TSK x REVO

2.0:
Week 3! This week we will be all about programming your arduino, and learning how to make it talk to your computer. We will be using a python program that receives data from the arduino. This is useful for many applications like managing wireless sensor networks or uploading data to the Internet.

2.1:
“The USB” is actually a correct way to refer to the universal serial bus. This is one of many ways to communicate from your computer to your device. In the case of the bootloader on your TSK arduino, the ATMEGA328P is running VUSB, a library that lets the chip simulate being a USB device. There are four kinds of USB control transfers, isochronous, bulk, interrupt and control transfers. They are pretty much what they sound like. Isochronous transfers are scheduled and guaranteed to happen, because the bus controller reserves time for them. Bulk transfers move a lot of data, but are not scheduled. Interrupts happen as-called, but do not conflict with isochronous transfers. The program we will be using was ghetto-coded to work, and uses control transfers, which are normally used to tell the USB controller what kind of device is plugged in. Fortunately, they can also transfer four 2 digit hexadecimal numbers, allowing four channels of “Data” to be passed back and fourth. All these transfers can be directed in or out, but they are always initiated by the host. That means if the device has data to send to the computer, the computer has to request it in a way that the device knows how to send it. The “knowledge” of how and what to send is in the devices memory as known instructions. Some USB devices identify as particular devices, like human interface devices (HID) and have special universal protocols. That is how your plug-and-play mice work without drivers- you plug them in, they identify as a mouse, and then transmit standardized packets that the computer already knows how to accept.

2.2:
Most arduinos use the serial library and a virtual serial port for communication in and out of the computer. Back in the day, most computers had a serial printer port that could be used for programming microcontrollers, but today these ports are rare. Rare in desktops, and nonexistient in laptops. The Arduino has a chip made by Future Technologies Devices Inc. (FTDI) on it that turns the USB signals that your computer spits out into serial signals for the device and vice versa. If you are in linux try running $: lsusb; you will see that the device that your serial bus recognizes is an FTDI USB to serial converter.

Serial of course, just refers to how the bits are sent. Serial indicates that the bits are sent one at a time, one after the other. Parallel would mean that multiple bits are sent at a time, which is faster, but takes more pins on a microcontroller. May atmega mcus have UART modules built in for serial commnunications.

2.3:
Many new chips actually have USB built in. Our does not, but it has VUSB, so that is almost as good. By running a simple python script, we can use control transfers to send or capture data. Lets look at an example that uses the UsbSimple library, and some stuff we learned about sensors last week.

For this, we will need to set up the UsbSimple library. I will have it on a flash drive for the class, and if I get Kevin's permission, I will put it up on the downloads page of my website.

You will get a folder with two important things in it. First, a python script in a folder called “python”. You may as well rename it pystream. Take this folder and put it on your desktop. Take the rest of the folder and copy it to arduino/libraries.

With that done, sudo arduino, or whatever you do to start up the arduino IDE. Also get ready to run the python script in the pystream folder. In linux, or windows, as long as python is installed, you can just run it from the command line with “python stream.py”. Don't run it quite yet, as it will fail because there is no USB device attached yet!

First, attach an led from A5 to ground, with the short lead in the ground, and the long lead in A5. Then put an LDR from A4 to ground. On pins 9 and 10, put LEDs in the usual configuration

Now lets put some code into the IDE:

```
//include the UsbSimple library, so that you can use the functions in it
#include <UsbSimple.h>

//make some integers for storing data in.  One is for a LED sensor, and the other is for a LDR.
int LDR;
int LED;

//void setup is where we start up connections, initialize pins, etc.
void setup(){
  USB.begin();  //USB.begin(); tells the chip to start acting like a usb device
  pinMode (A5, INPUT);  //set up analog in 5 as an input from the LED
  pinMode (A4, INPUT);  //ditto for A4, but from the LDR
  pinMode (9, OUTPUT); //set up pin 9 as output to the LED
  pinMode (10, OUTPUT); // LED output
  digitalWrite(A4, HIGH); //Set up pull up resistor for pin A4, with the LDR
}

void loop(){
  USB.refresh(); //This function has to be called every now and then

//read in the values from the LDR and LED
  LDR=analogRead(A4);
  LED=analogRead(A5);

//send data out using the USB connection.  There are four channels, 0-3, here we use 0 and 1
  USB.set(0,LDR);
  USB.set(1,LED);

//use map to convert the 10 bit adc value to something we can pwm out.
  LDR=map(LDR, 0, 1023, 0, 255);
  LED=map(LED, 0, 1023, 0, 255);
//write to the output leds the light values of the LDR and LED.
  analogWrite(9, LDR);
  analogWrite(10, LED);
}
```

Once that is flashed to the chip, give it a few seconds and then go ahead and run stream.py. You should see some numbers; that is what the chip is reading as the digital value from the ADC!