SOEN 422

Tutorial 1

Agenda

1. Introduction
2. Installing Arduino IDE and Teensyduino
3. Basics of Arduino Programming (What you need to know for Lab 1…)

Introduction

- My name is Patrick
- Best way to contact me is at patrick.ayoup@gmail.com
    - I usually reply very quickly to emails.
- Final Year Student in Software Engineering
- This course is really fun!

Installing Arduino IDE and Teensyduino

- Download and install the Arduino IDE for your operating system. The current latest version is 1.0.5.
    - The Arduino IDE is written in Java. You may need to install the JRE if it is not already installed on your workstation.
    - Installation instructions are found here
- Teensy is an Arduino Compatible development board. Since it is not manufactured by Arduino, some middleware is necessary for loading code onto the device.
- Download and install Teensyduino for your operating system.
- Verify that your development environment works by following the instructions on the Teensy website for loading and running the "Blink" program.

Introduction to Arduino

- Arduino programs (called sketches) are written in C or C++.
- The Arduino platform is set of high level C++ libraries used to abstract the details of programming microcontrollers.
- Additionally, a small framework is provided to manage the control flow of the sketch execution.

Anatomy of an Arduino Sketch

- A sketch is structured into two functions: `setup()` and `loop()`.
- At the beginning of the sketch's execution, the `setup()` function is called once.
    - This is an ideal place to put initialization code.

```
void setup() {
  pinMode(led, OUTPUT);
}
```

- After calling the `setup()` function, the `loop()` function is executed as the body of an infinite loop.

```
void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

Anatomy of an Arduino Sketch

- Finally, constants, variables, and helper functions should be declared at the top of the sketch.

```
int led = 13; // or #define led 13

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

Digital Output

- To output a digital signal on a pin, that pin's mode must be set to `OUTPUT` using the `pinMode()` function.
- To set a pin to `HIGH` or `LOW`, use the `digitalWrite()` function.

```
int MY_PIN = 13;

// Set pin 13 to output mode and output a HIGH signal through that pin.
pinMode(MY_PIN, OUTPUT);

digitalWrite(MY_PIN, HIGH);
```

Digital Input

- To read a digital signal on a pin, that pin's mode must be set to either `INPUT` or `INPUT_PULLUP`.
    - When set to `INPUT`, if no source is connected to the pin, the state will be uncertain and could be read as either `HIGH` or `LOW`.
    - When set to `INPUT_PULLUP`, if no source is connected to the pin, the state is "pulled-up" to a `HIGH` state due to an internal pull up resistor in the microcontroller.
- The state of a pin is read using the `digitalRead()` function.

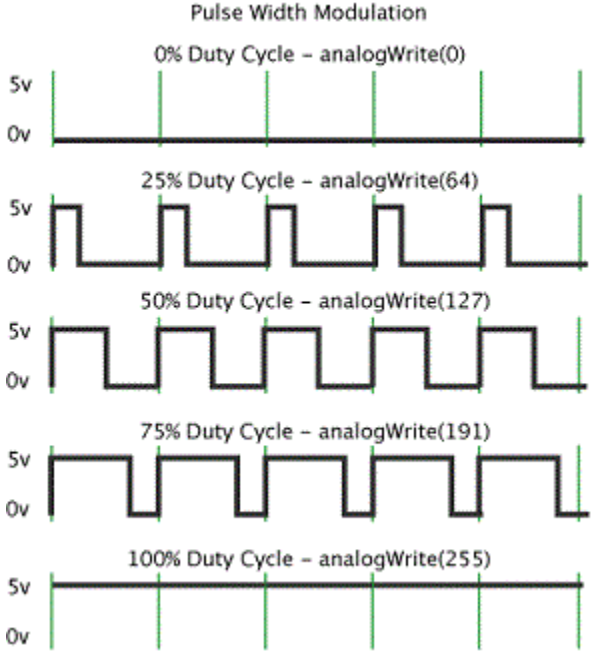
```
int MY_PIN = 13;

// Set pin 13 to input mode and store its value in a variable.
pinMode(MY_PIN, INPUT);

int value = digitalRead(MY_PIN)
```

Analog Output

- To output an analog signal on a pin, pulse width modulation (PWM) is used.
  - PWM is a technique for getting an analog signal.
  - A square wave is generated by rapidly switching a pin between 0V and 5V.



Analog Output

- The fraction of a given time interval where the pin is set to 5V is called the duty cycle.
    - Ex. If the signal is high for 25% of the interval and low for 75% of the interval, the duty cycle is 25% and a voltage of 1.25V should be read (0.75 * 5 = 1.25).
- To achive this in Arduino code, the `analogWrite()` function is used.
    - This function takes two parameters, the pin to write to and the duty cycle which is a value between 0 and 255.
    - For a 50% duty cycle, we provide 256 / 2 = 128.
- NOTE: Not all pins support PWM, you must verify on your pinout charts to find those which do.

```
int MY_PIN = 6;

// Set pin 6 to output mode and output a PWM with 25% duty cycle.
pinMode(MY_PIN, OUTPUT);

analogWrite(MY_PIN, 64);
```

Analog Input

- To read an analog signal on a pin, that pin's mode must be set to `INPUT`.
- The voltage at a pin is read using the `analogRead()` function.
- NOTE: Not all pins suport analog input, you must verify on your pinout charts to find those which do.

```
int MY_PIN = A0;

// Set pin A0 to input mode and store its value in a variable.
pinMode(MY_PIN, INPUT);

int value = analogRead(MY_PIN);
```

Serial Output

- The Arduino platform provides convenience functions for serial IO.
- To initialize the Serial library, use `Serial.begin()`.
    - You must provide a baud rate to this function. Usually we use 9600.

```
// Initialize the serial library to 9600 baud rate.
Serial.begin(9600);

// Output a string.
Serial.println("foobar");
```

Serial Input

- The Arduino platform provides many convenience functions to read serial input.
- Input is ASCII encoded. Convenience functions are available for parsing ascii representations of numeric characters to C integer types.
- View Serial documentation here.

Arduino IDE Serial Monitor

- The Arduino IDE comes with a serial monitor to view and send serial data sent through the USB connection between your workstation and the microcontroller.

Hardware Timers and Interrupts

- The microcontroller on the Teensy++2.0 has hardware timers which can be accessed through third party libraries.
- The TimerOne library is used to access the 16 bit timers.
- The timer is set with a given period and at the end of every period, an interrupt function is executed.
- The timer can also switch PWM pins based on the provided period.
- See the API documentation for more information.

Hardware Timers and Interrupts

- To initialize the TimerOne library, use `Timer1.initialize()`.
  - You must provide a period to this function in microseconds.
- To attach an interrupt handler, use `Timer1.attachInterrupt()`.
  - You must provide a callback function to this funciton.

```
// Initialize the period to 1 second.
Timer1.initialize(1000000);

// Attach an interrupt handler.
Timer1.attachInterrupt(callbackFunction);
```

Notes about the Teensy Timer Libraries

- The choice of whether to use TimerOne or TimerThree depends on which PWM pins you are using.
- On the Teensy++ 2.0
  - **TimerOne**: Pins 25, 26, 27
  - **TimerThree**: Pins 14, 15, 16

Installing Third Party Arduino Libraries

- The Arduino community has developed many open source third party libraries.
    - Often, these libraries are distributed as a zip file of C++ code.
- To install a zipped library, from the Arduino IDE select "Sketch > Import Library... > Add Library..." and select the zip file.
- The library should then be available. You can confirm that the library was installed by looking at the list of installed libraries under "Sketch > Import Library..."
- More information on installing libraries is available at the Arduino website.
- Use include statements to include a library in your sketch.

```
#include <TimerOne.h>
```

Other Helpful Functions

- To pause execution for a period of time, use the `delay()` function.
    - Takes an integer representing the time to delay by in milliseconds.

```
// Delay for 1 second.
delay(1000);
```

- When dealing with asynchronous code (interrupts), protect critical sections with `noInterrupts()` to disable interrupts and `interrupts()` to re-enable them.

References

- Arduino Reference
- INPUT vs INPUT_PULLUP
- Pull-Up Resistors
- PWM
- PWM Video
- Arduino Serial Module
- Teensy Timer Libraries

Copyrights

- PWM image on Slide 10 is provided by Timothy Hirzel.