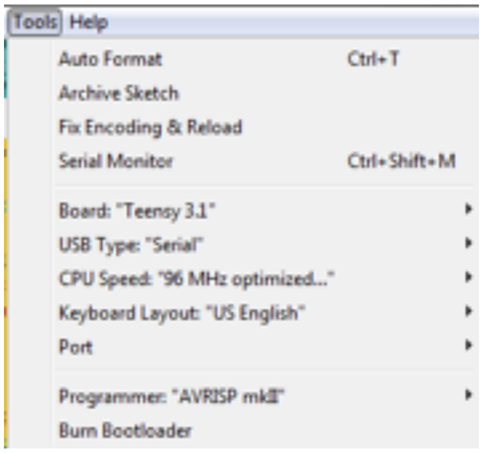
Appendix A: Programming the Teensy Microcontroller

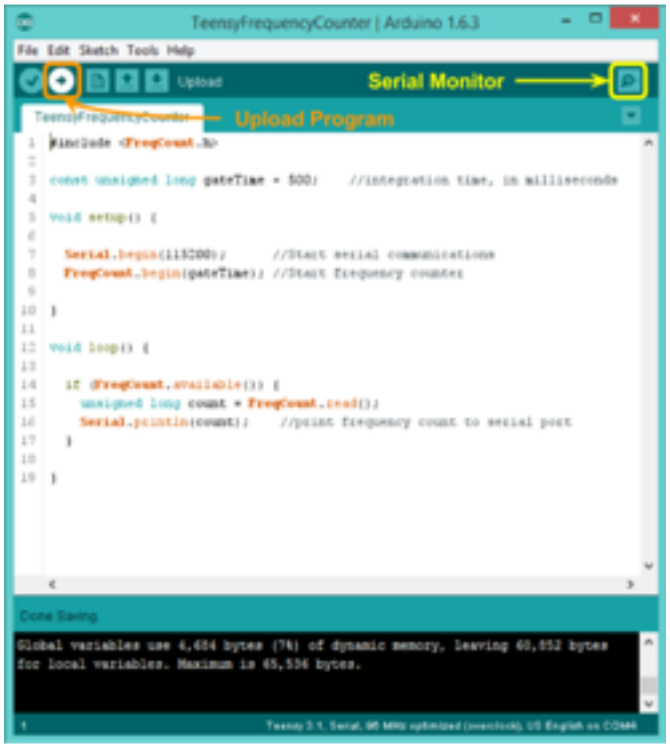
The Teensy microcontroller connects to a computer via a micro-USB cable. Your instructor has already installed the Arduino software it utilizes:
https://www.arduino.cc/en/Main/Software

Your instructor has also installed the Teensyduino software (with all libraries installed – we will make particular use of the FreqCount library):
https://www.pjrc.com/teensy/td_download.html

Connect the microcontroller and open the Arduino.exe program. The following settings need to be selected in the Tools menu for this to function properly:



Board: "Teensy 3.1"
USB Type: "Serial"
CPU Speed: "96 MHz optimized..."
Keyboard Layout: "US English"
Programmer: "AVRISP mkII"



```
#include <FreqCount.h>

const unsigned long gateTime = 500;    //integration time, in milliseconds

void setup() {

  Serial.begin(115200);       //Start serial communications
  FreqCount.begin(gateTime); //Start frequency counter

}

void loop() {

  if (FreqCount.available()) {
    unsigned long count = FreqCount.read();
    Serial.println(count);    //print frequency count to serial port
  }

}
```

```
Global variables use 4,484 bytes (7%) of dynamic memory, leaving 60,852 bytes
for local variables. Maximum is 65,536 bytes.
```

Teensy 3.1, Serial, 96 MHz optimized (overclock), US English on COM4

To program the Teensy, open Henry's version of the code you'll need (or, alternatively, copy and paste the code shown on the next page) and click the **Upload** button as shown above. Open the **Serial Monitor** to communicate with the Teensy over the serial interface.

```
//-------------------------SPADCounter02.ino-------------------------
#include <FreqCount.h>//frequency counting library
#include <Bounce.h>//debouncing library

const int sensePin = 13;//pin for detection events
const int intervalPin = 1;//pin that sets the system into interval counting mode
const int freqPin = 12;//pin that sets the system into frequency counting mode
const int intervalLED = 23;//status indicator for interval mode
const int freqLED = 14;//status indicator for frequency mode

byte oldState;
volatile unsigned int bangTime = 0;//variable created to store the number of
microseconds between detection events
volatile int dataSent;//logical value of whether or not the NEW time interval data has
been sent. 1 true,0 false
const unsigned long senseLength = 1000;//time DURATION to collect interval data
(specified in milliseconds)

Bounce intervalButton = Bounce(intervalPin, 25);//bounce is a library used to
debounce. Bounce is the appearance of spikes of electric signal during
//logic signal switching, such as mechanical effects. Bounce(uint8_t pin, unsigned
long interval_millis ) creates an instance of the
//Bounce class, attaches pin and sets interval to interval_millis. The effect is to
remove bounce of the pin with the chosen parameter. See github bounce libraray
//for more info
Bounce freqButton = Bounce(freqPin, 25);//freqButton is a bounce instance, and the
instance is attached to pin 12 (freqPin=12) with interval 25.
//Bounce is used for mode switching pins because they are attached to mechanical
buttons which need debounce to be stable.

void setup() {
  pinMode(1, INPUT_PULLUP);//set mode of pin to resistive pullup input with internal
pullup resistor. Default electrical value would be HIGH, and becomes LOW when
  //a grounded button is pressed. Search pullup resistor for more information.
  pinMode(12, INPUT_PULLUP);

  pinMode(intervalLED, OUTPUT);
  pinMode(freqLED, OUTPUT);
  digitalWrite(intervalLED, LOW);
  digitalWrite(freqLED, LOW);

  Serial.begin(115200);//set baud rate of serial communication

}

int countMode = 0;     //mode = 0: nothing happens; mode = 1: frequency counter
operational; mode = 2: Interval Timer
elapsedMillis elapsedTime;
elapsedMicros senseTest;//a variable that increases as time goes to store the time
elapsed between events
```

```
void loop() {
//pins assigned to a bounce instance can be called (read change in value, read value,
detecting edge, etc.) with bounce methods.
//update, fallingEdge are some bounce methods. See bounce github for more info
  if (freqButton.update()) {
    //if the frequency mode button updates
    if (freqButton.fallingEdge() && countMode == 0) {
//only triggers the code to start the frequency counter when the button is pressed,
which corresponds to LOW electric signal. Remember the pin is in resistive pullup
mode.
//fallingEdge is used so that risingEdge does not trigger the code again to avoid
double triggering.
//This starts frequency counter when the button is pressed and when we are in nothing
mode.
      FreqCount.begin(1000);//FreqCount is a library for counting the number of events
over an INTEGRATION TIME (specified in milliseconds; default setting = 1000)
      elapsedTime = 0;
      Keyboard.print("time [ms]");
      Keyboard.set_key1(KEY_TAB);
      Keyboard.send_now();
      Keyboard.set_key1(0);
      Keyboard.send_now();
      Keyboard.println("Frequency [Hz]");
      digitalWrite(freqLED, HIGH);//indicate frequency counting is in progress
      countMode = 1;//show that we are in frequency counting mode.
    }
    else if (freqButton.fallingEdge() && countMode == 1) {
      //if we are already in frequency counting mode and the button is pressed, stop
the counter.
      FreqCount.end();
      digitalWrite(freqLED, LOW);//indicate frequency counting mode is over
      countMode = 0;//show that we are back to nothing mode
      Keyboard.set_modifier(MODIFIERKEY_CTRL);
      Keyboard.set_key1(KEY_HOME);
      Keyboard.send_now();
      Keyboard.set_modifier(0);
      Keyboard.set_key1(0);
      Keyboard.send_now();

    }
  }

  if (intervalButton.update()) {
    if (intervalButton.fallingEdge() && countMode == 0) {
      digitalWrite(intervalLED, HIGH);
      countMode = 2;

      Keyboard.print("integration time [ms]: ");
      Keyboard.println(senseLength);
      Keyboard.println("interval time [us]");
      dataSent = 1;
      elapsedTime = 0;
      senseTest = 0;
      attachInterrupt(digitalPinToInterrupt(sensePin), isr, RISING);//enable sensePin
interrupt. Whenever sensePin receives a rising edge electric signal,
      //the interrupt is triggers and executes the interrupt code. The interrupt
function is called isr.

    }
  }

  if (countMode == 1 && FreqCount.available()) {
    unsigned long freq = FreqCount.read();
    Keyboard.print(elapsedTime);
    Keyboard.set_key1(KEY_TAB);
    Keyboard.send_now();
    Keyboard.set_key1(0);
    Keyboard.send_now();
    Keyboard.println(freq);
    Serial.println(freq);
  }

  if (countMode == 2 && dataSent == 0) {
    //if in interval counting mode and new time interval data has been obtained but
not yet sent (dataSent=0)
    Keyboard.println(bangTime);
    dataSent = 1;//new data is sent, so the logic becomes true
  }

  if (countMode == 2 && elapsedTime > senseLength) {
    //if in interval mode and time elapsed exceeds the time intended for data
collection
    detachInterrupt(digitalPinToInterrupt(sensePin));//disable interrupt of sensePin
    digitalWrite(intervalLED, LOW);//indicate that interval counting mode stopped
    countMode = 0;//shows that we are in nothing mode
  }

}

void isr() {
  //when sensePin receives a rising edge electric pulse, the code is executed.
  bangTime = senseTest;//set time taken for a pulse to be detected as senseTest, which
is the time elapsed
  senseTest = 0;//clear the time elapsed between detection events
  dataSent = 0;//since new data of time interval is acquired, we change the logic of
dataSent to false 0 so that the code will send this data.
}
```