ECE271: Microcomputer Architecture and Applications — University of Maine

Lab #6 Stepper Motor in Assembly

Week of 7 March 2022

Goals

1. Learn how to make function calls with Thumb2 assembly language.
2. Learn how to use the Thumb2 ABI when making function calls.

Pre-lab

1. Complete the pre-lab before attending lab. The pre-lab is in a separate pdf file, found on the website.

2. Be sure to bring a breadboard and wires to the lab. Also bring your stepper motor/drive board from the previous lab.

Lab Procedure

The end goal of this lab is to be able to exhibit fine-grained control over the position of the stepper motor.

You will need to be able to rotate the motor shaft exactly 360 degrees counter-clockwise via both half and full stepping.

Part A – Hardware Setup

1. This is the same as Lab#5.

Part B – Code – Initialization

1. This lab will be done in Assembly.

2. Use your code from Lab#4 (LED assembly) as a base. Update `main.s` by adding the stepper motor routines.

3. (Note, if you are using Linux, the filenames will be slightly different.

4. Add a `Stepper_Pin_Init` function.

   (a) For this lab, follow the ABI discussed in class.

      i. Pass any arguments in r0 - r3
      ii. Any return values go in r0
      iii. If you use any of r4 - r10, you will have to save them at the beginning of your function with `push` and restore them at the end with `pull`
      iv. If you call another function, you will need to save and restore `LR` (r14)

v. Be sure to return at the end of the function.

(b) Be sure GPIOBEN is enabled and that pins GPIOB pins 2, 3, 6, and 7 are set as digital outputs in the `MODER` register.

5. Add a call (via a branch and link instruction) to this `Stepper_Pin_Init` function from your main routine.

Part C – Code – Full Stepping

1. Set up the full-stepping code. As a summary:
   - The internal motor has 32 steps per revolution
   - Gear reduction of 1/63.68395, or approximately 1/64
   - Thus it takes 32*64 = 2048 steps for one full turn of the input shaft.
2. Create a function,

   ```
   Stepper_Full_Step
   ```

   that takes one parameter (the angle) and rotates the motor shaft by `angle` degrees counter-clockwise.
3. Use the hex values you calculated in Lab#5 for the four steps, updating the

   ```
   GPIOB->BSRR
   ```

   register for each step.
4. You will want to delay between each step. Create a function, `Delay` that does this. The function should take a single parameter which is the amount of times to repeat a simple loop.

   You must use this `Delay` routine for your timing delays.
5. Remember that in lecture we went over what this code will look like.
6. It takes 2048 steps, or 512 repeats of the 4-step pattern, to complete a rotation. Your function should do the math to convert the value in degrees to a number of steps, and then do the rotation. Use integer math for this, no floating point.
7. Call this function with various angles and be sure it does the right thing. We will have you demo 360 degrees.

Part D – Code – Half Stepping

1. Set up the half-stepping code. As a summary:
   - The internal motor has 64 steps per revolution
   - Gear reduction of 1/63.68395, or approximately 1/64
   - Thus it takes 64*64 = 4096 half-steps for one full turn of the input shaft.
2. Create a function,

```
Stepper_Half_Step
```

that rotates the motor shaft by the `angle` passed in (counter-clockwise).

3. Use the hex values you calculated in Lab#5 for the eight steps, updating the

   ```
   GPIOB->BSRR
   ```

   register for each step.

4. Again, use the `Delay` routine for delaying.

5. It takes 4096 half-steps, or 512 repeats of the 8-step pattern, to complete a rotation. Your function should do the math to convert the value in degrees to a number of steps, and then do the rotation.

6. Call this function with various angles and be sure it does the right thing. We will have you demo 360 degrees.

Part E – Something Cool

Do something cool! You can come up with something on your own, but here is a list of ideas you can use.

1. Do whatever you did for Lab#5. This might be difficult if you used the keypad or LCD in your Lab#5.

2. Modify the LED code to live in a function, and call it to blink the RED and GREEN LEDs while the motor is stepping.

3. Try out some sort of advanced feature, such as storing to a local variable on the stack, or using advanced addressing modes to access arrays (this might be useful if you had the stepping values in an array in Lab#5)

Lab Demo

1. Submit your code
   - Complete a README with the post-lab answers.
   - Make sure the code is properly commented.
     This includes a header at the top of your main.s with your name and a brief summary of the lab.
   - Check your code and README into your gitlab tree.
2. Demo your implementation to your lab TA.
   (a) Rotate the motor 360 degrees counter-clockwise using full stepping.
   (b) Rotate the motor 360 degrees counter-clockwise using half stepping.

Post-Lab

- Place your answers to the question in a file Readme.md
- Submit with your code via the gitlab server.
- Questions:
  1. Why is an ABI useful when writing code? Why not just pick any register we want for passing parameters?
  2. Why does C store local variables on the stack, rather than forcing you to just use all global variables?
  3. Find the binary image that is being uploaded to your board. How big is this file (written in assembly) compared to the C version from Lab#5?