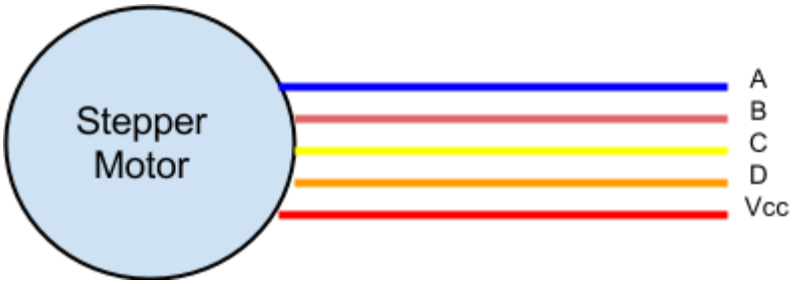
Stepper Motors

(2-day)

How Stepper Motors Work:

Inside a stepper motor is a “gear” shaped piece of iron, and electromagnets. The “teeth” of the piece of iron are attracted to the electromagnets when the magnets are charged. By energizing the electromagnets in a circular fashion, the “teeth” are pulled in a circular fashion, resulting in a rotating motor.

This lab uses a stepper motor that has a 5.625 degree step angle. The iron gear has 64 teeth, which means that 64 “phases” are required to rotate the motor 5.625 degrees. A phase can be described as a state where specific electromagnets are energized and others are not. This stepper motor used in this lab is a 28BYJ48 Arduino stepper motor. The 28BYJ48 has eight unique phases: A, AB, B, BC, C, CD, D, and DA. For the stepper motor to rotate, these phases must be executed in order. Once the last of the eight phases is reached, the cycle repeats again. Note: The name of each phase represents which lines are set to 1, the remaining lines are set to 0.



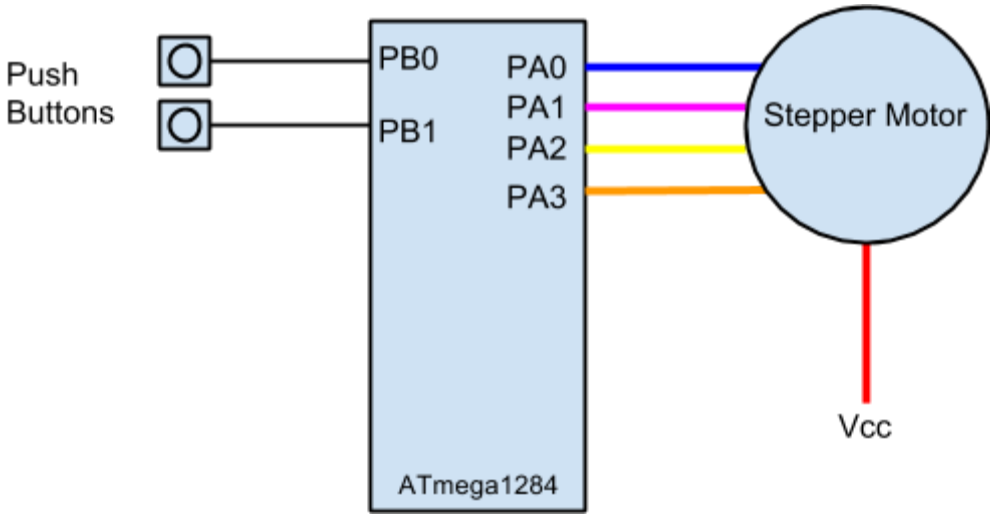
For further explanation, and a graphical representation of how a stepper motor works, follow the link below:

http://en.wikipedia.org/wiki/Step_motor

Why Stepper Motors:

Stepper motors are ideal for projects where exact rotation is desired. Since the user has direct control over the charging sequence of the electromagnets, the user can specify how far the motor rotates before stopping.

Wire your breadboard as the one pictured below



Note: It might be helpful to attach a long, straight object (like a pencil or unfolded paper-clip) to the rotating part of the motor. Doing this will make it easier to assess the results of the following exercises.

Exercise 1: Rotating the motor

Use two buttons (connected to PB0 and PB1) to rotate the Stepper motor clockwise or counter-clockwise. Design a state machine that transitions between the eight phases: A -> AB -> B -> BC -> C -> CD -> D -> DA -> A ...

Criteria:

- When the button connected to PB0 is pressed and held, the motor rotates counter-clockwise.
- When the button connected to PB1 is pressed and held, the motor rotates clockwise.
- When both buttons are pressed, or no buttons are pressed, the motor stops rotating.

Hints:

- To rotate the motor clockwise, transition between the phases in the order given in the intro. To rotate counter-clockwise, transition in reverse order.
- Remain in each phase for 3ms, before transitioning, to make the motor run smoothly

Video Demonstration: http://youtu.be/qqM-RjDtOPs

Note: If your batteries are low, the motor may not turn properly. You might notice either vibrations or the motor turns initially but then stops. If this is the case, replace the batteries or use the wall adapter.

Exercise 2: More Precise Rotation

Use six buttons to rotate the motor clockwise or counter-clockwise 90, 180, or 360 degrees

Criteria:

- Three buttons (connected to PB[2:0]) rotate the motor counter- clockwise. One rotates the motor 90 degrees. Another rotates the motor 180 degrees. The last rotates the motor 360 degrees.
- The remaining three buttons (connected to PB[5:3]) rotate the motor clockwise as described above
- As soon as a button is pressed, the motor begins rotating until completion. This happens whether the button is released mid rotation, or remains pressed past rotation

Hints:

- Since the 64 phases are required to rotate the motor 5.625 degrees, use this knowledge to determine how many phases are needed to rotate 90, 180, and 360 degrees.

```
// Number of phases to rotate 180 degrees
int numPhases = (180 / 5.625) * 64;
```

Video Demonstration: http://youtu.be/WQKsll_qeMM

Exercise 3: Exact Rotation

Use the keypad to enter a desired angle for the motor to rotate clockwise. Use keypad.h to drive the keypad.

Criteria:

- Connect the keypad to PORTD.
- The user can type in any angle between 0 and 999 degrees.
- The '#' button signifies when the user is submitting the angle
- If an invalid button is pressed ('A', 'B', '*', etc.), the motor should not rotate

Hints:

- Use a 'char' array to hold the button inputs.
- The library stdlib.h has a function that converts a char* to an int. This function is called "atoi" and takes in one char* parameter. "atoi" ignores leading whitespace and stops when whitespace and characters other than '0' - '9' are encountered. Some examples of "atoi" are given below. Make sure to include the library <stdlib.h>.

```
#include <stdlib.h>

int num_steps = 0;
unsigned char exp1[4] = { '3', '6', '0', '#' };
unsigned char exp2[4] = { '1', '8', ' ', '0' };
unsigned char exp3[4] = { ' ', '9', '0', '#' };
unsigned char exp4[4] = { ' ', '9', '0', ' ' };
num_steps = atoi(exp1); // num_steps = 360
num_steps = atoi(exp2); // num_steps = 18
num_steps = atoi(exp3); // num_steps = 90
num_steps = atoi(exp4); // num_steps = 90
```

Exercise 4 (challenge): Dual Direction

Expand upon part 3 of the lab by allowing the user to decide the direction of the rotation

Hints:

- Instead of pressing '#' to submit the angle, press either 'A' or 'B'. If 'A' is pressed, rotate counter-clockwise. If 'B' is pressed, rotate clockwise.

Video Demonstration: http://youtu.be/DcfUAzxuM3k

Exercise 5 (challenge): Reset Position

Expand upon the previous extra credit by adding a button input to rotate the motor to a starting position. For example, if the user submits ‘C’, the motor will return to the starting position

Hints:

- Have 0 represent the starting position and use a variable to keep track of the position of the motor. Use this variable as a reference to return to the starting position if the reset button is pressed.

Video Demonstration: http://youtu.be/Cv4gaP0dKD0