Kyle Groom

CS238

2/12/2015

Lab 3 report

Purpose: To become familiar with controlling stepping motors by adapting existing code to the control of a small stepping motor with a driver board.

Set Up:



Goal: Develop C code for the TMS123 microcontroller that will home drive the stepping motor through a specified number of degrees.

This Lab requires a bit more understanding of the board and how to access the pins on it. Most of this is done for us in Stepper\_Init, which pre-initializes the four GPIO\_PORTD Pins you need, as well as setting the direction to output. At that point the only thing you need to work with is what the values for the PORTD Pins need, these are done in the data array beforehand, so the only real bit you need to do is make sure you use SysTick properly, which can be slightly difficult as it appears that some of the functions have been mislabeled/mis-commented. This, however is a simple issue to fix as there are only two options for the SysTick Timer. The work that needs to be done is the first loop shown, which rotates the Stepping Motor clockwise. The Loop right below it rotates the motor in the counterclockwise direction. What these loops are doing, which is only visible if the Wait timer is set to a very large value, is turning pins on and off in a pattern.

while(1)

{

//clockwise

for(c = 0; c < 4; c++)

{

//Change the stepper Pin Value

GPIO\_PORTD\_DATA\_R = data[c];

//wait to advance

SysTick\_Wait(100000);

}

}

The Image below shows what the cycle typically looks like. High denotes that that part is on, and low denotes that there is no current flowing through that part. Following the first while loop goes through steps 1 through 4 in the typical fashion. The Second while loop however starts at step 1, goes to step 4, then step 3, and lastly step 2 before resetting to step 1. This is how the Stepping motor’s shaft rotates.

while(1)

{

//counter clockwise

for(c = 0; c < 4; c++)

{

//Change the stepper Pin Value

GPIO\_PORTD\_DATA\_R = ~(data[c]);

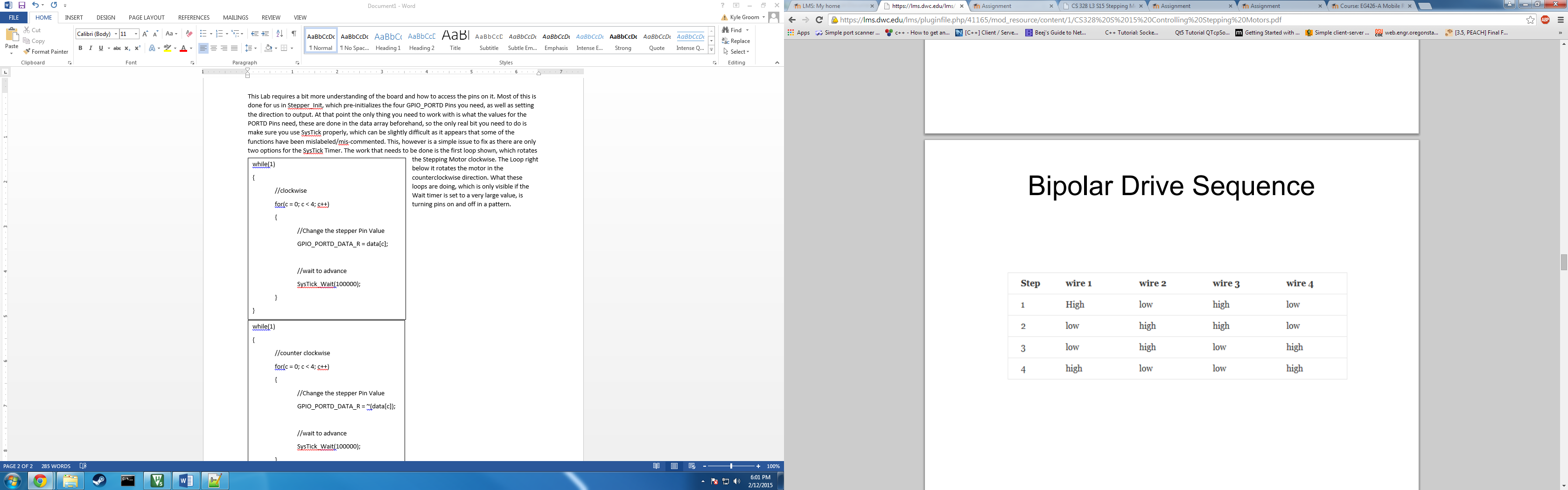
//wait to advance

SysTick\_Wait(100000);

}

}

For a slightly more visual appeal, I used some code from blinky.c and incorporated it into this so that the High/Low of a Wire was denoted by a specific LED Light on the board itself. Red was Wire1, Blue was Wire2, Purple was Wire3, and Green was Wire4. When multiple wire’s are set to high both of those corresponding LEDs are turned on. The code below the Step Wire Sequence show the clockwise and counterclockwise implementations of this respectively.



while(1)

{

//clockwise

for(c = 0; c < 4; c++)

{

//Enable the GPIO pin for the LED

GPIO\_PORTF\_DIR\_R = color[c];

//Set the direction as output, and enable the GPIO pin for digital function.

GPIO\_PORTF\_DEN\_R = color[c];

//Turn on the LED

GPIO\_PORTF\_DATA\_R |= color[c];

//Change the stepper Pin Value

GPIO\_PORTD\_DATA\_R = data[c];

//Wait To Advance

SysTick\_Wait(100000);

//Turn Off The LED

GPIO\_PORTF\_DATA\_R &= ~(GPIO\_PORTF\_DATA\_R);

}

}

while(1)

{

//counter clockwise

for(c = 0; c < 4; c++)

{

//Enable the GPIO pin for the LED

GPIO\_PORTF\_DIR\_R = color[c];

//Set the direction as output, and enable the GPIO pin for digital function.

GPIO\_PORTF\_DEN\_R = color[c];

//Turn on the LED

GPIO\_PORTF\_DATA\_R |= color[c];

//Change the stepper Pin Value

GPIO\_PORTD\_DATA\_R = ~(data[c]);

//Wait To Advance

SysTick\_Wait(100000);

//Turn Off The LED

GPIO\_PORTF\_DATA\_R &= ~(GPIO\_PORTF\_DATA\_R);

}

}