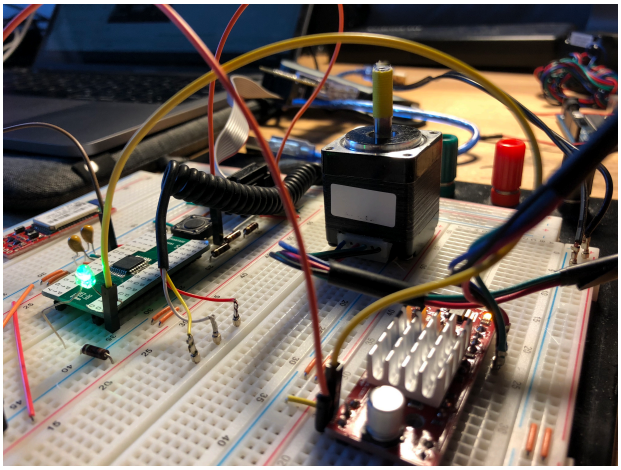
2020-05-08
Project Update



Overview

Here is a summary of progress made on my project.

Camera Shutter Release

I was able to actuate my camera shutter release with the 8051 and also with Arduino. I had previously accomplished this through Arduino, however I was mostly "guessing" when it came to the three chords—red, white, and yellow shown in the image above—from the spliced open shutter release cable. Now I actually understand the basis of the chords—I read the spec sheet and know what the heck they do. Yellow is for the camera focus, which I didn't need to use. The remaining lines operate much like the "light an LED" problems that we solved in class. I tied the white cable to ground, and then the red cable to a pin, which is normally tied low until my program throws it high, thus releasing the shutter. Just noting my understanding.

Actuating the Stepper
8051_ stepper_off

Please see the above video link. You'll see that I adapted the bitflip assembly code to actuate the stepper motor. Not exactly what I wanted the stepper motor to do but I have to say this was amazing to watch when I did it. Seeing the stepper motor turn in response to something I was doing with my own custom microcontroller was just awesome.

Improved Actuation of the Stepper

8051_ stepper_on

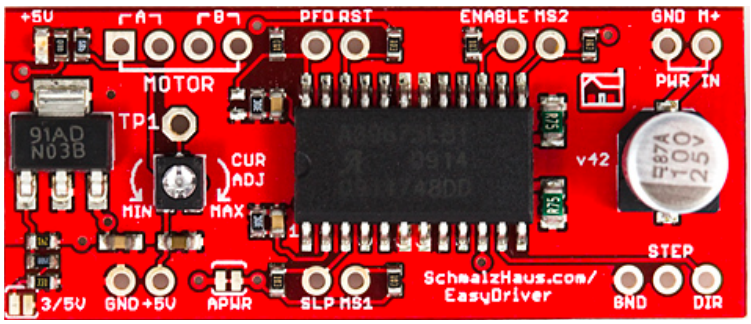
Please see the above video link. You'll see I've come up with some better Assembly code here. I believe this is an adaptation of Scott's file. It allows me to press a button and turn the motor using an interrupt pin. I am still working on smoothing out the movement of the motor, which I seem to have some leads on. Spec sheets and tutorials are telling me that the time between the edges you send the motor matters, so I'll see if I can play with that this weekend.

Shoot-Move-Shoot
camera_motor_sync

Please see the above video link. I have successfully prototyped my "shoot-move-shoot" sequence thus far on Arduino. This was extremely exciting for a few reasons. I have previously done something kind of like this with Javascript, but again it was during a time when I just didn't really know what I was doing, and when it worked it was more of a feeling of having "gotten lucky" with the shots in the dark I was taking, than a feeling of accomplishment. This time, I actually knew what I was doing. It was all quite intuitive actually. I am sharing my C++ code file for this but please be aware I am sharing it in an incomplete state as I am troubleshooting some cleverness I am trying to add to it. I plan to port this over to my 8051 and Assembly language this weekend.

Improved Understanding of Stepper Motor Drivers

I have worked with stepper motors before, but have always been kind of mystified by them, as I have mostly treated them and the drivers as black boxes. I knew what they do mechanically but not really electronically. For this project, I took out a spare stepper motor driver I had in my stash, and it was immediately more legible to me than it had been prior to taking this class.



First of all I understood why it had its own power input, and why it had a range on it that went above 5V. Yes you can use stepper motors with Arduino, which have handy 5V output pins, but what if you want to use a big stepper motor that needs 12V? Well forget those 5V Arduino and give the driver its own 12V power supply. I just didn't really understand that all power for all devices did not have to come from the measly 5V off of a microcontroller. Basic, I know, but this was a revelation.

In the past I've also burnt out a lot of stepper motor drivers, and I've been totally unsure as to why that has happened. I watched a whole bunch of Youtube walk-throughs as part of my

project and one guy happened to mention that “if you unplug the stepper motor from the driver while the driver is still powered on, it will burn out the driver.” Wow, what a revelation. That’s what was happening to me all of those times. Though I learned this on Youtube, it was part of a new process of really getting to know the components that I was working with, including reading through their documentation and observing use cases, in order to understand what the heck they were doing, instead of just assuming things. I owe that process to this class. I’m no longer really that intimidated by spec sheets and pin names. When I grabbed my stepper motor driver this time around, I saw things like RST, GND, 5+, ENABLE—I know what those are!

The Enable Pin

Let’s talk about ENABLE real quick, because this was another cool moment for me. I know what an ENABLE pin does now! So I used it in my code to do something pretty cool. I wanted to run a routine that with my motor-camera device whose duration was effectively the angle of rotation that I programmed in. So for instance, I want to shoot a time lapse of a flower spinning on my motorized turn table for precisely 270 degrees of rotation, and then halt. Well that’s quite easy now that I know what an ENABLE pin does. The motor driver will only act on STEP and DIR instructions so long as the ENABLE pin is low. So in my code I setup a loop that throws the ENABLE pin high after a for loop is completed.

```
void loop() {

for(int k = 0; k<SHOTS; k++){

      for(int i = 0; i<DEGREES_CONVERTED; i++){
            digitalWrite(STEP_PIN, HIGH);
            digitalWrite(STEP_PIN, LOW);
            delay(1);                    //this section sends an edge to the stepper
            }

       delay(300);                       //pause between motor stop and camera shutter fire
       digitalWrite(SHUTTER_PIN, LOW);   //camera shutter gets fired
       delay(300);                       //time shutter is held (bulb)
       digitalWrite(SHUTTER_PIN, HIGH); //shutter goes back to high position
       delay(500);                       //time between shots

    }

  digitalWrite(ENABLE_PIN, HIGH);          //stop the rotations are complete stop the motor

}
```

I know this is simple but I think it is really cool. A few more points about it. As I adapt this to my Assembly language I plan to try to mimic this with DJNZ. Wowzers that would be neat. Lastly I’d also like to say that I can use this enable pin in another way. I could be a kind of “emergency switch” too. For instance I could setup a temperature sensor that throws the ENABLE pin high if the motor or its driver get too hot. Or I could setup a mechanical switch

on the end of a rail, so that if a camera carriage transverses the length of the rail and bumps the switch, the motor stops and elegantly prevents the carriage running off of its edge.

Now that I think of it, what about using the DIR pin for that instead? If the camera reaches the end of the rail, the switch flips the bit on the DIR pin and then, boom, the carriage goes back in the opposite direction. Yowzer!

Edge Triggering

Another thing I learned while playing with the stepper motor and my code this week is that stepper motors, or really their drivers, are edge-triggered devices. And I know what that means! The STEP pin produces a step (at whatever angle of rotation is specified on the data sheet) when it receives a low then high edge. You need a tiny bit of delay in between the next one, from what I gleaned from documentation and tutorials. Certainly makes sense to me from this course. So in order to run my stepper motor, all I really had to do was setup a loop where I was flipping the bit associated with the pin attached to the STEP input.

You'll see in my (unfinished!) code that I am trying to get clever with this loop. I'll be refining it so that there is a conversion factor going into the variable determining when the loop ends. This will allow me to set the angle of rotation for my time lapse cycle.

Mechanics and Photography

Why sync a motor and a camera shutter? I have previously built the mechanics of a turntable controlled by a stepper motor for shooting time lapses with rotational motion of a subject, like a mushroom growing for instance. I never really shot anything with this device because honestly I just got hung up with the code, software, and mysteriously burnt-out motor drivers, and just put it down. Now I have the know-how to persevere and use it! Considering this, I've prepared some elements of this mechanical device and photography as well.

I reassembled the turntable and reattached the stepper motor. It works well. I'll add a video of it working for my final submission. I also went ahead and installed it into a dark room I built inside of my apartment, taking it upon myself to drill a two-inch hole through my table so the device can elegantly float above the table while it's connected to a tripod below. Exciting!

I've also mounted the camera, lightbox, and lighting, so I can capture a nice shot with good even lighting. "Lighting" is actually a topic I am looking forward to gaining more understanding of in the Fall section (fingers-crossed). I'd like to build my own photography light based off analog knowledge, but I'll cross that bridge later.

Sensors

I got my sensors in the mail a bit late, just in the last few days, so I haven't been able to play with them. Still they're kind of awesome looking (check out my CO2 sensor!), and I'm getting friendly with their spec sheets. I'll hopefully have an update with my final submission. That's all for now.

```
//this code is unfinished. It has been working really well but I am currently making changes but
I wanted to get this submission in for tonight so this snapshot is a version that is in flux.

int DIR_PIN = 2;
int STEP_PIN = 3;
int ENABLE_PIN = 4;
int SHUTTER_PIN = 10;


int STEPS_PER_ROTATION = 40;
int DEGREES = 360;
int DEGREES_CONVERTED = (DEGREES*STEPS_PER_ROTATION)/360;

//int DEGREES = ;
//int INT = ;
int SHOTS = 50;
int TRAVEL = STEPS_PER_ROTATION/SHOTS;

void setup() {
  pinMode(DIR_PIN, OUTPUT);
  pinMode(STEP_PIN, OUTPUT);
  pinMode(ENABLE_PIN, OUTPUT);
  pinMode(SHUTTER_PIN, OUTPUT);
  digitalWrite(SHUTTER_PIN, HIGH);
  digitalWrite(ENABLE_PIN, LOW);
  Serial.begin(9600);

  Serial.print("DEGREES = ");
  Serial.println(DEGREES);
  Serial.print("STEPS_PER_ROTATION = ");
  Serial.println(STEPS_PER_ROTATION);
  Serial.print("DEGREES_CONVERTED = ");
  Serial.println(DEGREES_CONVERTED);
  Serial.print("");

}

void loop() {

for(int k = 0; k<SHOTS; k++){

      for(int i = 0; i<DEGREES_CONVERTED; i++){
            digitalWrite(STEP_PIN, HIGH);
            digitalWrite(STEP_PIN, LOW);
            delay(1);                    //this section sends an edge to the stepper
            }

       delay(300);                       //pause between motor stop and camera shutter fire
       digitalWrite(SHUTTER_PIN, LOW);   //camera shutter gets fired
       delay(300);                       //time shutter is held (bulb)
       digitalWrite(SHUTTER_PIN, HIGH);  //shutter goes back to high position
       delay(500);                       //time between shots

    }

  digitalWrite(ENABLE_PIN, HIGH);         //stop the rotations are complete stop the motor

}
```