ECE 271 – Microcomputer Architecture and Applications Lecture 12

Vince Weaver

http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

24 February 2022

Announcements

- Read Chapter 16 for Stepper Motor info
- Read Chapters 7 and 8
- Midterm, likely 8 March (two weeks)
  more info on that as it gets closer

Stepper Motors

- How do normal motors work?
- What if you want accurate positioning of result?
- Servo-motors (we'll deal with them in a future lab) have some sort of sensor that provide feedback on positioning
- Stepper motors also allow exact positioning, but by carefully stepping one position at a time

Bi-Polar Motors

- (See figure 16-1 in book)
- Use single coil to set S or N
- Need to fully reverse polarity of voltage to switch polarity, which requires H-bridge

Uni-Polar Motors

- (See figure 16-2 in book)
- Coil tapped in middle. So can switch polarity by applying voltage to either end
- Only half of coil energized so not as strong as bi-polar

Stepper Internals

- (See figure 16-6 in book, movies from prelab)
- Two permanent magnets with alternating teeth, S N S N S N
- Offset coils, when activate attract
- Need to send precise set of waveforms to increment/decrement by one step

Driving the Motor

- Stepper motor around 50 Ohms resistance, 5V, so V=IR, I=V/R, I=5/50 = 100mA (0.5W)
- Can the GPIO pins provide 100mA of current? No, then can only provide 10mA
- So instead we used a ULN2803 Darlington Array (See the ULN2803 datasheet for diagram)
- This amplifies the current using darlington-connected transistors
- Also includes diodes to avoid kickback (when you stop

powering motor, inductance in magnetic field collapses and sends back-current into the inputs and can fry chip)

Connecting the Motor

- We will use 4 GPIOs (PB2, PB3, PB6, PB7)
- Also connect the board to 5V and GND
- Use the 5V pin

Notes on Voltages on the Board

- 5V – presumably 5V regulated
- 5V_I – input, if supplying 5V externally
- 5V_U – 5V coming in from the USB cable(?)
- VBUS – 5V from the USB OTG controller(?)
- 3V3 – regulated 3.3V
- 3V – regulated 3V
- 2V5 – regulated 2.5V

Wave Stepping

See figure 16-8 in book.

```
           0     1     2     3     4
PB2        :____:     :     :     :
 A         |    |____:____:____:

PB3        :     :     :____:     :
!A         :____:____|     |____:

PB6        :     :____:     :     :
 B         :____|     |____:____:

PB7        :     :     :     :____:
!B         :____:____:____|     |
```

Full Stepping

Higher torque as pushing *and* pulling (see Fig 16-9)

```
        0    1    2    3    4
PB2     :____:____:    :    :
 A      |         |____:____:

PB3     :    :    :____:____
!A      :____:____|    :    |

PB6     :    :____:____:    :
 B      :____|    :    |____:

PB7     :____:    :    :____:
!B      :    |____:____|    :
```

Half Stepping

- Pattern is A!B, A, AB, B, !AB, !A, !A!A, !B
- Smoother and can got half-steps
- Can be less torque
- See Figure 16-11 in book

Micro-Stepping

- Instead of being full on or off, instead set partial voltages
- Generally sine/cosine on A/B.
- Smooth transition. Lot more complicated.
- How would you generate not-full voltage using GPIOs? DAC? PWM?
- See Chapter 16.6 in book

Steppers – Lab

- Will do both Full and Half stepping

Steppers – Programming

- To do this, we will use 4 GPIOs to control things
- The BSRR register makes it a bit easier to set/clear the GPIO pins at the same time.
- We will use 4 pins in the GPIOB register
- There will be a pattern we send on the pins that will cycle through and advance the stepper
- Function where you enter angle, and it rotates that much

Programming

- Delay with busy loop

Calculating Angle in C

- Motor is 32 full steps per revolution
- However it has gear reduction of 64, so 2048 steps
- So it takes 2048/4=512 repeats of the 4-step pattern to rotate 360 degrees when doing full-stepping

Calculating Angle in C

- Don't use floating point in the lab
  It might work, but we haven't learned about it yet.
- Multiply/divide ordering in C
  - steps=(512*degrees)/360;
  - steps=512*(degrees/360);
  - are the above equivalent? Mathematically, yes.
    In C, no. When using 32-bit integers, a number like 270/360 is going to evaluate to "0.75" which C will truncate to "0", not giving the result you expect.

Apple II example

- Stepper motors used when need exact control
  Example: Disk][drive in original Apple II
  Unusual in that it was purely software controlled, leading to lots of interesting copy protection methods

Character Encodings

- What makes a text character?
  - Our processor only understands binary.
  - The letter 'A' we say is 65 (0x40).
  - Is that implicit in the processor or in the nature of the letter 'A'?
  - No, it's arbitrary
  - Why have a standard like this? Otherwise it would be impossible to communicate text! Every computer would treat letters differently.

- ASCII – American Standard Code for Information Interchange
  - Standard from the 1960s
  - Nice features
  - Numbers are consecutive, from 0x30-0x39 (easy to convert to decimal)
  - Letters are consecutive
  - Lower case has constant offset from uppercase, easy conversion
  - Technically 7-bit. What do you do with 8-bit? Parity? Extended characters?

- ◦ Also control chars in bottom. Things like BELL (control-G), linefeed, carriage return, escape, etc.

- EBCDIC – IBM's standard. There were others. Some put char in 6 bits.
- Old systems missing chars? Uppercase only? How did people cope? How did the C compiler cope? Trigraphs.
- What about non-English languages. ess-tset? Umlauts?
- Unicode? 16-bit?
  wchar_t? Windows? Java? Will all languages fit in 16-bits? no
- UTF-8?

Top bit 1 indicates more than 1-byte long, can encode in up to 4 bytes. Regular C string manipulation will work on UTF-8, 7-bit ASCII is a subset

- Combining chars, security aspect of letters that look the same
- Politics involved.
- Emojis?