GRAND VALLEY STATE UNIVERSITY
SCHOOL OF ENGINEERING

EGR 346 Mechatronic Systems Dynamics and Control

Lab Experiment # 1

Microcontroller setup

Prof. Nael Barakat

Last Revised Aug. 22, 2022

1.Lab 1 - Programming the Atmel ATmega324PMicrocontroller and using the Arduino Board

1.1. PURPOSE

To program and use the Atmel Atmega328P microcontroller (on Arduino UNO) for data exchange and system control, in preparation for its utilization in control systems.

1.2. OBJECTIVES

- To program the microcontroller onboard the Arduino for performing specific tasks using A PC software environment.
- To use the Arduino in the exchange of digital and analog data.

1.3. BACKGROUND and THEORY

A microcontroller is a reprogrammable device capable of performing complex tasks and data manipulation and exchange. Atmel ATmega328P is a microcontroller that has been installed on a board named Arduino UNO. This is the board used for this course. The following URL contains useful information about the board and the microcontroller:
http://arduino.cc/en/Main/ArduinoBoardUno or **https://store.arduino.cc/usa/arduino-uno-rev3**
A photograph of the top side of the Arduino board is provided in Figure 1.1. Hardware description of the Arduino board has been provided in EGR 107 and the above link. Some important information is summarized in the next sections.

1.3.1. HARDWARE DESCRIPTION

A. Input Power

The Arduino Uno can be powered using the USB connection or with an external power supply, such as the laboratory bench supply. The power source is selected automatically. The USB port provides a 5V DC power supply at up to 500mA of current. External (non-USB) power can come either from an AC-to-DC adapter, battery, or bench power supply. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the GND and VIN pin headers of the POWER connector. Note that the *silkscreen* (the white lettering) on the top of the Arduino Uno board indicates which connector pin is used for this purpose. Also, the ground (black) terminal of the power supply must be connected to the GND power connector terminal when externally powered.
The VIN pin header provides input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or voltage applied at the DC power jack). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is therefore 7 to 12 volts.

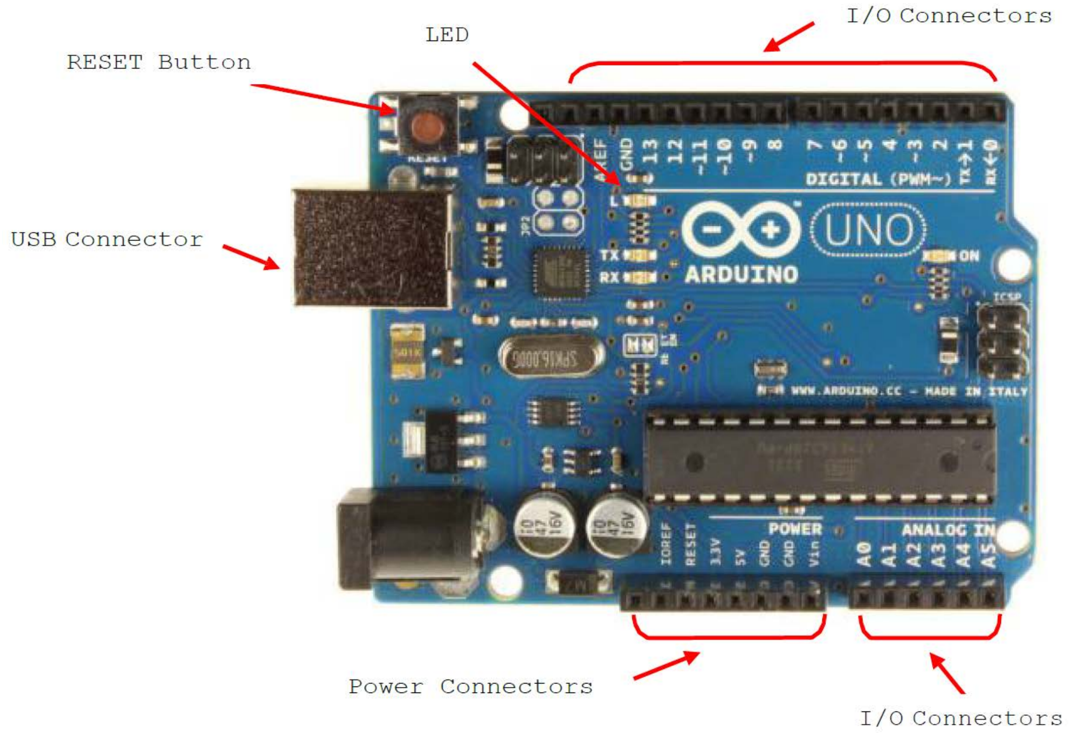


Figure 1.1 Photograph of the top of the Arduino Uno board.

B. Output Power
Regardless of where the power input to the Arduino Uno board comes from, a 5Voutput is available at the power connector pin labeled “5V” on the silkscreen. This pin can be used to power other devices, such as 74XX logic devices, a logic probe, etc. As before, the GND power connector terminal must be connected to the ground reference terminal of whatever is being powered.
Along with 5V output, a 3.3V output is also provided by the on-board regulator at the power connector pin labeled “3.3V” on the silkscreen. A small amount of current (50 mA) can be drawn from this 3.3V output.

C. I/O Ports
There are 3 general-purpose I/O ports on the ATmega328P microcontroller, named Port B through Port D. Port C is an analog Port and it has six pins in total. Port B and Port D are digital ports and have 7 pins each.The digital I/O pins operate at 5 volts and each pin can provide or receive a

maximum of 240 mA current, though total microcontroller current, over all pins, should be limited to 100mA. In addition, some pins have specialized functions as listed below:

1. **Serial: pin ‘0’ -> RX and pin ‘1’ -> TX**
Used to receive (RX) and transmit (TX) serial data. These pins are connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.

2. External Interrupts: pin ‘2’ and pin ‘3’
These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.

3. PWM: pins ‘3’, ‘5’, ‘6’, ‘9’, ‘10’ and ‘11’
These pins provide 8-bit PWM output.

4. SPI: pins ‘10’, ‘11’, ‘12’ and ‘13’
These pins support SPI communication. (a method of interconnecting digital components).

5. LED: pin ‘13’
There is a built-in LED (see Figure 1.1) connected to digital pin 13. When the pin is HIGH, the LED is ON, when the pin is LOW, the LED is OFF. A “logic high” on an I/O port will measure as approximately 5V, while a “logic low” will be approximately 0V.

1.3.2. CONVERSION FROM ANALOG TO DIGITAL SIGNALS

An analog to digital (A/D or ADC) converter converts an analog input voltage to a digital value. A block diagram of a successive approximation A/D converter is shown in figure 1.2. The main operation concept is based on the successive approximation logic. Once the reset is toggled the converter will start by setting the most significant bit of the 8 bit number. This will be converted to a voltage Ve that is a function of the +/-Vref values. The value of Ve is compared to Vin and a simple logic check determines which is larger. If the value of Ve is larger the bit is turned off. The logic then repeats similar steps from the most to least significant bits. Once the last bit has been set on/off and checked the conversion will be complete, and a done bit can be set to indicate a valid conversion value.

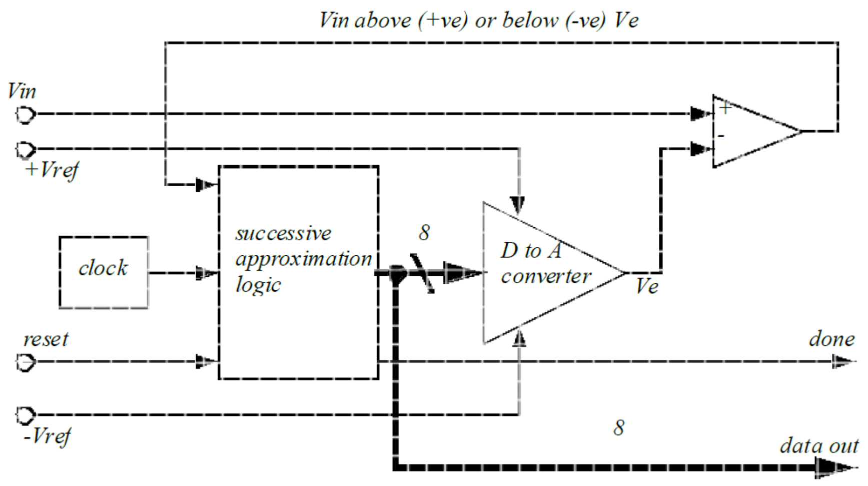

Vin above (+ve) or below (-ve) Ve
Vin
+Vref
clock
successive approximation logic
8
D to A converter
Ve
reset
done
-Vref
8
data out

Figure 1.2 A block diagram of a typical Analog to Digital converter.

In the ATMega, the A/D converter is 10 bit, and there are up to eight 0-5V inputs available. Once the analog input value has been read it can be converted back into a voltage using the following equation:

$$V_c = \left(\frac{V_I}{R-1}\right)(V_{max} - V_{min}) + V_{min}$$

where

R = resolution of the ADC , $R = 2^N$, N = number of bits

V_I = the integer value from the ADC

V_C = the voltage calculated from the integer value

INTRODUCTION TO SPEED CONTROL

Controlling electro-mechanical devices (such as motors) can be achieved through dedicated circuitry that transforms one form of input (a knob, a switch, etc.) to the necessary electrical signals. Dedicated circuitry, however, is inflexible (it can only do what it was built to do) and subject to variations in temperature and component aging. A more flexible control approach is to use programmable digital hardware (e.g., a microcontroller) to affect the necessary signals in response to some input. The benefit of this approach is flexibility and repeatability. The control policy can be easily changed simply by running a different program, without hardware changes, and the digital circuitry is much less sensitive to issues such as temperature and component aging.

Applications

Unlike a stepper motor, a simple brush DC motor will turn, and keep turning, when a voltage is applied to its 2 terminals. The higher the voltage, the faster it turns (within limits, of course). The speed of a DC motor may be easily varied by varying its applied voltage. Problems arise, however, when we try to vary the speed of a DC motor with this approach when all we have is a constant-voltage DC supply (like a battery). If the supply provides a potential of 42V and we want a motor speed that requires only 12V applied to the motor, what do we do? We can dissipate the excess voltage in a resistor or transistor as heat, but that is very wasteful and leads to problems in dissipating that heat.

A similar application is to control the brightness of an LED by varying the current that is applied to the LED. One can vary this current by using a potentiometer (variable resistor); the more current that flows through the LED the brighter it is, lower current makes it dimmer. Just like with the motor, power is dissipated as heat in the resistor, which is wasteful.

A better approach is to use the principle of PWM, which is an acronym for pulse-width modulation. By alternately applying and not applying the full DC voltage, we can affect varying amounts of average power. The motor's speed or the LED's brightness will be proportional to this average power. The same idea can be applied to other devices that convert electrical energy to some other form of energy. For example, microwave ovens use PWM to implement their varying power levels; conventional ovens use PWM to maintain a constant temperature, and so on.

1.3.3. PULSE WIDTH MODULATION (PWM)

Pulse Width Modulation (PWM) is a method for controlling analog output using the duty cycle of a digital pulse output. The method is demonstrated by Figure 1.3. If the output is on all the time, the effective output voltage is the maximum voltage of the output. If the output is only on half the time, the effective output voltage is only half the maximum level of the pulse. By varying the ratio of on-time to off-time, the effective voltage is varied. The percentage of time that the signal is on compared to a total cycle time is called the duty cycle. So, if the voltage is only on half the time, the effective voltage is half the maximum voltage and the duty cycle is 50%. This method is popular because it can produce a variable effective voltage efficiently. It turns out, to a first approximation, the average electrical power delivered to the motor or LED (hence its speed or intensity) is linearly related to duty cycle. Thus, by increasing or decreasing duty cycle, we can speed up or slow down a motor or brighten or dim an LED.

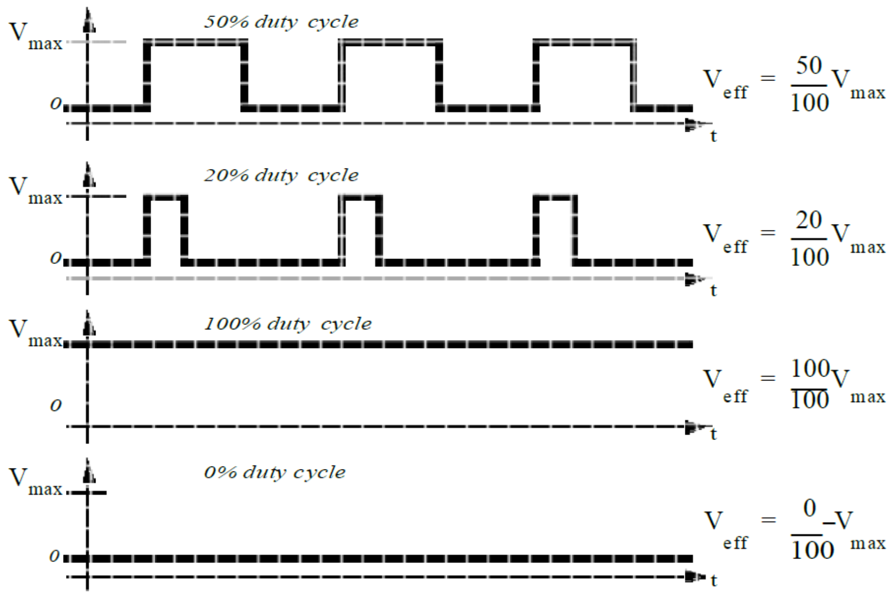

50% duty cycle
V_{max}
0
t
$V_{eff} = \frac{50}{100} V_{max}$
20% duty cycle
$V_{eff} = \frac{20}{100} V_{max}$
100% duty cycle
$V_{eff} = \frac{100}{100} V_{max}$
0% duty cycle
$V_{eff} = \frac{0}{100} V_{max}$

Figure 1.3 Demonstration of the concept of Pulse Width Modulation (PWM).

1.3.4. SERIAL COMMUNICATIONS

Serial communication typically involves sending bytes between devices. These bytes are often ASCII encoded characters. Multiple bytes make up a string. For this lab's purposes, serial communication will be the method used to send commands to the microcontrollers, and to get responses (including data) back. Serial communications can be done using multiple ways. The simplest method will be over the USB port, using a virtual serial port under windows. Then, a terminal emulator program such as Hyperterminal or the one available in the ninja.exe program interface might be used to interact with the microcontroller.

The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).The Arduino hardware has built-in support for serial communication on pins 0 and 1 (which also goes to the computer via the USB connection). The native serial support happens via a piece of hardware (built into the chip) called a UART (Universal Asynchronous Receiver / Transmitter). This hardware, along with the Serial Arduino library, allows the ATmega328 chip to receive serial communication even while working on other tasks, as long as there room in a 64 byte serial buffer.

In addition to the dedicated hardware functions of the Serial library using pins 0 and 1, the Software Serial library has been developed to allow serial communication on other digital pins of the Arduino, using software to replicate the functionality (hence the name "SoftwareSerial"). For your ATmega328P C program, the following functions are provided by the Serial library:

• Serial.begin (speed)
This function sets the speed (baud rate) for the serial communication. Supported baud rates are 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 38400, 57600, and 115200. A sample function call is shown below:

```
// Opens serial port and sets the data rate to 57600 bits-per-second
   Serial.begin(57600);
```

Serial.println (data)
This function prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n'). The syntax for this function is:

```
Serial.println(val)
Serial.println(val, format)
```

where: val represents the value to print - any data type format: specifies the number base (for integral data types) or number of decimal places (for floating point types). A sample function call is shown below:

```
Serial.println("--- This program will print what you entered ---");
```

• Serial.available ()
This function returns the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer (which holds 64 bytes). A sample function call is shown below:

```
// print message only when you receive data:
if (Serial.available() > 0)
Serial.println(“I received something”);
```

Serial.read ()
This function reads the incoming serial data. This function returns the first byte of incoming serial data available (or -1 if no data is available). The return type is int. A sample function call is shown below:

```
char incomingByte = 0; // for incoming serial data
// read the incoming byte:
incomingByte = Serial.read();
```

1.4 PRE-LAB

1. Review C programming using previous course materials.

2. Review the EGR 107/EGR 226 (or equivalent) course materials.
3. Review the Arduino material (https://www.arduino.cc/en/Guide/HomePage) to become familiar with the microcontroller.
4. Review the fundamentals of ASCII strings.

1.5 REQUIRED SOFTWARE

Note: You can install the tools for free for your own use by going to the following website.
http://arduino.cc/en/Main/Software
Software development for the Arduino boards requires a combination of tools. In this laboratory, you will refresh you familiarity with these tools while developing simple programs.

A. The Arduino IDE

The Arduino IDE is an *integrated development environment* (IDE) that pulls together many tools and makes it easier to use them as a cohesive unit (much like Xilinx ISE did with schematic capture, compilation, simulation, and programming tools). The tools provided by Arduino IDE include:

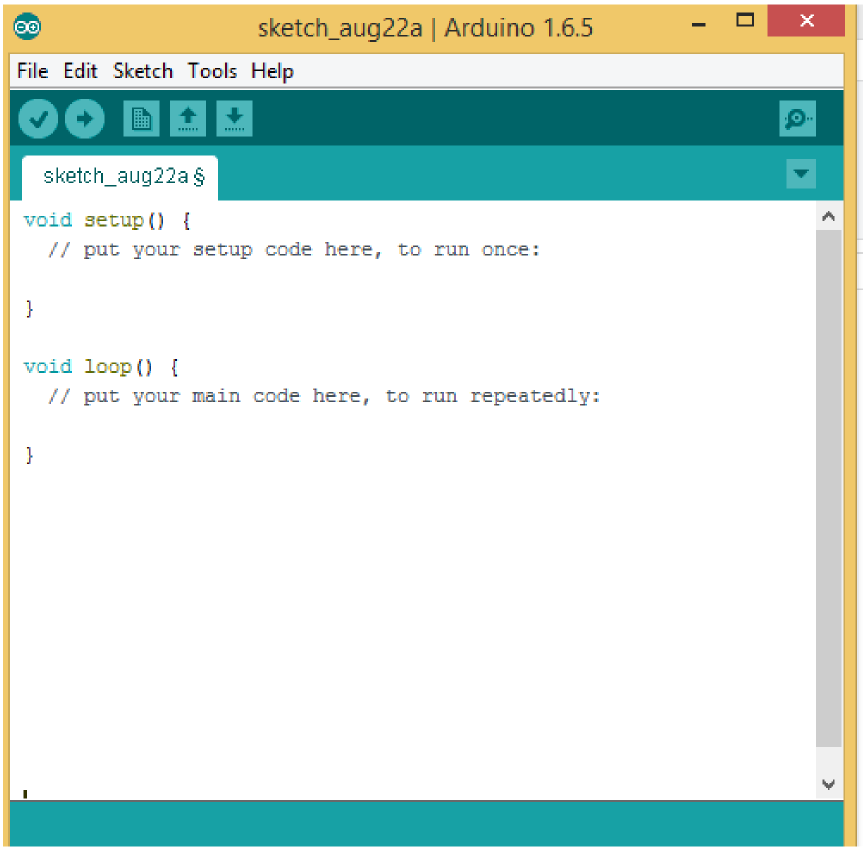
- ➢ A text editor with syntax highlighting, brace matching and automatic indentation (likeGVim)
- ➢ A plug-in that calls the AVR-GCC compiler so that you can compile your program from within the IDE
- ➢ A plug-in that calls AVR DUDE to upload programs to the board with a single click.

1.6. EQUIPMENT

1. Computer with an Internet connection
2. Assembled controller board (Arduino)
3. Breadboard
4. 8 LED‟s.
5. Digital Multi Meter (or any multi meter).
6. Oscilloscope.
7. 5 V Power supply.

1.7. EXPERIMENTALPROCEDURE

1. ***If you are familiar with activating the Arduino and able to download programs to it, then you can skip the steps from 2 to 12 and start at step 13.***
2. Start **Arduino** by going to **Start -> Programs -> Arduino -> arduino.** Once the application starts, select the **File -> New** menu item. A new project (also referred to as ***sketch***) is created as shown in Figure 1.4 below.



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

}

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

}
```

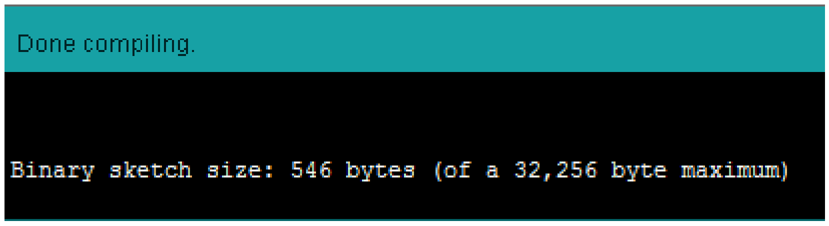
Figure 1.4: The New Project Dialog in Arduino

3. Now, select **File -> Save** to open “**Save sketch folder as**” dialog box. Make sure you are pointing to **C:/Temp** folder (create this directory on your own computer if it doesn’t already exist) and set the **File name** text box to be „**Labtut**“ and click on the **Save** button.
4. After the project is created, you will see the **Labtut** application file (Labtut.ino) that will be the main source file for your project.
5. Type the following code into the editor and save the file.

This example below shows the simplest thing you can do with an Arduino to see physical output: it blinks an LED.

```
/*************************************************************************
  * This program turns on an LED on for one second, then off for one second, repeatedly.
 *************************************************************************

 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup()
{                pinMode(led, OUTPUT);      // initialize the digital pin as an output.

}

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

6. Click on **Sketch -> Verify/Compile** or ✓ icon to compile and build your application.
7. If compiled successfully, you should see a message in the console window as shown below (Figure 1.5).



```
Done compiling.

Binary sketch size: 546 bytes (of a 32,256 byte maximum)
```

Figure 1.5: Successful compilation message

8. Now that you have successfully compiled your program, it is time to write it to the FLASH memory of the ATmega328P microcontroller. Plug in your Arduino board into a free USB port. Wait until all of the Windows messages proceed to indicate the board is ready to use. If you are prompted to install drivers, select the recommended automatic driver installation.

NOTE: If you run into problems with the drivers for this board on your personal computer, you may point to Arduino installation directory to look for the drivers.

These drivers have already been installed **BUT WILL NOT BE ACTIVE UNTIL YOU PLUG IN YOUR BOARD!** It is important, therefore, that you plug in your board before trying to program the FLASH.
9. Going back to the Arduino application, select **Tools -> Board -> Arduino Uno** to select the intended board. Then select **Tools -> Serial Port -> (The COM Port number).**

To the PC, the Arduino board appears as a virtual serial port. That is, whereas the PC may have an actual hardware serial port (e.g., COM1) which is accessible from a hardware connector, the Arduino board appears as, for example, COM3.
To find the COM Port number that has been assigned to your Arduino board, you can go to Control Panel -> System -> Device Manager and under Ports section you should be able to see the Port number assigned to your board.

10. After the selection process is complete, click on File -> Upload or ➔ icon to upload your program to the microcontroller.
11. After a few seconds, the yellow **LED** on the top side of the board will begin to blink at a very fast rate. Your program is now running. Your program is now "permanent" in that it will always execute after you press the reset button or apply power to the
board. The program can be changed, of course, by uploading a new program using the steps described above.
12. To convince yourself that this is true, try unplugging the Arduino board from the computer, then plugging it back in.

-- The Required Part of the lab ------------------------------------

13. Repeat the above program, for blinking an LED by connecting an LED on digital pin 4. Use a resistor (anything between 220 ohm to 1K ohm) between LED and the digital pin.

Analog Read

A **potentiometer** is a device that provides a varying amount of resistance when its shaft is turned. By passing voltage through a potentiometer and into an analog input on your Arduino, it is possible to measure the amount of resistance produced by a potentiometer as an analog value. The voltage that you read as an input is the **analog voltage**.

The Arduino has a circuit inside called an **analog-to-digital converter** that reads this changing voltage and converts it to a number between 0 and 1023. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and the input value is 0. When the shaft is turned all the way in the opposite direction, there are 5 volts going to the pin and the input value is 1023.

In the setup function, you need to begin serial communications, at 9600 bits of data per second, between your Arduino and your computer with the command: `Serial.begin(9600)`
Serial.println() is used to print information on to serial monitor.

14. Write a program to read an analog input using a potentiometer and print it on your serial window as a voltage between 0 to 5 v.

15. Write a program to the Arduino board to accept an ASCII character from the keyboard and display it back on the screen.

16. Connect 8 LEDs, as suggested by figure 1.6, to output pins on the Arduino to formulate a binary counter. Write a program that will count in a binary sequence and output the values on the selected port. Connect the positive side of a multi meter to one of the outputs. Connect the negative side of the multi meter to one of the grounded terminals. For lower order bits the multi meter should switch values about once per second, higher order bits will switch more slowly. The LEDs circuit should help you see the sequential counting. Note that LEDs are direction sensitive and if connected in reverse will not light. Note also that you need a protection resistor between the LED and the pin of 220 Ohms value.

17. Show the results to your instructor.

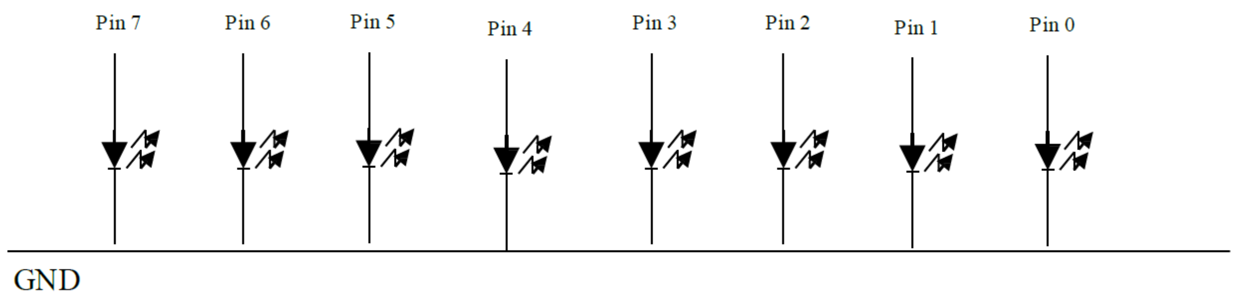


Figure 1.6: LED counting circuit to verify the programs on the Arduino board.

18. Modify the LEDs program (from step 16) to accept a number from the user throughthe PC keyboard, between 0 and 255, then display that using the LEDs in binary mode. Watch the communication through the monitoring tool set during step 15.

19. Show the results to your instructor.

1.8. POST-LAB

1. Report your observations and results of executing the experimental procedure above with insightful comments and discussions.

2. Demonstrate the functions of the programs you have written to your instructor.

3. Provide a listing of all the programs you have written and used in this lab with comments.