Introduction

This goal of this document is to detail how to configure and use a development environment for bare-metal development on the Arduino Nano for SOEN422. There will be several sections detailing configuration procedures for macOS, Linux and Windows.

Acquiring the Development Tools

Linux

Acquiring the toolchain for Linux is rather straight forward. The tools are typically readily available via package managers on most commonly used Linux distributions.

Debian Based Linux Distributions

For Debian based Linux distributions such as Debian and Ubuntu, run the following commands:

```
sudo apt-get update
sudo apt-get install avrdude gcc-avr
```

These commands will automatically install the tools necessary to compile and upload code to the Arduino.

RedHat Based Linux Distributions

For RedHat based distributions such as CentOS, Fedora and RHEL, run the following command:

```
sudo yum install avrdude gcc-avr
```

macOS

To set up the development environment on macOS, Homebrew is recommended. To install Homebrew, follow the instructions at `https://brew.sh`. Once Homebrew has been installed, run the following command in a Terminal:

```
brew install avrdude avr-gcc
```

Windows

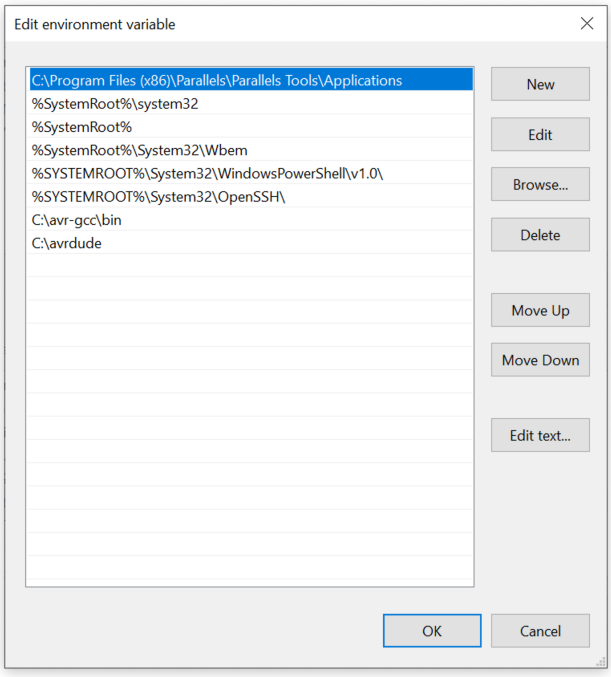
These instructions are written with Windows 10 in mind. The AVR compiler for Windows is readily available from Microchip (the manufacturer of the AVR microcontrollers) at

`https://www.microchip.com/mymicrochip/filehandler.aspx?ddocname=en607654`

The file downloaded will be a zip file. The file must be extracted somewhere where it will not be modified. For this example, it has been extracted to `C:\avr-gcc`

As well, AVRDUDE must be downloaded as well and will be extracted to `C:\avrdude`. It is available at `http://download.savannah.gnu.org/releases/avrdude/avrdude-6.3-mingw32.zip`

Once both zip files have been extracted, the path must be added for both programs. This can be done by going to the Windows settings application, searching *Path* and selecting *Edit the system environment variables*. A window should appear, select *Environment Variables*. Under *System Variables*, select *Path* and click *Edit*. Click *New* and then add in the location of `avr-gcc` and `\bin`. Same for avrdude. Once done, click *Ok* and close all the windows. The result should look like something similar to the image presented below:



Before launching avrdude, `libusbK` must be installed. It is available at `https://sourceforge.net/projects/libusbk/files/latest/download`. To test if

everything works, run the command `avrdude` in a command prompt to ensure that it executes, as well as `avr-gcc`.

Using the Toolchain

Introduction

This section details the use of the toolchain to compile and uploading software to the Arduino. In this case, there are two `.c` files called `myfile1.c` and `myfile2.c` that need to be compiled.

Step 1: Compiling

The typical usage of the compiler is the following:

```
avr-gcc -Wall -mmcu=atmega328p -Os -o myprogram.bin myfile1.c
                            myfile2.c
```

This will compile `myfile1.c` and `myfile2.c` and produce a binary file called `myprogram.bin`. The reasoning for the flags is as follows:

- `-Wall`:

  Display all warnings. This is a good practice because generally speaking, a warning could indicate some kind of possible runtime error. This is commonly seen especially in errors involving typecasting.

- `-mmcu=atmega328`:

  Set the microcontroller targer to the ATmega328p. This target is required since the Arduino Nanos used in the SOEN422 labs have these microcontrollers.

- `-Os`:

  Optimize for storage. There are different optimization flags and levels including `-O1`, `-O2` and so on. Be careful when using optimziation flags because some functions may not necessarily behave as expected. It is critical to keep this in mind when debugging software.

- `-o myprogram.bin`:

  The linked binary output of the compiler. This name could be changed to whatever is desired.

  For more information on compiler flags, consult the documentation at `https://www.nongnu.org/avr-libc/user-manual/using_tools.html`

Step 2: Convert to a .hex File

Once the compilation is complete, a `.bin` file is produced. While this is a binary file, it is not suitable for being uploaded onto the microcontroller. To prepare the file in a more appropriate format, the file must be converted to an Intel hex format. To do this, the `avr-objcopy` tool is used. This is done by the following command:

```
avr-objcopy -O ihex myprogram.bin myprogram.hex
```

In this command, we specify that the output format is supposed to be Intel Hex via the `-O ihex` flag. Once complete, the code is now ready to be programmed onto the Arduino.

Step 3: Uploading Code

Determining the Serial Port on Linux

For Linux, the serial port can be determined by running

```
ls /dev | grep ttyUSB
```

In this case, the command returned just solely `ttyUSB0`. This means that the port to be used is `/dev/ttyUSB0`. It is also a good idea to add the current user to the dialout group so that avrdude does not require sudo elevation each time it is ran. This can be done by running the command

```
sudo adduser my_user dialout
```

then logging out and back in again.

Determining the Serial Port on macOS

On macOS, this can be done by running the following command:

```
ls /dev | grep usb
```

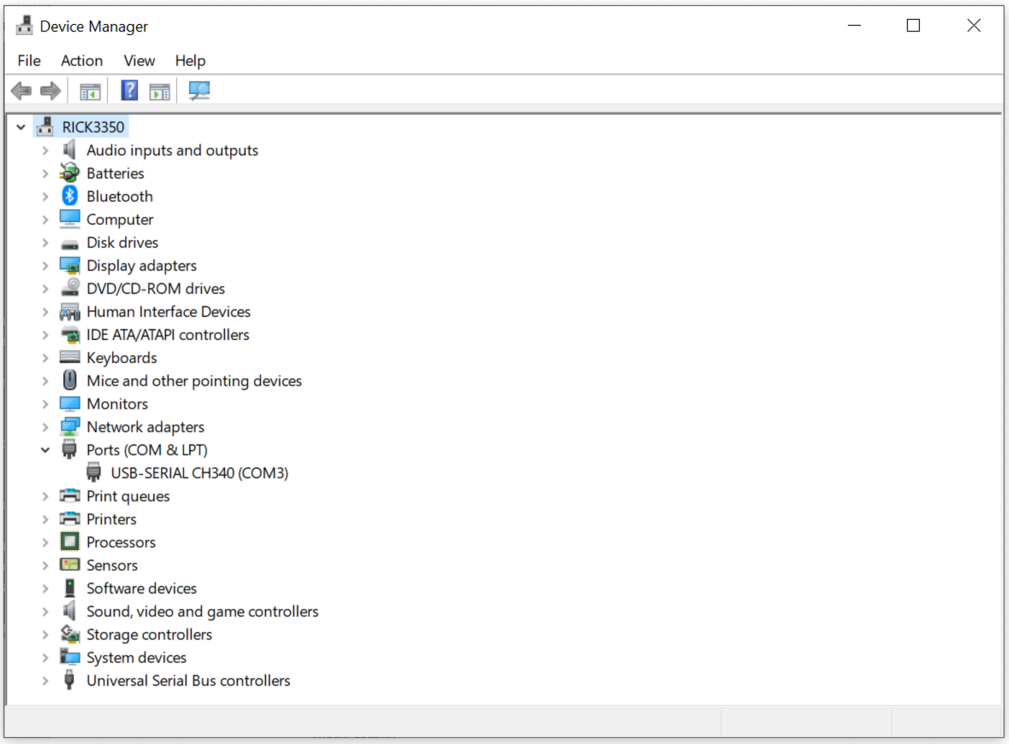
which returns:

```
cu.usbserial-1430
tty.usbserial-1430
```

and implies that the serial port here is `/dev/cu.usbserial-1430`.

Determining the Serial Port on Windows

On Windows, go to *Settings* and then search for *Device Manager*. Once in the device manager, the port will be listed under *Ports (COM & LPT)*. Look for *USB-SERIAL CH340*. The value in the bracket is the serial port used. In this case, `COM3` is the port used as shown in the following image:



Using AVRDUDE

Now that the port has been determined, it is time to upload the code. As mentioned before, the desired `.hex` file to upload is called `myprogram.hex`. The following command is used:

```
avrdude -c arduino -p m328p -P myserport -b 57600 -D -U
                flash:w:myprogram.hex:i
```

where `myserport` is substituted for the serial port determined just before. As an example, to program on macOS, a suitable command can be:

```
avrdude -c arduino -p m328p -P /dev/cu.usbserial-1430 -b 57600 -D
                -U flash:w:myprogram.hex:i
```

The reasoning for the flags is as follows:

- `-c arduino`:

  This defines the programmer type as an Arduino.

- `-p m328p`:

  The microcontroller we are uploading to is the ATmega328p. A list of available parts is available in the AVRDUDE documentation.

- `-P myserport`:

  This defines the serial port used. This will vary depending on hardware configurations and operating systems used.

- `-b 57600`:

  This flag defines the baud rate to communicate with the Arduino to program, it must be set to 57600 baud.

- `-D`:

  This flag tells AVRDUDE not to wipe the chip entirely before programming. **This flag must be present at all times. Without this flag, the Arduino bootloader may be wiped and can become unprogrammable via USB.**

- `-U`:

  Perform a memory operation. In this case, the argument used is

  `flash:w:myprogram.hex:i`

  which tells AVRDUDE that the flash memory is being used, is being written to, then the source file is provided along with a `i` flag denoting that the Intel hex format is used.

A sample output of `avrdude`, assuming the programming was successful would be:

```
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: reading input file "main.hex"
avrdude: writing flash (144 bytes):

Writing | ################################################## | 100% 0.08s

avrdude: 144 bytes of flash written
avrdude: verifying flash memory against main.hex:
avrdude: load data flash data from input file main.hex:
avrdude: input file main.hex contains 144 bytes
avrdude: reading on-chip flash data:
```

```
Reading | ################################################## | 100% 0.06s

avrdude: verifying ...
avrdude: 144 bytes of flash verified

avrdude: safemode: Fuses OK (E:00, H:00, L:00)

avrdude done.  Thank you.
```

Additional Resources

Below are some useful resources that can aid in low level development for the AVR platform:

- Argument and Option List for AVRDude
  `https://www.nongnu.org/avrdude/user-manual/avrdude_4.html#Option-Descriptions`
- Common avr-gcc Flags:
  `https://www.nongnu.org/avr-libc/user-manual/using_tools.html`
- ABI and General Compiler information
  `https://gcc.gnu.org/wiki/avr-gcc`