Stepper Motor Operation

Performance Check

______	(20) 1: IO Testing
______	(35) 2: Motor driving modes and rpm measurements
______	(35) 3: Move motor to specific angle (must complete check off 2)

Introduction

The stepper motor is a unipolar motor. The center shaft has a series of magnets mounted on it, with 32 steps for the magnetic rotor inside the motor to complete a full 360 rotation. Turning ratio is 11.25 deg for each step. In half step mode, it has 5.625deg for a full rotation. This is without the reduction gear. The rotor shaft has a gear ratio of 64:1 ratio with the motor shaft, this means 32× 64 steps for one revolution. Each step is now equal to 0.18 deg. For half step, this is 0.09deg, giving you a lot of precision.

The unipolar stepper motor has five wires and two coils with the center connections of the coils tied together, used as the power connection. Thus the name unipolar, the power always comes in on this one pole (color Red).

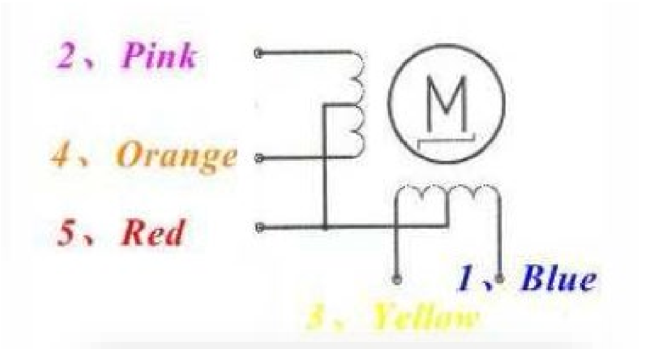


Figure 1: 28BYJ-48 - 5V Stepper Motor (Kiatronics datasheet)

To drive the stepper motor, the ULN2003 motor driver module is used. This is a high-current darlington transistor arrays, used to drive relays, stepper motors, lamps etc.

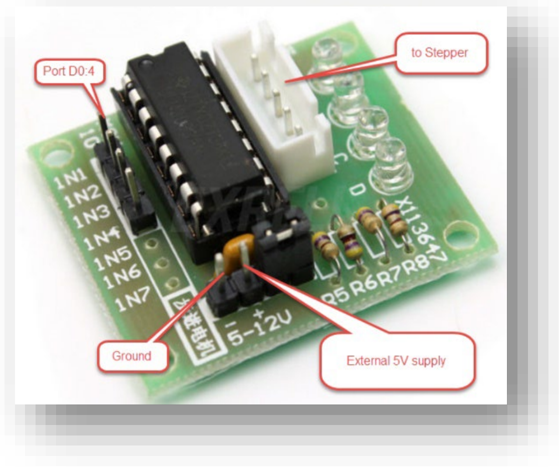


Figure 2: ULN 2003 Driver module

Wave step:

Steps	**IN4**	**IN3**	**IN2**	**IN1**	**Value**
1	0	0	0	1	**0x01**
2	0	0	1	0	**0x02**
3	0	1	0	0	**0x04**
4	1	0	0	0	**0x08**

Full step: fill in the Values (yellow cells) below, use Wave step as an example, where IN 1 = bit 0, IN 4 = bit 3

Steps	**IN4**	**IN3**	**IN2**	**IN1**	**Value**
1	0	0	1	1	🟨
2	0	1	1	0	🟨
3	1	1	0	0	🟨
4	1	0	0	1	🟨

Half step:

Steps	**IN4**	**IN3**	**IN2**	**IN1**	**Value**
1	1	0	0	1	🟨
2	0	0	0	1	🟨
3	0	0	1	1	🟨
4	0	0	1	0	🟨
5	0	1	1	0	🟨
6	0	1	0	0	🟨
7	1	1	0	0	🟨
8	1	0	0	0	🟨

Calculation:

Wave and Full Step mode:

- each step rotates 0.175°
- number of steps needed to rotate one revolution = $\frac{360°}{0.176°/step} = 2048$
- number of elements inside an array = 4
- theoretical time it takes to rotate one revolution= $rev_{time} = 2048 \times interstep_{time}$
- rpm = $\frac{1rev}{revtime} \times \frac{60sec}{1min}$

Half Step mode:

- each step rotates 0.09°
- number of steps needed to rotate one revolution = $\frac{360°}{0.09°/step} = 4096$
- number of elements inside an array = 8
- theoretical time it takes to rotate one revolution= $rev_{time} = 4096 \times interstep_{time}$

Objectives

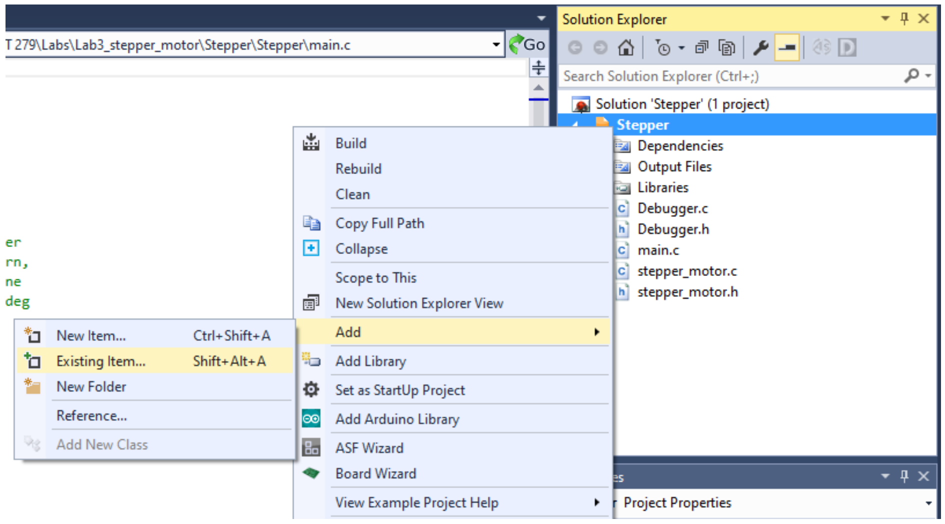
The purpose of this lab is to practice control of a stepper motor board using different drive modes and start the habit of ensuring the wirings are correct before testing the operation.

Procedure 1: Setting up the project

1. Determine hardware connection for stepper motor and the control switches.
   - Hardware connection: Fill in where you'd like to connect the IOs. In the blank below, fill in e.g. SW0 à PINA.0

Outputs (Stepper Motor Board)	**Inputs (your light and switch board)**
IN1 _____________	SW0 (Wave step mode) _____________
IN2 _____________	SW1 (Full step mode) _____________
IN3 _____________	SW2 (Half step mode)_____________
IN4 _____________	SW3 (Angle) ______________

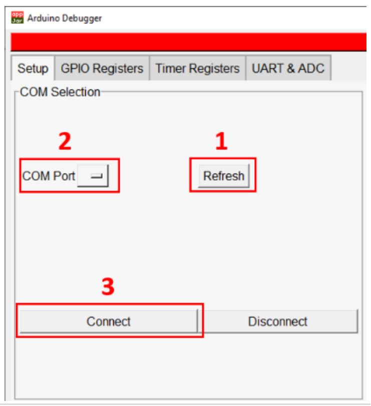
2. Download the ATMega Debugger 1.2 zip file from Brightspace.
3. Unzip the file and store at a location you choose, please ensure you keep both the 'Serial.exe' and 'appJar.ini' in the same folder. You will be using these files most of the labs this semester.
4. Follow the instruction in the **first lab** to start a new project in Microchip Studio
5. The debugger will be used to verify IO register configuration at the start of each lab, it can also be used to view Timer, UART, and ADC register configuration that will be used in later labs.
6. **Adding the Debugger to your project**
   - Copy the 'Debugger.c' and 'Debugger.h' files into the same folder as your project 'Main.c' file.
   - **Right click** on your Project Name in the Solution Explorer, select **Add**, then select **Existing Item…** (or press Shift + Alt + A)



   - Select the 'Debugger.c' and 'Debugger.h' files and click **Add**
   - In 'Main.c' add the following line between **#include <avr/io.h>** and **int main(void)**:

     #include "Debugger.h"
   - Inside int main(void), but outside the while(1) loop, add the following line:

     initDebug();
   - To see the register values once the program is running, open '**Serial.exe**'. In the Setup tab, click **Refresh**, then select the **COM port** your ATMega 2560 is connected to. Click **Connect**. Both connection status bars should change to green.
   - General Purpose IO Register values can be seen in the **GPIO Registers** tab.



7. Add an IO_init function in the main.c file
8. Set up the IO according to the table you established in step 1
9. Connect your Pushbutton inputs, and ensure the correct pins turn green in the debugger
10. In the main function, add the following line of code: Substitute INPUT_PORT with your input port (excluding the brackets).
    PORT[OUTPUT_PORT] = 0x55;

Instructor Check off 1

1. Show IO Table
2. Demonstrate pushbuttons work with the debugger
3. Demonstrate the Stepper Motor LEDs match the Debugger

Procedure 2: Setting up the project

11. Create a project called **Stepper** and write a .C program to run the stepper motor in any one of three modes namely:
    - Wave step mode
    - Full step mode
    - Half step mode

    - Output to the stepper motor should be included in the stepper_motor module. This module includes
      - Initialization for stepper motor outputs
      - Functions that control the motor using various modes (reference below)
    - Output to the L&S board is mainly used for troubleshooting purposes and control of the stepper motor.
12. Operation:
    - When SW0 is pressed, motor will rotate 2 revolutions using Wave step mode
    - When SW1 is pressed, motor will rotate 2 revolutions using Full step mode
    - When SW2 is pressed, motor will rotate 2 revolutions using Half step mode

    Your code must include three **arrays** with data corresponding to the bit pattern necessary to move the stepper motor in each of the three modes above. Two of the arrays will have 4 elements and the third will have 8. See class notes for details. You may name these arrays Full [4], Wave [4], and Half [8]. Elements inside these arrays correspond to the steps shown in above tables.

    Your code will also include a "for loop" that drives the motor for one revolution. This will make the motor turn one revolution each time the motor runs. You may change this number of iterations later during lab.

13. If using LED lights to troubleshoot, the delay must be long enough for you to see. Therefore, use a 300mS delay between each step. Use the system delay function _delay_ms(); Use the LEDs on the driver board to observe the outputs.
14. Include header comments and hardware descriptions in your .c file.
15. You must use a function to run the stepper motor. Do not hard code each step.
16. When using the simulator, comment out the _delay_ms() lines, or set breakpoints to step out of these delays. Make sure to set the optimization to 00 for no optimization.
17. We are making a multi-module program. You will make another module (.c+.h) with the function of driving

the stepper motor for different parameters.

How to make a module (refer to these steps in later labs):

18. We will add a module called stepper_motor. **A module includes two files: .c and .h. They have the same name, but different extensions (.c and .h)**
    - In the Solution Explorer window, right click your project
    - Select **Add**
    - Select **New Item**
    - See the picture below

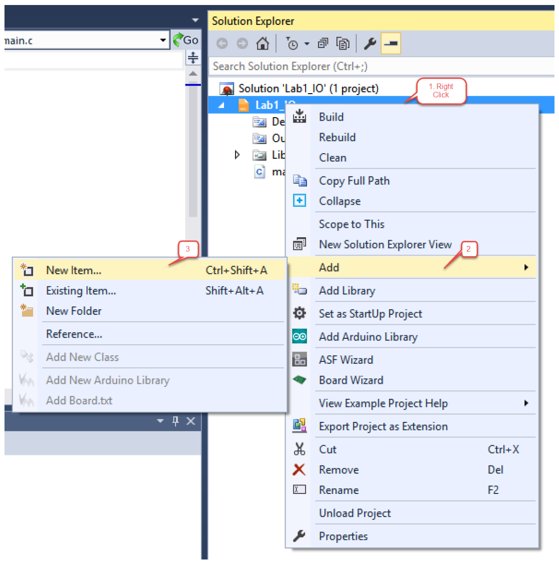
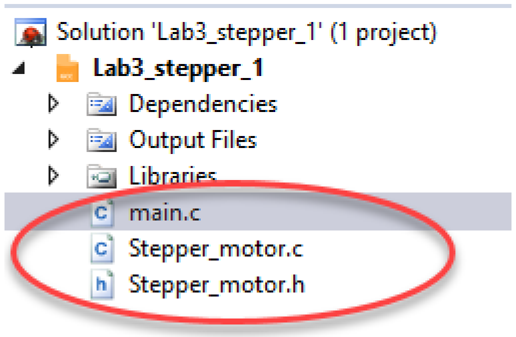


Figure 3: Add File to project screenshot

19. Select **C File**, name the file as stepper_motor.c, click the Add button to add this .c file
20. You should see the new c file stepper_motor.c added in this project.
21. Add another New Item, select **Include File**, name it stepper_motor.h, then click the Add button.
22. You should have stepper_motor.h file added in this project.
23. The .c and .h file that you just created is a **module.** A multi-module program contains multiple modules.
24. The Solution Explorer should now look like this:



25. We need to include this new module in the main.c file.

    Add this line in the main.c file, between the #include <avr/io.h> and the main function:

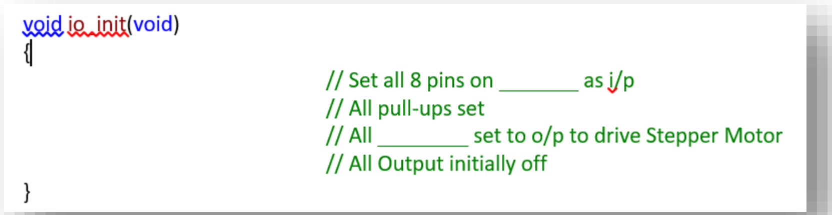
    #include “stepper_motor.h”

26. Below skeleton of the code for your reference, comments are left out on purpose, make sure you put meaningful comments in every line, this is not all the code and it is not all syntactically correct, you will need to understand it to find out what's missing. The variable n is the number of revolutions. Review lecture 4 to make sure the codes are located at the correct location of the project.

```
/*
 * Name_of_your_file.c
 *
 * Created: Date and time of your file
 * Author : Your name
 *
 *
 * Project name: Three_Modes_Stepper
 * Description:
 * Control stepper motor thru three different modes:
 * Wave step mode
 * Full step mode
 * Half step mode
 *
 * Three arrays with data corresponding to the bit
 * pattern necessary to move the stepper motor.  Code
 * has a for loop in either 4 or 8 iterations depending
 * on the drive mode for the stepper
 *
 * Use system delay function _delay_ms(3) Delay 3mS in between
 * each step (below 1ms won't work)
 *
 *      Hardware: (output) Active high / low
 *      IN1 ---- __________
 *      IN2 ---- __________
 *      IN3 ---- __________
 *      IN4 ---- __________
 *
 *      Input: how were they connected?
 *      SW0 ---- __________      WaveStep mode
 *      SW1 ---- __________      FullStep mode
 *      SW2 ---- __________      HalfStep mode
 *
 *      The code is written as a multi module program
 *

*****************************************************/
```

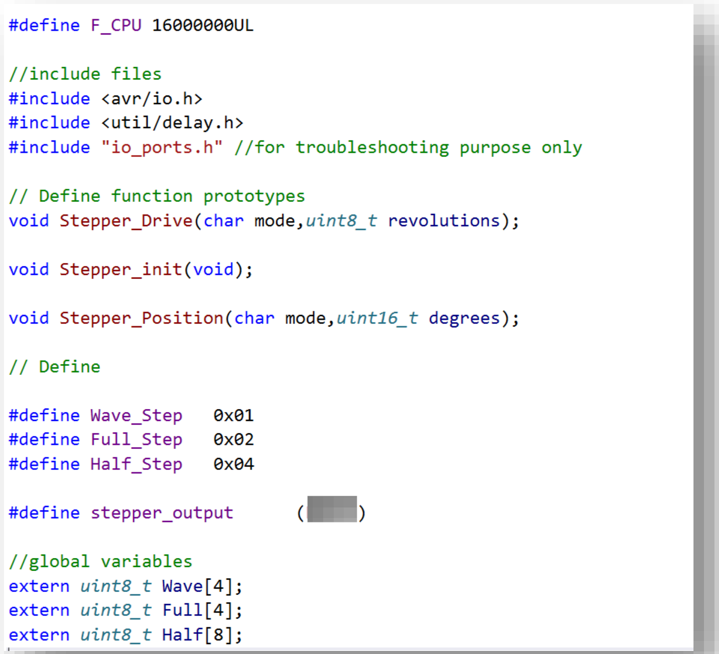
```
while (1)
{
    switch (PushButton)
    {
        case Wave_Step:     //if SW0 is pressed, start Wave Step mode
        {
            Stepper_Drive('W',n);
            break;
        }
        case Full_Step:     //if SW1 is pressed, Full Step mode
        {
            Stepper_Drive('F',n);
            break;
        }
        case Half_Step:             //if SW2 is pressed, Half Step mode
        {
            Stepper_Drive('H',n);
            break;
        }
    }

    if (~PINA & 0x08)    //position pushbutton
    {
        Stepper_Position('H',angle);
    }
    PORTD = 0x00;    //turn off motor when complete
```

- this is located inside the main.c



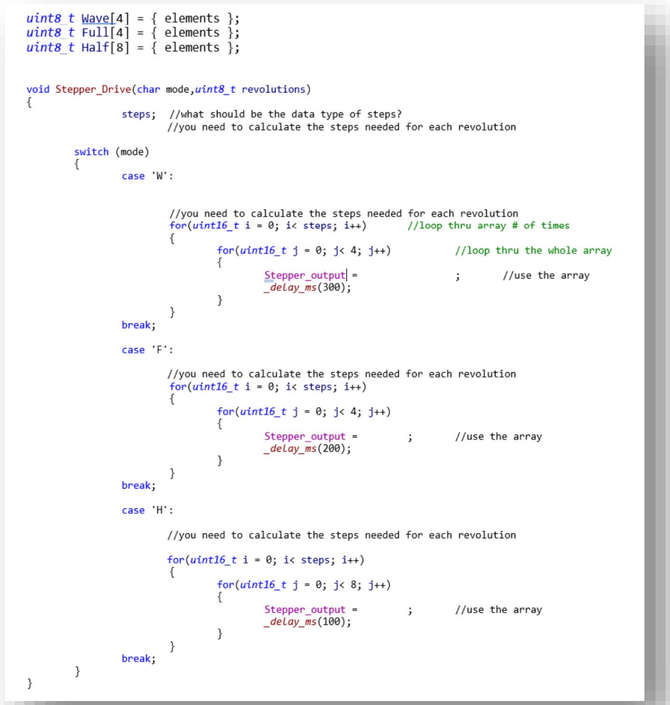
```
void io_init(void)
{
                                        // Set all 8 pins on ________ as i/p
                                        // All pull-ups set
                                        // All _________ set to o/p to drive Stepper Motor
                                        // All Output initially off
}
```

- Inside the stepper_motor.h file, #define and function prototypes of the motor. The stepper_output represents the output port you defined in table in procedure 1



```
#define F_CPU 16000000UL

//include files
#include <avr/io.h>
#include <util/delay.h>
#include "io_ports.h" //for troubleshooting purpose only

// Define function prototypes
void Stepper_Drive(char mode,uint8_t revolutions);

void Stepper_init(void);

void Stepper_Position(char mode,uint16_t degrees);

// Define

#define Wave_Step   0x01
#define Full_Step   0x02
#define Half_Step   0x04

#define stepper_output      ([redacted])

//global variables
extern uint8_t Wave[4];
extern uint8_t Full[4];
extern uint8_t Half[8];
```

- Inside the stepper_motor.c file, elements are the array, read the comments to complete the line of code. The code below has two for loops (inner and outer). The steps variable represents the number of times the inner for loops runs. E.g. if the inner for loops repeats 4 times, and the steps variable equals 10, then the total number steps the motor runs equals 4 * 10 = 40. Each time the for loop runs take 300 ms, thus to run through both for loops, it will take 300ms * 40 = 12000 ms, thus 12 seconds.



```
uint8_t Wave[4] = { elements };
uint8_t Full[4] = { elements };
uint8_t Half[8] = { elements };


void Stepper_Drive(char mode,uint8_t revolutions)
{
                steps;  //what should be the data type of steps?
                        //you need to calculate the steps needed for each revolution

        switch (mode)
        {
                case 'W':


                        //you need to calculate the steps needed for each revolution
                        for(uint16_t i = 0; i< steps; i++)      //loop thru array # of times
                        {
                                for(uint16_t j = 0; j< 4; j++)          //loop thru the whole array
                                {
                                        Stepper_output =                ;       //use the array
                                        _delay_ms(300);
                                }
                        }
                break;

                case 'F':

                        //you need to calculate the steps needed for each revolution
                        for(uint16_t i = 0; i< steps; i++)
                        {
                                for(uint16_t j = 0; j< 4; j++)
                                {
                                        Stepper_output =        ;       //use the array
                                        _delay_ms(200);
                                }
                        }
                break;

                case 'H':

                        //you need to calculate the steps needed for each revolution

                        for(uint16_t i = 0; i< steps; i++)
                        {
                                for(uint16_t j = 0; j< 8; j++)
                                {
                                        Stepper_output =        ;       //use the array
                                        _delay_ms(100);
                                }
                        }
                break;
        }
}
```

27. Once you completed filling in blanks for the above modules and the main code, use simulator to verify its operation. Remember to comment out the delay code.

Procedure 3: Three drive modes and rpm calculations

1. Modify the code to perform the following. Delay time change from X00ms to specified timing according to the steps below. You may use a timer to measure the rpm, this rpm will be used to compare with the expected (calculated) value
2. **WAVE STEP mode:**

   Connect pins IN1, IN2, IN3 and IN4 on the stepper motor board to the pins 0 thru 3 respectively of PORT____, connect 5V and ground from your **arduino** to the stepper motor module.

   - Set the for loop count to run **2** revolutions and set the delay to 3ms. When SW0 is pressed, motor will turn for 2 revolutions. Build and download your code to drive the motor in **WAVE STEP** mode.

   RPM calculation:

   - Use the delay time (3ms) in your code along with the number of steps per revolution in Wave STEP mode to calculate the **<u>expected</u>** RPM

     Show your work:
     Expected motor speed (RPM) =
     Measured motor speed (RPM) =
     Demonstrate motor operations to instructor

3. **FULL STEP mode:**

   Leave the delay at 3ms and the SW to drive the motor in **FULL STEP** mode.

   RPM calculation:

   - Use the delay time (3ms) in your code along with the number of steps per revolution in FULL STEP mode to calculate the **<u>expected</u>** RPM

     Show your work:
     Expected motor speed (RPM) =
     Measured motor speed (RPM) =
     Demonstrate motor operations to instructor

4. **HALF STEP mode:**

   - Leave the delay at 3mS and press the SW to drive the motor in **Half STEP** mode.
   - Check the operation and calculate and compare the speed in RPM as before using the two methods.

   RPM calculation:

   - Use the delay time (3mS) in your code along with the number of steps per revolution in HALF STEP mode to calculate the RPM

     Show your work:
     Expected motor speed (RPM) =
     Measured motor speed (RPM) =
     Demonstrate motor operations to instructor

Instructor Check off 2

4. Explain code operations, steps calculations correct
5. How rpm calculations (both measured and expected values)
6. Show motor movements from the code with revolution = 2, with the correct inputs.

Procedure 3: Control position

You must complete previous steps to receive scores from this step

1. Add another function to your project stepper_motor module. **Not another module.**
2. This function should be called stepper_Position
3. Make sure you add the function prototype in the Stepper_motor.h file.
4. Write a function to drive the stepper motor to xx deg (or something specify by your instructor) from its current position. When you write this code, think about how to make the change easily. It is most appropriate to use a #define to specify the angle, e.g. place this line of code: #define Angle 90 in the stepper_motor.h file. Then you can change the angle in the #define statement, instead of searching the line of code that you need to change.
   - Use the function you created from previous steps, modify the revolution to number of degrees. Be careful to choose the data type, number of degrees can vary from 0° to 360°. Write the equation in the code. You may **not** use the float data type.
   - Use the step mode of your choice to drive the motor
   - Operation: **Add another input (SW3)**, when SW3 is pressed, the stepper motor turns to the specified position using the corresponding mode.

Instructor Check off 3

1. Explain code operations, steps formula correct
2. Instructor provide step angle, student change appropriate #define line, demonstrate operation with hardware.