Laboratory 04 – UART & Stepper Motors

By: **Caleb Probst**

Spring 2023 ECE 381 Microcontrollers

Introduction

In this laboratory, the main concept was using the Universal Asynchronous Receiver/Transmitter (UART) as well as controlling stepper motors. To show this concept, we made a simple command line interface that would be able to interface from the device we were controlling it with to the PSoC. For communication between the two devices, we used the program Realterm (https://sourceforge.net/projects/realterm/) which is a program dswhich uses the USB interface in order to connect to a serial device. The use of this device has 3 commands: ‘S’, ‘F’, and ‘B’ which controls the speed, forward steps, and backwards step, respectively.

Goal

The goal of this laboratory is to recreate a motor driver that is controlled via UART. To do this, we first need to be able to communicate to our external device (laptop, computer, etc.) through a serial connection. Next, we need to be able to rotate our stepper motor both clockwise and counterclockwise. For our purposes, we used full step drive, which the table for the leads of the motor are given below in *Figure 1*.

	Step Sequence			
Coil Lead	**0**	**1**	**2**	**3**
$A(Blue)$	1	0	0	1
$B(Pink)$	1	1	0	0
$A(Yellow)$	0	1	1	0
$B(Orange)$	0	0	1	1

Figure 1: Full Step Drive Step Sequence

Next, the program should be able to take in the commands ‘S’, ‘F’, or ‘B’ and run their respective commands. ‘S’ should be able to take in a floating point RPM value for the motor and calculate the delay based on that RPM value. ‘F’ should be able to take in an integer value for the number of steps forward taken, and should run the motor forward for the required amount of steps. ‘B’ should do the same as ‘F’ except backwards. Finally, the program should echo both the RPM value and calculated delay for each ‘S’ command input, and should echo the amount of steps taken for the ‘F’ and ‘B’ commands after the steps have been taken.

Materials

PsoC4 4200M Microcontroller	1
28BYJ-48 Stepper Motor	1
ULN2003A Driver Board	1

Procedure and Results

Part 1 – Setup in PSoC Creator

Figure 2 shows the Top Design setup for the laboratory. The setup is very simple and only requires the UART module and a control register. Ensure that the mode is “Standard”, direction is both TX and RX, Baud rate is 57600, data bits set to 8 bits, parity is none, and stop bits is 1. For the control register, ensure that it is a 4 bit register with all modes set to direct and all initial values to 0. Then connect 4 output pins PhsAPos, PhsANeg, PhsBPos and PhsBNeg to the control register.

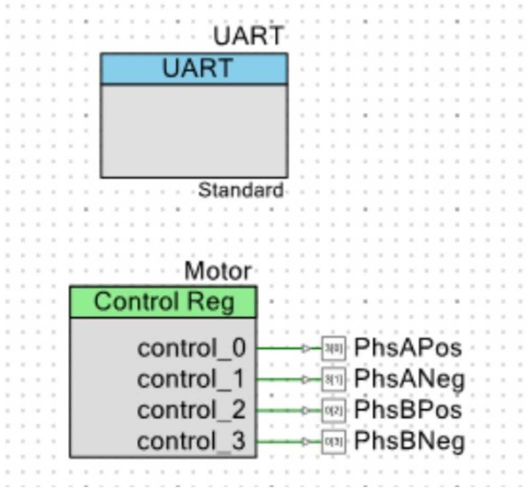


Figure 2: Top Design

UART:rx	P7[0]
UART:tx	P7[1]
PhsANeg	P3[1]
PhsAPos	P3[0]

PhsBNeg	P0[3]
PhsBPos	P0[2]

Table 1: Pin Outs of Hardware on PSoC4

Part 2 – Hardware and Wiring

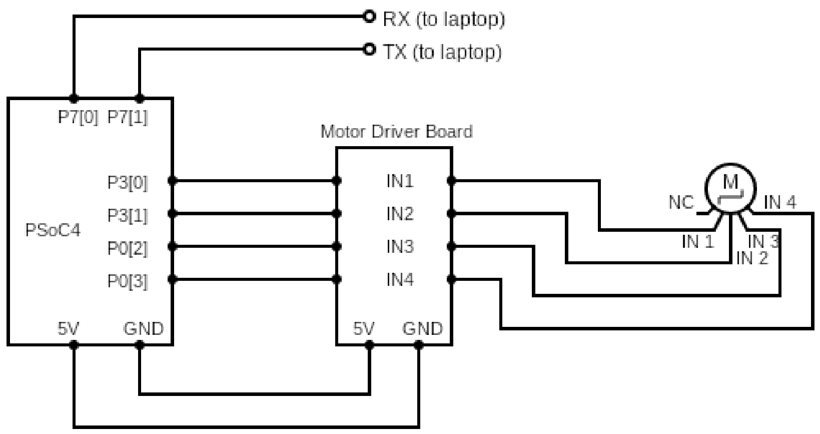


Figure 3: Wiring Diagram

The wiring of this diagram is very simple. We only need a motor, the driver board, and the wire for the UART connection. For the motor, we used a 28BYJ-48 Stepper Motor, with a ULN2003A Driver Board. The driver board only had a voltage that could be driven in between 5V and 12V, and the input pins for the motor. Then, the motor connected directly to the board.

Part 3 – Software

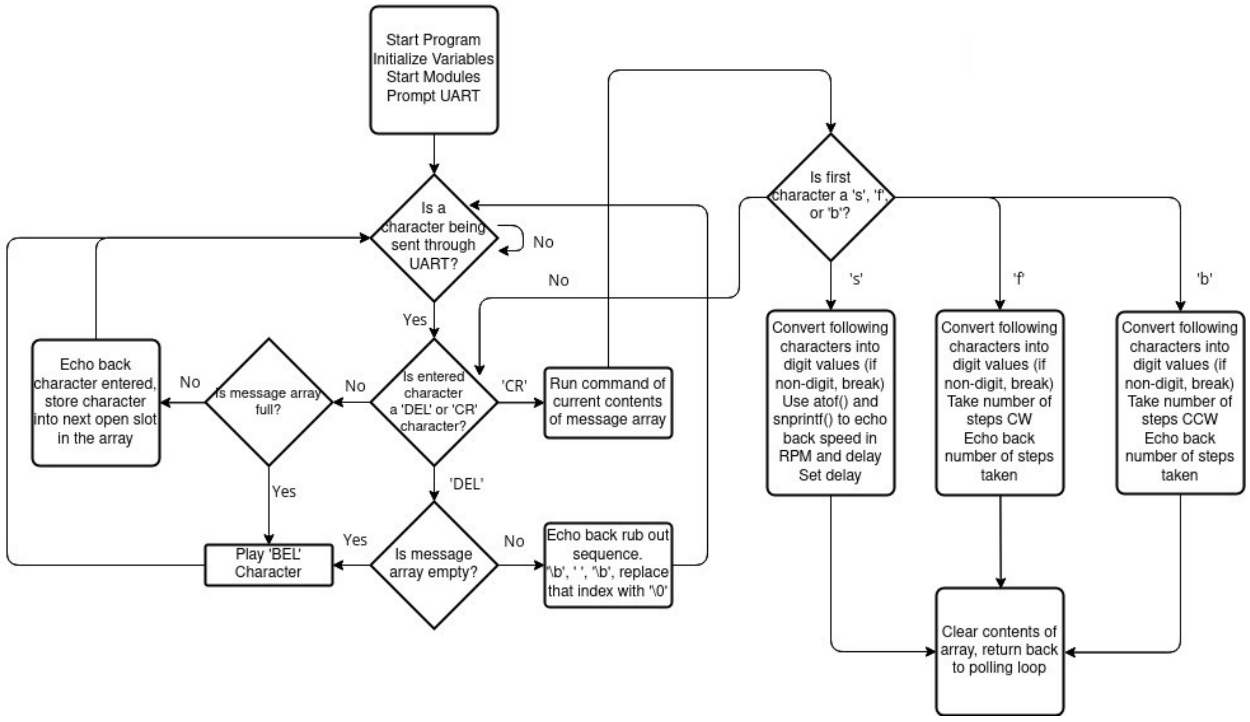

Start Program
Initialize Variables
Start Modules
Prompt UART
Is a character being sent through UART?
No
Yes
Is entered character a 'DEL' or 'CR' character?
'CR'
Run command of current contents of message array
No
Is message array full?
No
Echo back character entered, store character into next open slot in the array
Yes
'DEL'
Is message array empty?
Yes
Play 'BEL' Character
No
Echo back rub out sequence. '\b', ' ', '\b', replace that index with '\0'
Is first character a 's', 'f', or 'b'?
No
's'
Convert following characters into digit values (if non-digit, break) Use atof() and snprintf() to echo back speed in RPM and delay Set delay
'f'
Convert following characters into digit values (if non-digit, break) Take number of steps CW Echo back number of steps taken
'b'
Convert following characters into digit values (if non-digit, break) Take number of steps CCW Echo back number of steps taken
Clear contents of array, return back to polling loop

Figure 4: Flowchart of Program

The software of this laboratory contributes almost entirely to the operation of the laboratory. All code can be found in the Appendix, under *Main Code*. First, the program initializes all starting variables, such as the array for storing the command and the index where the next character needs to be stored. Next, it starts the UART module and prompts the user for input. The three valid commands are 's' for controlling the speed, 'f' for moving forward (Clockwise), and 'b' for moving backward (Counter-Clockwise). It then polls the UART, waiting for user input. If there is user input, it first checks if its a 'DEL' character (given by hex value 0x7F) or a 'CR' character (given by hex value 0x0D). If it is a 'DEL', we need to be able to "rub out" the characters to make it look clean. To do this, we first check if the index is 0, meaning there has been no input yet. If there is no input, we play the 'BEL' character, indicating that they cannot do that action. If the index is not 0, we echo the sequence '\b', ' ', '\b'. What this does is it backspaces, replaces the last character with a space, then backspaces that. This effectively does a backspace in normal command lines. If the character is neither a carriage return or backspace, we need to first check if the index is the maximum (16). If this is true, we play the 'BEL' character and don't do anything. If it is not at the maximum, we simply store the character into the array and then increment index for the next character. If the character is a carriage return, we need to run the command that is currently stored in the command character array. This is done by the function `runCommand()`. For the run command, we have three separate commands, 's', 'f', and 'b'. For the 's' command, we check for both a capital and lowercase, and then get the values after the 's' character. This can be a floating point, so we also need to beware of that. If any of the characters after 's' is non-digit or not a decimal, we simply break. We then use the `atof()` function to first convert the string into a floating point value so we can operate on it, and then the `snprintf()` will convert it back to a string so we can output it. Based on the floating point value, which symbolizes the RPM of the motor, we change the delay based on the following function.

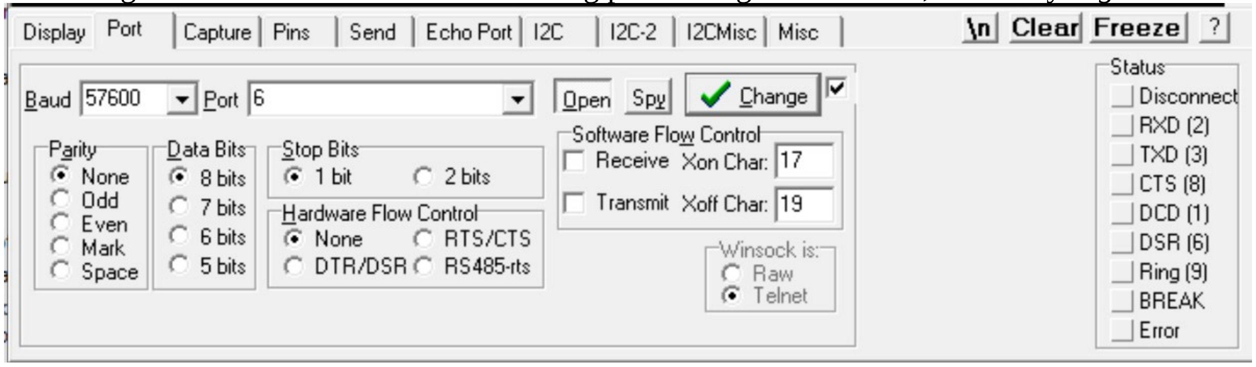
$$Delay = (\frac{1}{2048})(\frac{1}{rpm})(6 \times 10^7)$$

Equation 1: Calculating Delay from RPM

Then, delay is a global variable that gets changed when this command is ran. For the ‘f’ and ‘b’ command, we want to be able to move the motor either clockwise or counter-clockwise, respectively, a certain amount of steps. The arguments for these two commands are both integers so we simply have to use the `atoi()` function to convert the string into a integer. For these commands, we did not use the built-in `atoi()` function, but instead made our own. After we get the number of steps, we then take the number of steps using a simple for loop and either running the command `moveCW()` or `moveCCW()` based on if it is either ‘f’ or ‘b’. After taking the steps, we use the `snprintf()` function to echo the number of steps taken.

Testing Methodology

For the testing of this device, we used Realterm for our serial connection software, but you can use any software that can talk to a UART device through USB. First, we need to check that all the settings are correct. We used the following port settings in Realterm, shown by *Figure 5*.



Baud 57600 Port 6

Parity: [✓ None] Odd Even Mark Space

Data Bits: [✓ 8 bits] 7 bits 6 bits 5 bits

Stop Bits: [✓ 1 bit] 2 bits

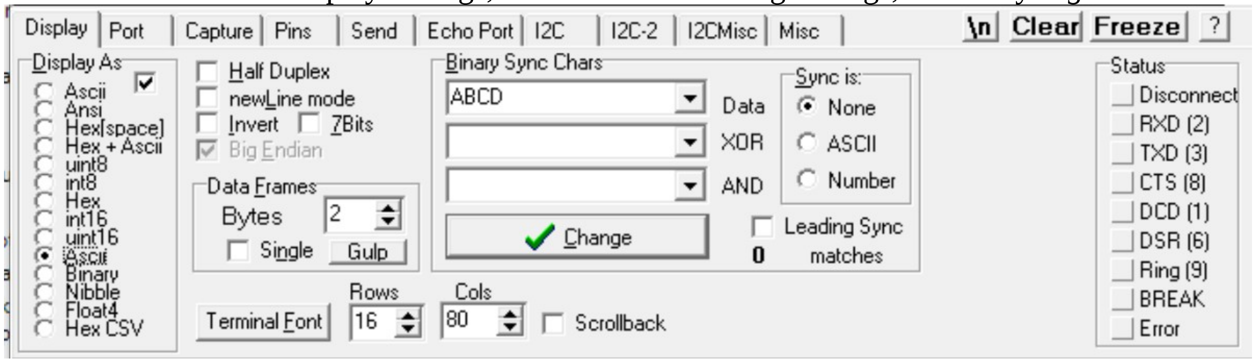
Hardware Flow Control: [✓ None] RTS/CTS DTR/DSR RS485-rts

Software Flow Control: Receive Xon Char: 17 Transmit Xoff Char: 19

Winsock is: Raw [✓ Telnet]

Figure 5: Port Settings in Realterm

For the port number, it depends on what port you are using for the PSoC. It should show up as USBSER###. For the display settings, we used the following settings, shown by *Figure 6.*



Display As: Ascii Ansi Hex[space] Hex + Ascii uint8 int8 Hex int16 uint16 [✓ Ascii] Binary Nibble Float4 Hex CSV

Half Duplex, newLine mode, Invert, 7Bits, [✓ Big Endian]

Data Frames: Bytes 2, Single, Gulp

Binary Sync Chars: Data ABCD, XOR, AND

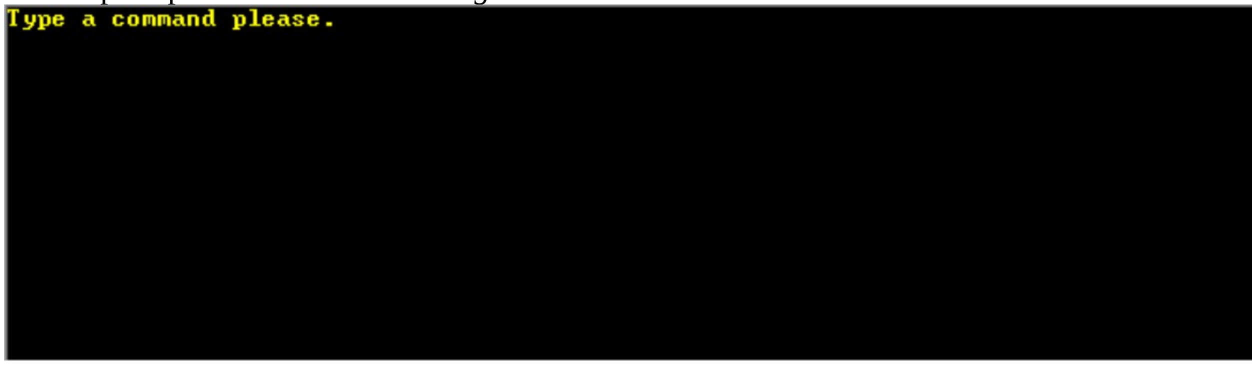
Sync is: [✓ None] ASCII Number

Leading Sync 0 matches

Rows 16 Cols 80 Scrollback

Figure 6: Display Settings in Realterm

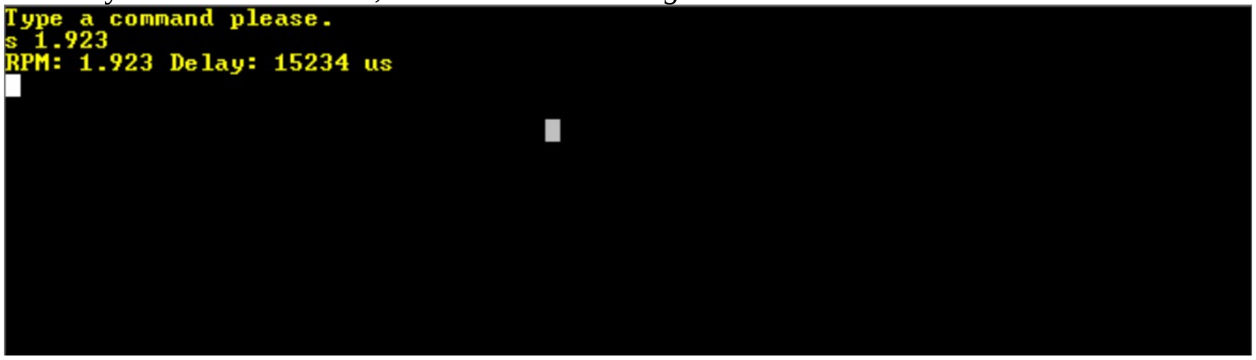
The first thing to test is the general output of the UART. Upon starting the program, we should see the prompt as shown below in *Figure 7*.



```
Type a command please.
```

Figure 7: UART Prompt

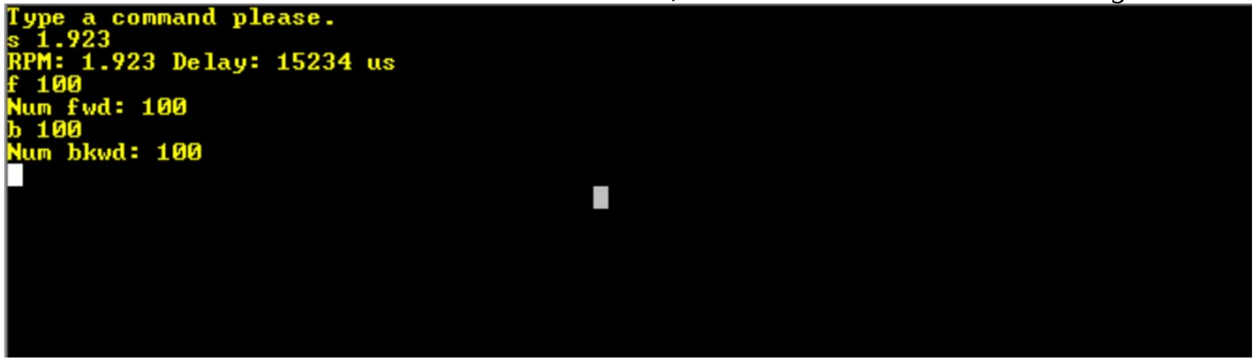
From this, we can see that our UART device is working. We can now test the backspace, and it should play the 'BEL' sequence. If we type over 16 characters, it should also play the 'BEL' character. Now, if we type the 's' command with a floating point value, we should get an echo of the delay and the RPM value, as shown below in *Figure 8*.



```
Type a command please.
s 1.923
RPM: 1.923 Delay: 15234 us
```

Figure 8: Speed Command

Now we can test forward and backwards commands, these are both shown below in *Figure 9*.



```
Type a command please.
s 1.923
RPM: 1.923 Delay: 15234 us
f 100
Num fwd: 100
b 100
Num bkwd: 100
```

Figure 9: Forward and Backward Command

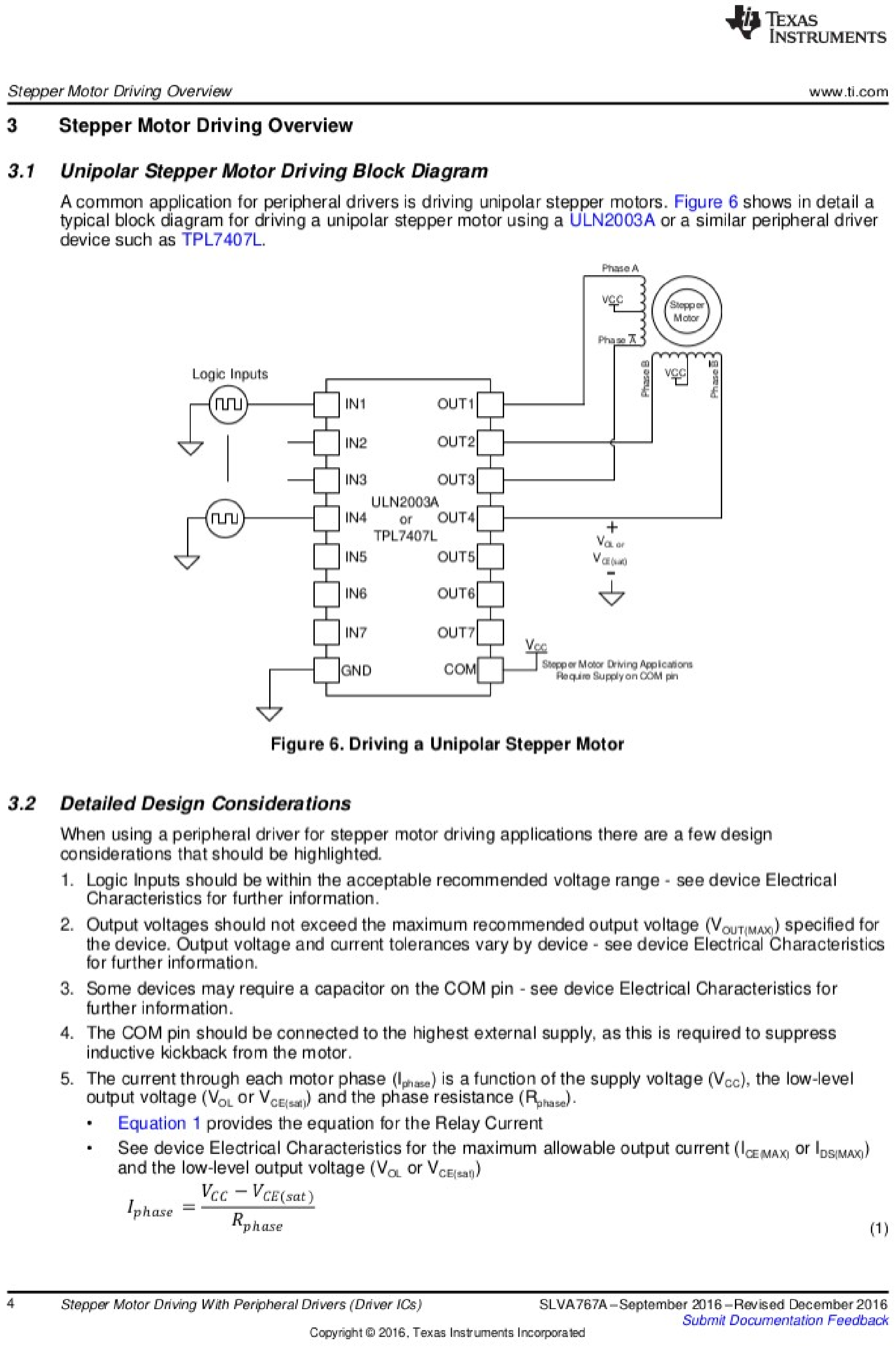
To test the actual motor itself, set the speed at 1 RPM and move it forward a large amount of steps. If we time it from when it starts to where it ends up in the same position, it should roughly be 1 minute. This is a good indicator whether or not the function is actually working properly.

Conclusion

This laboratory was relatively straightforward. We used a generic struct for our motor and was able to do it efficiently with the `moveCW()` and `moveCCW()` commands. The only thing that we had issues with was why we didn't use the built in C `atoi()` function, but since we were able to make our own, we didn't have much issues. Other than that, we didn't have any issues and the laboratory went very well.

Appendix

ULN2003A



Texas Instruments

3 Stepper Motor Driving Overview

3.1 Unipolar Stepper Motor Driving Block Diagram

A common application for peripheral drivers is driving unipolar stepper motors. Figure 6 shows in detail a typical block diagram for driving a unipolar stepper motor using a ULN2003A or a similar peripheral driver device such as TPL7407L.

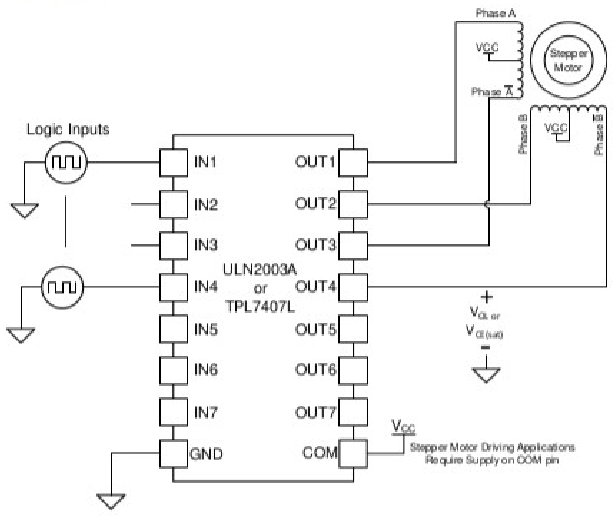


Figure 6. Driving a Unipolar Stepper Motor

3.2 Detailed Design Considerations

When using a peripheral driver for stepper motor driving applications there are a few design considerations that should be highlighted.

1. Logic Inputs should be within the acceptable recommended voltage range - see device Electrical Characteristics for further information.
2. Output voltages should not exceed the maximum recommended output voltage ($V_{OUT(MAX)}$) specified for the device. Output voltage and current tolerances vary by device - see device Electrical Characteristics for further information.
3. Some devices may require a capacitor on the COM pin - see device Electrical Characteristics for further information.
4. The COM pin should be connected to the highest external supply, as this is required to suppress inductive kickback from the motor.
5. The current through each motor phase (I_{phase}) is a function of the supply voltage (V_{CC}), the low-level output voltage (V_{OL} or $V_{CE(sat)}$) and the phase resistance (R_{phase}).
   - Equation 1 provides the equation for the Relay Current
   - See device Electrical Characteristics for the maximum allowable output current ($I_{CE(MAX)}$ or $I_{DS(MAX)}$) and the low-level output voltage (V_{OL} or $V_{CE(sat)}$)

$$I_{phase} = \frac{V_{CC} - V_{CE(sat)}}{R_{phase}} \quad (1)$$

28BYJ-48

Unipolar stepper motors come in three primary wiring configurations, 5-wire, 6-wire, and 8-wire, where 5-wire and 6-wire stepper motors are the most common. In a 5-wire stepper motor the center tap connections are shorted together internally as shown in Figure 2. In a 6-wire stepper motor the center tap connections are separate connections as shown in Figure 3. A 6-wire stepper motor can effectively act as a 5-wire stepper motor by connecting the two center tap wires.

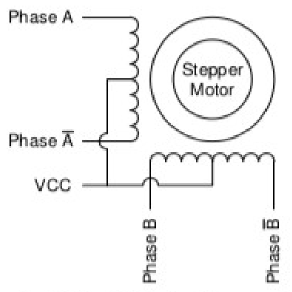


Figure 2. 5-Wire Unipolar Stepper Motor

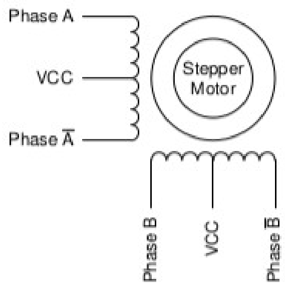


Figure 3. 6-Wire Unipolar Stepper Motor

2.2 *Bipolar Stepper Motors*

Bipolar stepper motors require both a low-side driver and a high-side driver (see Figure 4). This allows the coils to be biased in both directions, requiring two Half-H drivers (one Full-H bridge). The L293D is an example of a device that can drive these types of stepper motors, see Section 3.1 for additional information.

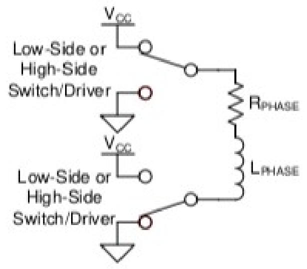

V_{CC}
Low-Side or High-Side Switch/Driver
R_{PHASE}
V_{CC}
L_{PHASE}
Low-Side or High-Side Switch/Driver

Figure 4. Variable Low-Side and High-Side Switches/Drivers

Bipolar stepper motors only come in a 4-wire configuration and do not have center tap connections. See Figure 5 for the wiring of the bipolar stepper motor.

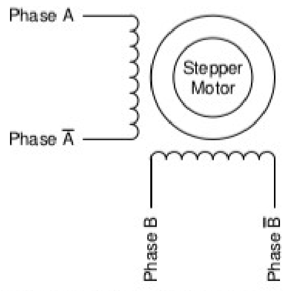


Figure 5. 4-Wire Bipolar Stepper Motor

Main Code

```
#include "project.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

// Create a struct for a motor, with a variable for position
struct MTR{
	unsigned int position : 4;
};
// Initialize motor with position 1
struct MTR motor = {1};
// Initialize motor delay for 3000 microseconds
int motorDelay = 3000;
// Function for moving counter-clockwise
int MVCCW(struct MTR motor){
	// If motor position is at edge, reset, otherwise increment
	if(motor.position ≥ 4){
		motor.position = 1;
	}
	else{
		motor.position += 1;
	}
	// Write correct motor position
	switch(motor.position){
		case 1:
			// 0011 → 1001
			Motor_Write(0x9);
			break;
		case 2:
			// 1001 → 1100
			Motor_Write(0xC);
			break;
		case 3:
			// 1100 → 0110
			Motor_Write(0x6);
			break;
		case 4:
			// 0110 → 0011
			Motor_Write(0x3);
			break;
	}
	// Delay defined by either default or RPM
	CyDelayUs(motorDelay);
	// Ground so motor doesn't get hot
	// Can change to where it only goes to ground only after all steps
have been taken
	Motor_Write(0b0000);
	// Return position of motor
	return motor.position;
}
// Function for moving clockwise
int MVCW(struct MTR motor){
	// If motor is at edge, reset, if not, decrement
	if(motor.position ≤ 1){
```

```
            motor.position = 4;
        }
        else{
            motor.position -= 1;
        }
        // Write correct motor position
        switch(motor.position){
            case 4:
                // 1100 -> 1001
                Motor_Write(0x9);
                break;
            case 3:
                // 1001 -> 0011
                Motor_Write(0x3);
                break;
            case 2:
                // 0011 -> 0110
                Motor_Write(0x6);
                break;
            case 1:
                // 0110 -> 1100
                Motor_Write(0xC);
                break;
        }
        // Delay
        CyDelayUs(motorDelay);
        // Ground so motor doesn't get hot
        Motor_Write(0b0000);
        // Return motor position
        return motor.position;
    }
    // Function that runs commands based on the letter given
    void runCommand(char* command){
        // Run through the character array
        for(int i = 0; i < 16; i++){
            // Change speed (floating point)
            if(*(command+i) == 's' || *(command+i) == 'S'){
                // New character array for value
                char value[14] = {};
                // Store last index variable
                int lastIndex;
                // Loop through the array starting at 2
                for(int j = 2; j < 16; j++){
                    // If null terminator, break and increment index by 2
                    if(*(command+j) == '\0'){
                        lastIndex = j+2;
                        break;
                    }
                    // Store character into value array
                    value[j-2] = *(command + j);
                }
                // Convert string to a floating point
                float floatValue = atof(value);
                char out[16];
                // Convert floating point value back to string using
    formatting
```

```
            snprintf(out, sizeof(out),"%.3f",floatValue);
            // 2048 steps/rev
            // rev/min -> rev/us
            // rev/us -> us/step
            // (1/2048)*(1/rpm)*(6*10^7)
            // Calculate delay
            int delay =
(int)((1/((float)2048))*(1/floatValue)*(6*10000000));
            // Store global variable
            motorDelay = delay;
            // Output "RPM: " to UART
            char output[56]={'R','P','M',':',' '};
            // Go through out array and store into output array
            for(int j = 0; j < lastIndex+1; j++){
                // If end of string, add 5 to last index and break
                if(out[j] == '\0'){
                    lastIndex = j+5;
                    break;
                }
                // Store out to output
                else{
                    output[j+5] = out[j];
                }
            }
            // Store " Delay: " into output, concantenated
            output[lastIndex++] = ' ';
            output[lastIndex++] = 'D';
            output[lastIndex++] = 'e';
            output[lastIndex++] = 'l';
            output[lastIndex++] = 'a';
            output[lastIndex++] = 'y';
            output[lastIndex++] = ':';
            output[lastIndex++] = ' ';
            // New array for next format
            char out2[16];
            // Format delay as a decimal formatted string
            snprintf(out2, sizeof(out2),"%d",motorDelay);
            // New index
            int lastIndex2;
            // Loop through out2 array
            for(int j = 0; j < 16; j++){
                // If end of string, add the lastIndex + where it is
into lastIndex2
                if(out2[j] == '\0'){
                    lastIndex2 = j+lastIndex;
                    break;
                }
                // Store out2 into output
                else{
                    output[j+lastIndex] = out2[j];
                }
            }
            // Add units to the end and the carriage return
            output[lastIndex2++] = ' ';
            output[lastIndex2++] = 'u';
            output[lastIndex2++] = 's';
            output[lastIndex2++] = '\r';
            output[lastIndex2++] = '\n';
            output[lastIndex2++] = '\0';
            // Output to UART
            UART_UartPutString(output);
        }
        // Forward steps
        else if(*(command+i) == 'F' || *(command+i) == 'f'){
            // Increment by 2 to skip letters
            i +=2;
            // Initialize value
            long value = 0;
            // Go through array (this is basically atod() function)
            for(int j = i; j < 16;j++){
                // Store value as the current char value - 0x30 (takes
numbers 0-9)
                int charVal = (int)(*(command+j)-0x30);
                // If any other character other than a number, break
                if(charVal < 0 || charVal > 10){
                    break;
                }
                // Add character value then multiply by 10 to go to
next number
                else{
                    value += charVal;
                    value *= 10;
                }
            }
            // Divide by 10 to reverse the last character
            value /= 10;
            // Move motor clockwise based on number of steps given
            for (int i = 0; i < value; i++){
                 motor.position = MVCW(motor);
            }
            // Start new output array with "Num fwd: "
            char output[26] = {
                'N', 'u', 'm', ' ', 'f', 'w', 'd',':',' '
            };
            // Initialize power of 10
            int pow10 = 10;
            // Find remainder, if its the value, break, if not,
increment power of 10
            // Finds order of value
            while(value%(int)pow10 != value){
                pow10*=10;
            }
            // Start index of output array
            int startIndex = 9;
            // If the power of 10 is 1, break
            while(pow10/10 != 0){
                // Decrement power of 10
                pow10/=10;
                // Store output array as character version of number
                output[startIndex++] = (value/(pow10))+0x30;
                // Subtract that number
                value -= value/(pow10)*pow10;
            }
            // End with carriage return and new line and null
terminator
            output[startIndex] = '\r';
            output[startIndex+1] = '\n';
            output[startIndex+2] = '\0';
            // Output to UART
            UART_UartPutString(output);
        }
        // Backwards
        else if(*(command+i) == 'B' || *(command+i) == 'b'){
            // Increment by 2 to skip letters
            i +=2;
            // Initialize value
            long value = 0;
            for(int j = i; j < 16;j++){
                // Store value as the current char value - 0x30 (takes
numbers 0-9)
                int charVal = (int)(*(command+j)-0x30);
                // If any other character other than a number, break
                if(charVal < 0 || charVal > 10){
                    break;
                }
                // Add character value then multiply by 10 to go to
next number
                else{
                    value += charVal;
                    value *= 10;
                }
            }
            // Divide by 10 to reverse the last character
            value /= 10;
            // Move motor counter-clockwise based on number of steps
given
            for (int i = 0; i < value; i++){
                 motor.position = MVCCW(motor);
            }
            // Initialize array as "Num bkwd: "
            char output[27] = {
                'N', 'u', 'm', ' ', 'b', 'k', 'w', 'd',':',' '
            };
            int pow10 = 10;
            while(value%(int)pow10 != value){
                pow10*=10;
            }
            // Create start index at 10
            int startIndex = 10;
            // If the power of 10 is 1, break
            while(pow10/10 != 0){
                // Decrement power of 10
                pow10/=10;
                // Store output array as character version of number
                output[startIndex++] = (value/(pow10))+0x30;
                // Subtract that number
                value -= value/(pow10)*pow10;
            }
            // Put end of string as carriage return, new line, and
null terminator
            output[startIndex] = '\r';
            output[startIndex+1] = '\n';
            output[startIndex+2] = '\0';
            // Output UART
            UART_UartPutString(output);

        }
    }


}
int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    /* Place your initialization/startup code here (e.g.
MyInst_Start()) */
    // Start UART
    UART_Start();
    // Initialize UART
    UART_UartPutString("Type a command please.\r\n");
    // Initialize command array
    char command[17];
    // Initialize null terminator for string
    command[16] = '\0';
    // Initialize index for 0
    unsigned int index = 0;
    for(;;)
    {
        /* Place your application code here. */
        // Initialize input as nothing
        char input = 0;
        // Poll UART
        do{
            input = UART_UartGetChar();
        }
        while(input == 0);
        // Backspace character (0x7F)
        if(input == 0x7f){
            // If index is 0, play BEL
            if (index == 0){
                UART_UartPutChar(0x07);
            }
            else{
                // Backspace, replace with space, backspace again
                UART_UartPutChar('\b');
                UART_UartPutChar(' ');
                UART_UartPutChar('\b');
                // If the index is not 0 (not technically needed)
                // Replace command array at index and index-1 to null
                // Decrement index
                if(index != 0){
                    command[index] = '\0';
                    command[index-1] = '\0';
                    index--;
                }
            }
        }
        // Carriage return
        else if(input == 0x0D){
            // Output carriage return
            UART_UartPutString("\r\n");
            // Create pointer
            char* pointer = command;
            // Use run command
            runCommand(pointer);
            // Reset array
            for(int i = 0; i < 16; i++){
                command[i] = '\0';
            }
            // Null terminator
            command[16] = '\0';
            // Reset index
            index = 0;
        }
        else{
            // If index is over 16 (max character) output BEL
            if(index == 16){
                UART_UartPutChar(0x07);
            }
            // If index is not 16, store character
            else{
                UART_UartPutChar(input);
                command[index++] = input;
            }
        }
    }
}
/* [] END OF FILE */
```