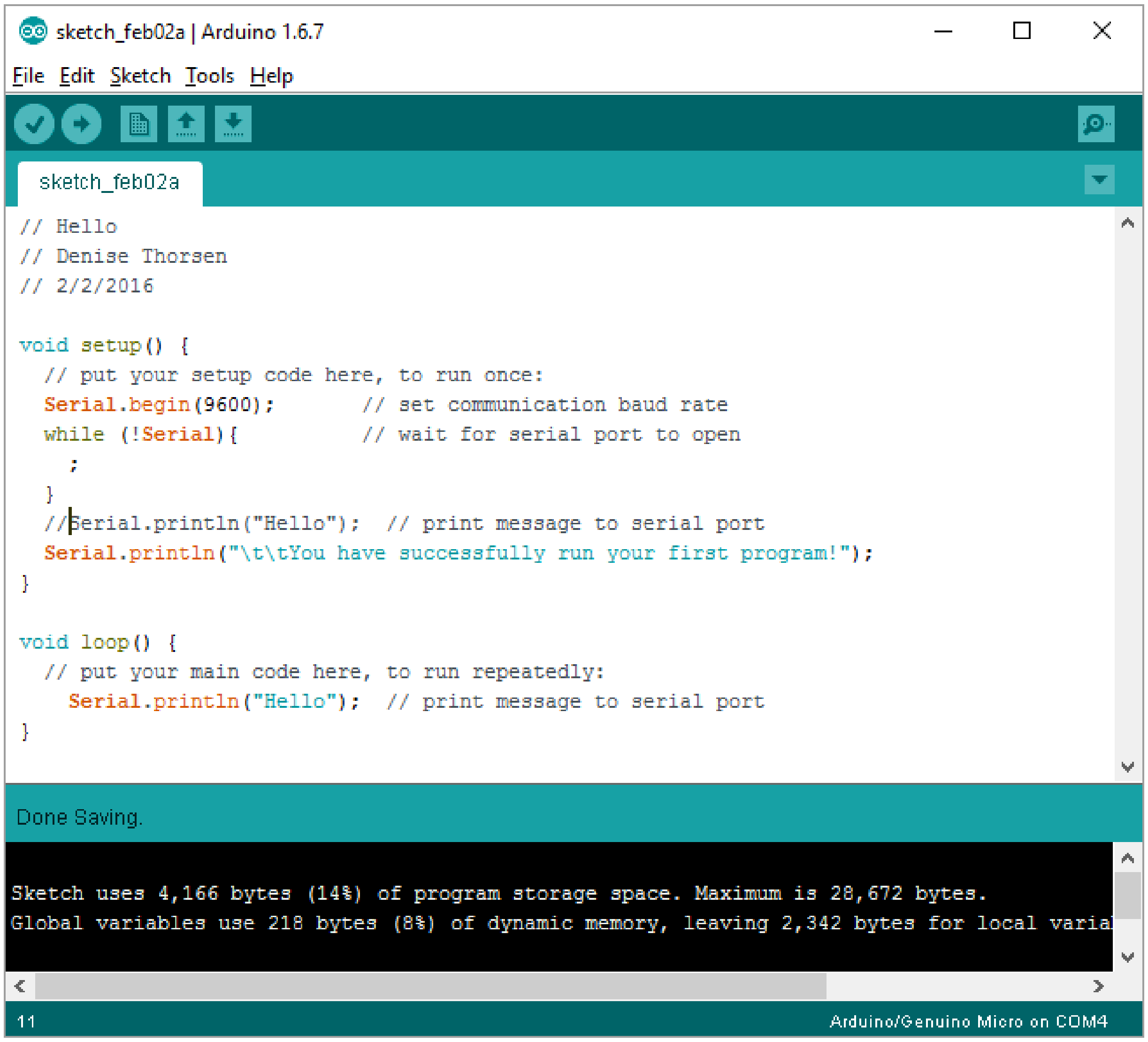
```
Hello
                You have successfully run your first program!
```

Figure 10. Output from the Hello program.

9 CONGRATULATIONS! You have now successfully programmed your Arduino – you are a computer programmer.

10 Comments about commenting. Arduino ignores comments but humans read them. Words become light gray if they are commented out. Put // in front of a line to comment out the whole line. To comment out an entire section, put /* at the beginning and */ at the end. Commenting is the MOST IMPORTANT THING!! It makes your code readable, provides context, helps draft what you want to do next and helps with debugging. **Make sure that all of your code starts with Comments that specify the purpose of the code, your name, and the date created.**

11 Try moving the **Serial.println("Hello")** command to the **void loop()** section (see Figure 11). What happens?


sketch_feb02a | Arduino 1.6.7
File Edit Sketch Tools Help
sketch_feb02a
Done Saving.
11
Arduino/Genuino Micro on COM4

```
// Hello
// Denise Thorsen
// 2/2/2016

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);       // set communication baud rate
  while (!Serial){          // wait for serial port to open
    ;
  }
  //Serial.println("Hello");  // print message to serial port
  Serial.println("\t\tYou have successfully run your first program!");
}

void loop() {
  // put your main code here, to run repeatedly:
    Serial.println("Hello");  // print message to serial port
}
```

```
Sketch uses 4,166 bytes (14%) of program storage space. Maximum is 28,672 bytes.
Global variables use 218 bytes (8%) of dynamic memory, leaving 2,342 bytes for local varia
```

Figure 11. Hello program with Serial.println("Hello") in void loop().

LED Timer Circuit

Now that we know how to program the Arduino, let us explore how we can control an LED. In this first example we will set up a timer or a clock and use that clock to turn on and off an LED. We will then examine at the timer signal on the oscilloscope and measure its frequency and duty cycle to see how those parameters can be controlled through the Arduino code.

1. Build the circuit shown in Figure 12. The USB connector supplies 5 V and ground to the Arduino. The 5 V and ground are internally connected on the board and are accessed through the appropriate pins. This schematic symbol for the Arduino labels the pin numbers starting at the upper left pin (by the reset button) as pin 1 and counting down the left side of the Arduino and up the right side. Therefore, the lower left most pin is pin 17, the lower right pin is pin 18, and the upper right pin is pin 34.

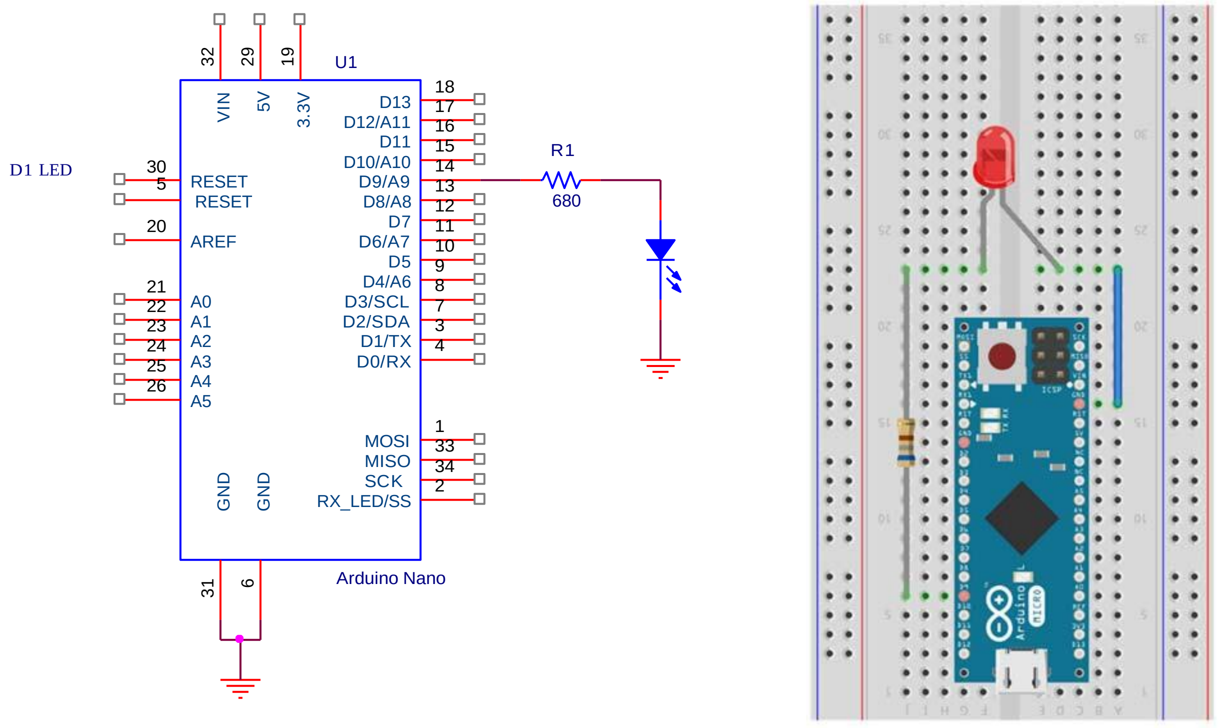
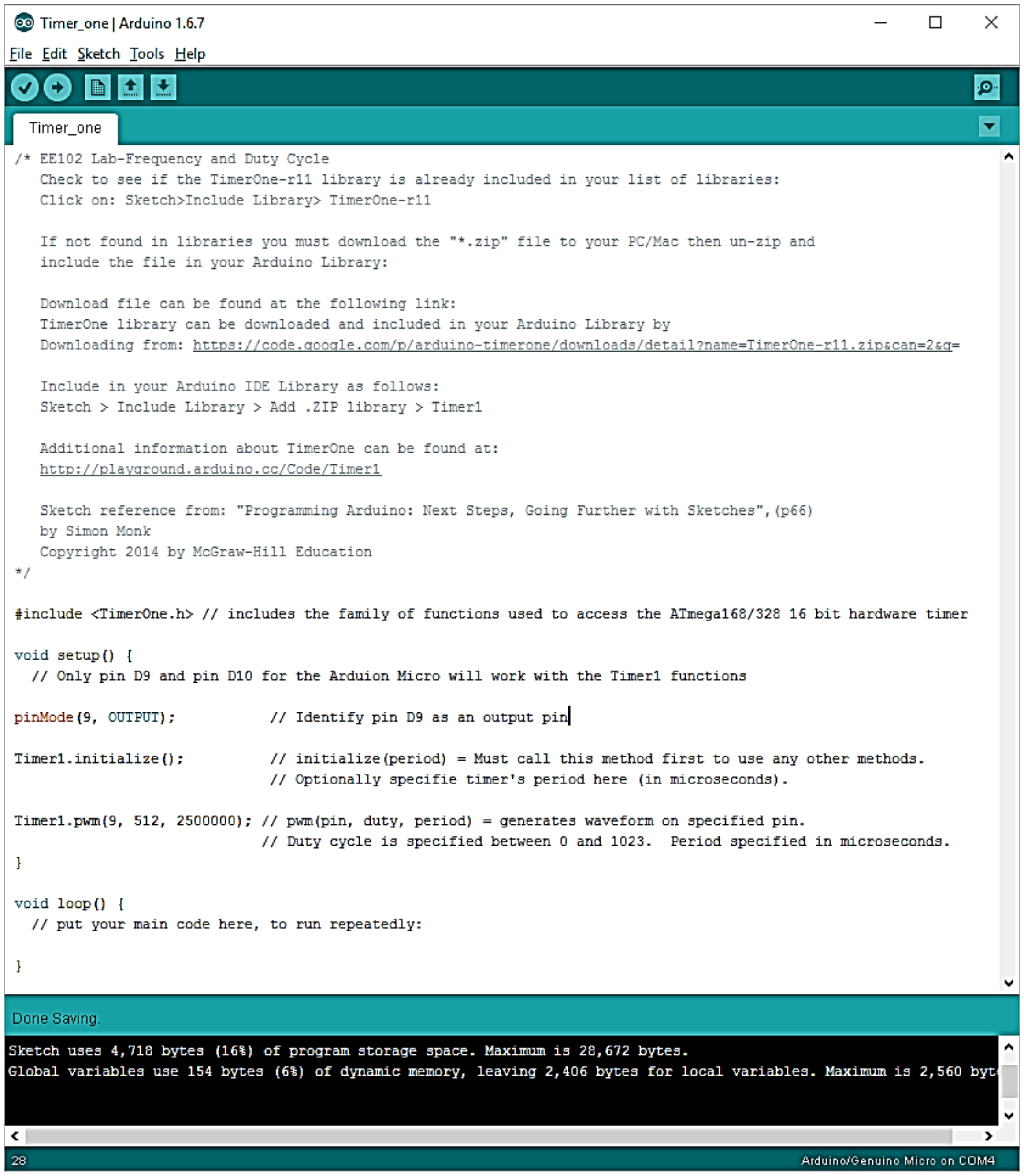


Figure 12. Schematic and breadboard layout for LED timer circuit

2. Download the **Timer_one.ino** sketch from Laboratory 3 page on CANVAS (*Arduino Library Files*) and open it in the Arduino IDE. It should look like Figure 13. This sketch requires the use of the TimerOne-r11 library. You should verify that this library is included in your list of libraries by selecting: ***Sketch -> Include Library -> TimerOne-r11*** If it is not found in the library list you must download the *.zip file and add the library by selecting: ***Sketch -> Include Library > Add .ZIP library -> Timer1***.



```
/* EE102 Lab-Frequency and Duty Cycle
   Check to see if the TimerOne-r11 library is already included in your list of libraries:
   Click on: Sketch>Include Library> TimerOne-r11

   If not found in libraries you must download the "*.zip" file to your PC/Mac then un-zip and
   include the file in your Arduino Library:

   Download file can be found at the following link:
   TimerOne library can be downloaded and included in your Arduino Library by
   Downloading from: https://code.google.com/p/arduino-timerone/downloads/detail?name=TimerOne-r11.zip&can=2&q=

   Include in your Arduino IDE Library as follows:
   Sketch > Include Library > Add .ZIP library > Timer1

   Additional information about TimerOne can be found at:
   http://playground.arduino.cc/Code/Timer1

   Sketch reference from: "Programming Arduino: Next Steps, Going Further with Sketches",(p66)
   by Simon Monk
   Copyright 2014 by McGraw-Hill Education
*/

#include <TimerOne.h> // includes the family of functions used to access the ATmega168/328 16 bit hardware timer

void setup() {
  // Only pin D9 and pin D10 for the Arduion Micro will work with the Timer1 functions

pinMode(9, OUTPUT);           // Identify pin D9 as an output pin

Timer1.initialize();          // initialize(period) = Must call this method first to use any other methods.
                              // Optionally specifie timer's period here (in microseconds).

Timer1.pwm(9, 512, 2500000); // pwm(pin, duty, period) = generates waveform on specified pin.
                             // Duty cycle is specified between 0 and 1023.  Period specified in microseconds.
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Done Saving.

```
Sketch uses 4,718 bytes (16%) of program storage space. Maximum is 28,672 bytes.
Global variables use 154 bytes (6%) of dynamic memory, leaving 2,406 bytes for local variables. Maximum is 2,560 byt
```

28 Arduino/Genuino Micro on COM4

Figure 13. Timer_one Sketch.

3. Compile and upload this sketch to your Arduino. If everything works you should see your LED flash periodically.
4. **Connect an oscilloscope to D9 of the Arduino at the resistor. Measure the frequency and duty cycle of the waveform. Modify the Timer_one sketch so that the frequency is 25 kHz with a duty cycle of 30%** (knowing that *frequency, in Hz = 1 / time, in seconds*)**. Capture an oscilloscope plot of the waveform showing your frequency and duty cycle measurements. Include the oscilloscope plot and your code in your lab report.**

LED Analog Input Control Circuit

The LED output signal may be controlled through the internal program or it may be controlled by an external circuit monitored by an analog input pin. In this first example we will monitor the external voltage, set by a potentiometer, on analog input pin 0 (A0), and use this value of voltage to turn on and off LED's.

1. Build the circuit shown in Figure 14. and Figure 15. The four LEDs are connected to Digital I/O pins 0, 1, 2, and 3.

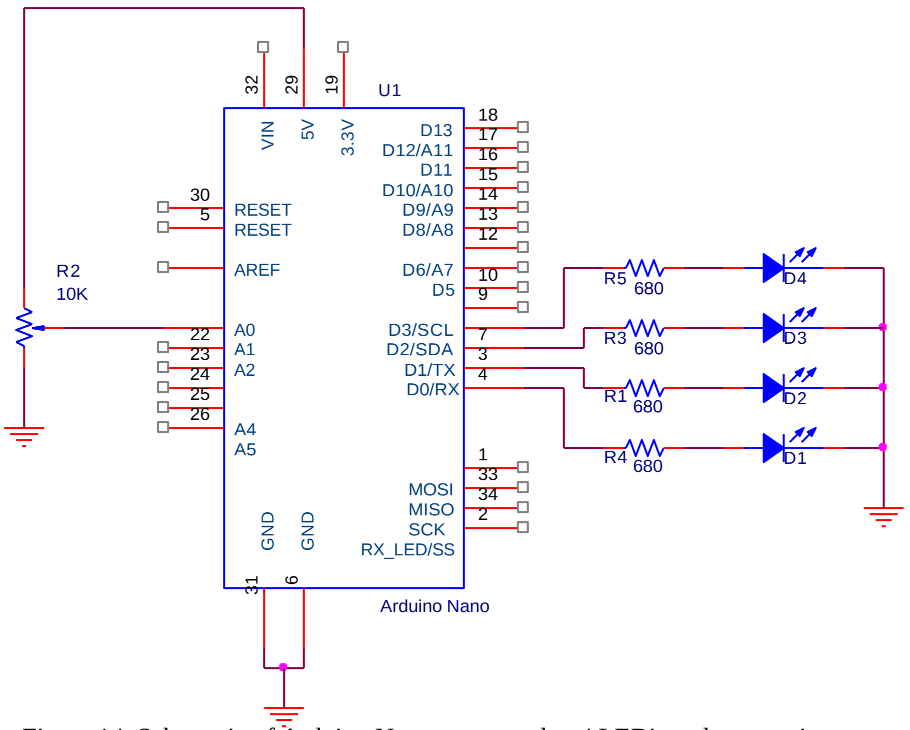


Figure 14. Schematic of Arduino Nano connected to 4 LED's and a potentiometer.

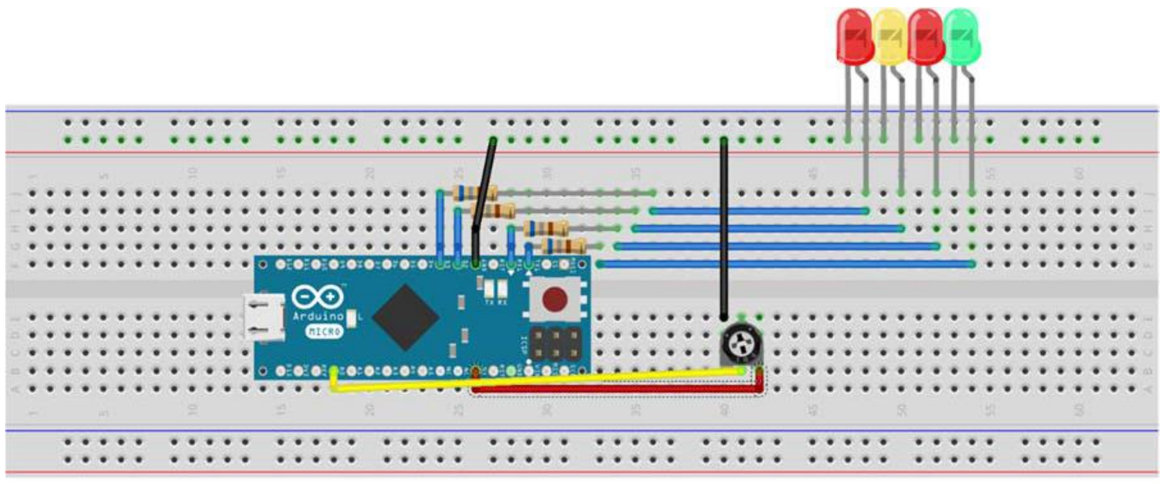


Figure 15. Breadboard of Arduino Nano connected to 4 LED's and a potentiometer.

2. Download the **LED4_Analog.ino** sketch from CANVAS and open it in the Arduino IDE. It should look like the following.

```
/*
 * EE F102 Introduction to Electrical and Computer Engineering
 * Spring 2022
 * Laboratory 3
 * PROGRAMMING THE ARDUINO NANO
 * LED4_Analog
 * Lights LEDs in response to Analog input voltage value
 */
 int led0 = 2;
 int led1 = 3;
 int led2 = 4;
 int led3 = 5;

 int sensorValue;
 float sensorVoltage;

void setup() {
  pinMode(led0, OUTPUT);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);

  Serial.begin(9600);
  while (!Serial) { };
  Serial.println("ready\n");
}

void loop() {
  sensorValue = analogRead(A0); //reads count (0 to 1023) on analog input A0
  sensorVoltage = sensorValue*(5.0/1023); //converts sensorValue count value to voltage value

  Serial.print(sensorValue);     //prints count value to Serial Monitor
  Serial.print("\t Voltage ");   //prints tab then Voltage to serial Monitor
  Serial.println(sensorVoltage); //prints voltage value to Serial Monitor

  if((sensorVoltage >= 1) and (sensorVoltage < 2)){         //Sets led 0 on if sensorVoltage greater
than 1 V.
    digitalWrite(led0,HIGH);
  }
  else {
    digitalWrite(led0,LOW);
    }
  if((sensorVoltage >= 2) and (sensorVoltage < 3)){         //Sets led 1 on if sensorVoltage greater
than 2 V.
    digitalWrite(led1, HIGH);
  }
  else {
    digitalWrite(led1,LOW);
    }
  if((sensorVoltage >= 3) and (sensorVoltage < 4)){         //Sets led 2 on if sensorVoltage greater
than 3 V.
    digitalWrite(led2, HIGH);
  }
  else {
    digitalWrite(led2,LOW);
    }
  if((sensorVoltage >= 4) and (sensorVoltage < 5)){         //Sets led 3 on if sensorVoltage greater
than 4 V.
    digitalWrite(led3, HIGH);
  }
  else {
    digitalWrite(led3,LOW);
    }
  delay(100);
  //digitalWrite(led0,LOW);
  //digitalWrite(led1,LOW);
  //digitalWrite(led2,LOW);
  //digitalWrite(led3,LOW);
}
```

(a) In the definitions area above **void setup()**, we need to declare our variables. Here we define the variables for our LEDs. For example, we are telling the Arduino to let a variable called **led2** represent an integer with the value of 2. Any time **led2** is used in the code, the Arduino sees 2. For example, 12 + **led2** = 14. We also need to declare a variable, **sensorValue**, which will contain the analog "count" registered on A0, and **sensorVoltage**, which will contain the actual voltage that corresponds to the count. Note that **sensorVoltage** is declared as a float, or a floating point data type. There are many more data types.

(b) In **void setup()** we assign our LED pins as outputs using the **pinMode(pin, mode)** function. "pin" refers to a specific digital I/O pin on the Arduino (in our case pins D0, D1, D2, and D3). "mode" is either INPUT or OUTPUT. OUTPUT sets up the pin so it can give outputs. INPUT sets up the pin so it can receive inputs.

(c) In **void loop()** we define when we want our LED's to turn on and off. **digitalWrite(pin, value)** sets the value of a specific pin. The value can be HIGH (5 V) or LOW (0 V). **delay(time)** tells the Arduino to wait a specific amount of time (in milliseconds) before going to the next line of code.

(d) In **void loop()** we need to read the analog signal on A0: **sensorValue = analogRead(A0)**. The analog inputs of the Arduino Nano have a 10-bin conversion or 2^{10} (0 to 1023) possible values. The actual voltage that must change in order to change the **sensorValue** is

$$\frac{\text{P1}}{\left(2^{10}\text{-}1\right)} \times 5\text{ V}=\frac{5\text{ V}}{1023} = 0.00488\text{ V} \qquad (1)$$

So the conversion from decimal count to voltage is

$$0.00488\text{ V} \times \text{count} = \text{voltage} \qquad (2)$$

(e) Finally, the **if** statements examine the **sensorVoltage** and turns on the correct LEDs based on the value read at A0.

3. Compile and upload the code to the Arduino. Is it working as you expected?
4. **Modify the LED_Analog sketch so that only one LED is lit at a time. Include the revised code in your lab report.**

In your lab report:

Your lab report should contain the following sections:

(a) <u>Title Page: </u>See previous lab reports for content.

(b) <u>Objective: </u>Include a statement of the purpose of the lab.

(c) <u>LED Timer Ci</u>rcuit: Provide your modified code that sets the frequency to 25 kHz with a duty cycle of 30%. Include your oscilloscope plot of the waveform showing your frequency and duty cycle measurements. Discuss how the frequency and duty cycle are set, specifically with respect to how your code works.

(d) <u>LED Analog Input Control C</u>ircuit: Provide your modified code that lights only one LED at a time while the potentiometer is varied.

(e) Conclusion: Discuss how you might use the test sketches examined in this lab to create a program that monitors the thermistor resistance and determines the current temperature. You don't need to have a complete solution for Lab 3, only provide a descriptive approach for how you might create a temperature sensor using the thermistor and Arduino. You will create the detailed design in Lab 4.