Lab 5. Stepper Motor Controller
COMP-462 Embedded Systems

Outline:

Warning: please turn off power while moving wires!!!

Warning: please place your voltmeter on the +5V line when you first turn on the circuit. If you do not get +5V, turn off power and fix the wiring mistake!!!

Warning: please connect pin 9 of the ULN2003 to +5V (this connection will prevent the motors from destroying your microcontroller and your laptop)

Preparation

Read Chapter 5 of the book and the FSM sample examples:

TableLineTracker_4C123
TableTrafficLight_4C123
StepperFSM_4C123
SysTick time delays in **SysTick_4C123**

Starter Code: Open the starter code from c:\Keil_v5\EE319KwareSpring2019\
Lab5_EE319K
Here is a video of Lab 5 solution https://youtu.be/5BIeSRukyVs

Purpose

This lab has these major objectives:

1. The understanding and implementing linked data structures
2. Learning how to create a software system
3. The study of real-time synchronization by designing a finite state machine controller

Software skills you will learn include advanced indexed addressing, linked data structures, creating fixed-time delays using the SysTick timer, and debugging real-time systems.

Design Overview

Consider a stepper interface circuit as shown below. The stepper motor will controlled by 5 output pins of the microcontroller. The ULN2003 driver must be used because the motor currents can be as high as 200mA. **Notice also pin 9 of the driver must be connected to +5V (VBUS).** With this circuit, outputs to PE4,PE3,PE2,PE1,PE0 will cause the motor to move. Each new output causes one step of the motor (4 degrees). Any five pins can be used. However, we recommend using one of the choices allowed by the simulator. You will also interface an LED to another output pin (not shown in Figure 5.1, you may use any pin for the LED interface, however we recommend using one of the choices allowed by the simulator). The user will control the windshield wiper system with two buttons. One button (**wiper**) will active just the wiper and the other button (**clean**) will active the wiper and the washer.

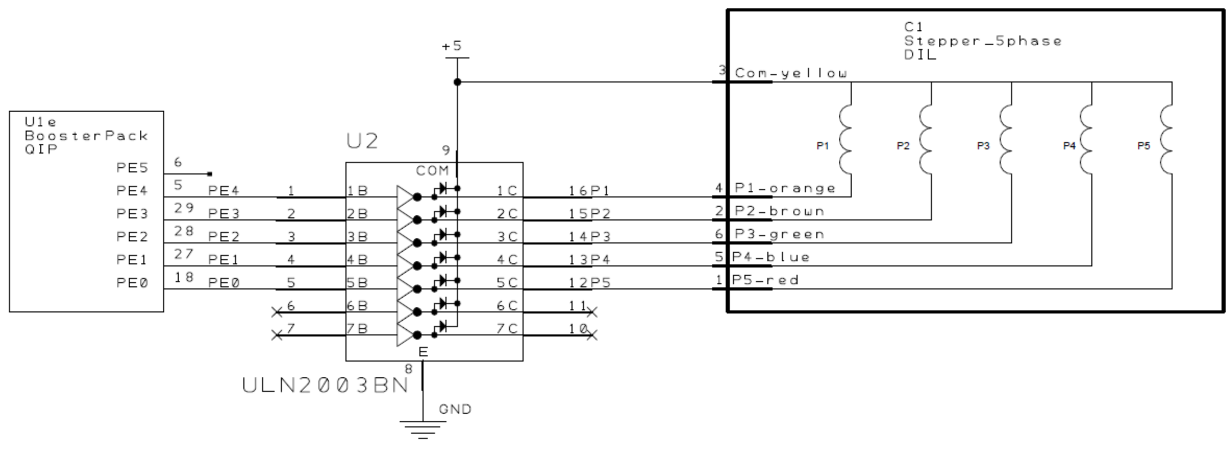


*Figure 5.1. Stepper motor (*Seagate 73192-193MCCMAS*) interface for Windshield wiper system. Switch inputs and LED output not shown.*

A wiper motion will be to step the motor 10 times in one direction and then 10 times back in the other direction; this sequence of 20 steps 1,2,4,8,16,1,2,4,8,16,8,4,2,1,16,8,4,2,1 will cause the motor to move +40 degrees forward then -40 degrees back. Each output causes the motor move exactly 4 degrees. The home position will be defined as the motor position at the time the software is started. While the wiper and/or washer input is active the wiper will move +40 degrees then -40 degrees back and forth. However, regardless of when you release the button, the controller will continue to move the motor so it stops at the home position.

- INPUTS - 2
    1. ***Wiper:*** The operator uses this button to activate the wiping motor.
    2. ***Clean:*** The operator uses this button to activate the wiping motor.and the washing LED.
- OUTPUTS - 6
    1. ***Stepper motor:*** You will interface five output pins to the motor using the ULN2003 driver.
    2. ***Water pump:*** You will interface an LED to a pin. This LED will flash to signify soapy water is being squirted onto the windshield. Pick any flash rate that is observable to the eye.

We are not specifying what the system should do if both switches are pressed. Feel free to implement whatever you wish when both are pressed. There are many options for simulation of inputs and outputs, but you interface the switches to Ports A, C, D, or E. You can interface the stepper to Ports A, B, D or E. You should wait about 50 ms per step so the motor has time to react to the software commands

Procedure
The basic approach to this lab will be to first develop and debug your system using the simulator and then interface with an actual stepper motor and switches on a physical TM4C123. As you have experienced, the simulator requires different amount of actual time as compared to

simulated time. On the other hand, the correct simulation time is maintained in the SysTick timer, which is decremented every cycle. The simulator speed depends on the amount of information it needs to update into the windows and the speed of your personal computer.

All software in this lab must be developed in C. The Lab 5 starter file has the appropriate files needed for Lab 5. The call to **TExaS_Init** will activate 80 MHz PLL. The function passed to **TExaS_Init** will specify which bits will be sent to the logic analyzer.

Part a - Pin/Port Selection

You can use any GPIO port pins for the inputs and outputs. However, during simulation, you must use a setting available in the Stepper Motor window. Since we will not reuse this lab in subsequent labs, feel free to use any pins. *You can't use the same port for input as output.*

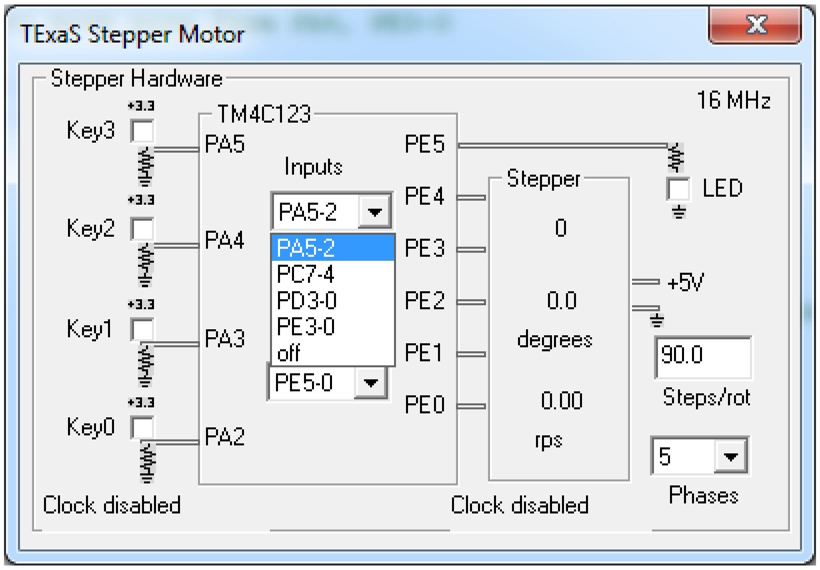


Figure 5.2. Simulation allows switches on PA5-2, PC7-4, or PD3-0

The LaunchPad connects PB6 to PD0, and PB7 to PD1. If you wish to use both PB6 and PD0 you will need to remove the R9 resistor. Similarly, to use both PB7 and PD1 remove the R10 resistor. You may discard R9 and R10, you will not need them. The R9 R10 jumpers are only needed for some very old MSP430 booster packs running on the TM4C123, so there is little chance you will ever need R9 and R10.

Part b - FSM Design

Design a finite state machine that implements the stepper motor control system. Include a picture of your finite state machine (state transition graph, STG) in the deliverables showing the various states, inputs, outputs, wait times and transitions.

Part c - Debug C Code In Simulation

Write and debug the C code that implements the stepper motor control system. In simulation mode, capture logic analyzer screen shots showing the operation of your controller when each switch is pressed individually.

Part d - Construct and Test Circuit

After you have debugged your system in simulation mode, you will implement it on the real board. Use the same ports you used during simulation. The first step is to interface two push button switches for the user controls. You should implement positive logic switches. There will be one LED output that will flash to signify the water squirting. Build the switch circuits and test the voltages using a voltmeter. You can also use the debugger to observe the input pin to verify the proper operation of the interface. The next step is to interface the five GPIO outputs to the stepper motor.

Do not place or remove wires on the protoboard while the power is on.

Part e - Debug on Real Hardware

Debug your combined hardware/software system on the actual TM4C123 board. https://youtu.be/5BIeSRukyVs

Demonstration
An interesting software testing questions are these,

“Can you prove the output never jumps between 1-4, 1-8, 2-8, 2-16, or 4-16?”
“Can you prove the data structure in ROM exactly implements the FSM graph?”

In real products that we market to consumers, we put the executable instructions and the finite state machine linked data structure into the nonvolatile memory such as Flash ROM. A good implementation will allow minor changes to the finite machine (adding states, modifying times, removing states, moving transition arrows, changing the initial state) simply by changing the linked data structure, without changing the executable instructions. Making changes to executable code requires you to debug/verify the system again. If there is a 1-1 mapping from FSM to linked data structure, then if we just change the state graph and follow the 1-1 mapping, we can be confident our new system operate the new FSM exactly as drawn in the STG. Obviously, if we add another input sensor or output, it will be necessary to update the executable part of the software, re-assemble or re-compile and retest the system.

Deliverables

1. Logic analyzer screenshot while in simulation mode, when the button is pressed and released.
2. Drawing of the finite state machine, by hand or done on computer.
3. All source files that you have changed or added should be uploaded to Canvas.

Remarks:

This lab was written in a manner intended to give you a great deal of flexibility in how you draw the FSM graph, while at the same time require very specific boundaries on how the FSM controller must be written. To have a better design and less complexity, try to meet these 10 requirements.

1. **<u>Input Dependence:</u>**
   This means each state has 8 arrows such that the next state depends on the current state and the input. This means you can not solve the problem by simply cycling through all the states regardless of the input. You should not implement a Mealy machine.

2. **<u>1-1 Mapping</u>**:
   There must be a 1-1 mapping between state graph and data structure. For a Moore machine, this means each state in the graph has an output, a time to wait, and 4 next state arrows (one for each input). The data structure has exactly these components: an output, a time to wait, and 8 next state pointers (one for each input). There is no more or no less information in the data structure then the information in the state graph. In other words, what you have down on your state graph is exactly mapped to your data structures in the software.

3. **<u>No Conditional Branches:</u>**
   There can be no conditional branches in your system, other than in **SysTick_Wait** and/or in **SysTick_Wait10ms**. This means there can be no do-while, while-loop, switch, if-then, or for-loops (other than the SysTick wait functions). See how the main program in book Example 5.2.1 executes without conditional branching. You will have an unconditional while-loop in the main program that runs the FSM controller.

4. **<u>Clear State Graph:</u>**
   The state graph defines exactly what the system does in a clear and unambiguous fashion.

5. **<u>Consistent State Format:</u>**
   Each state has the same format as every other state. This means every state has exactly 5-bits of output, one time to wait, and 4 next pointers.

6. **<u>Naming Convention:</u>**
   Please use good names (easy to understand and easy to change). Examples of bad state names are **S0** and **S1**.

7. **<u>No Bad Stepper outputs:</u>**
   The only valid outputs are 1, 2, 4, 8, and 16 (do not output any other value). The output sequence should not jump from 1 to 4, 4 to 1, 1 to 8, 8 to 1, 2 to 8, 8 to 2, 2 to 16, 16 to 2, 4 to 16, or 16 to 4. There must be 20 to 100ms between outputs.

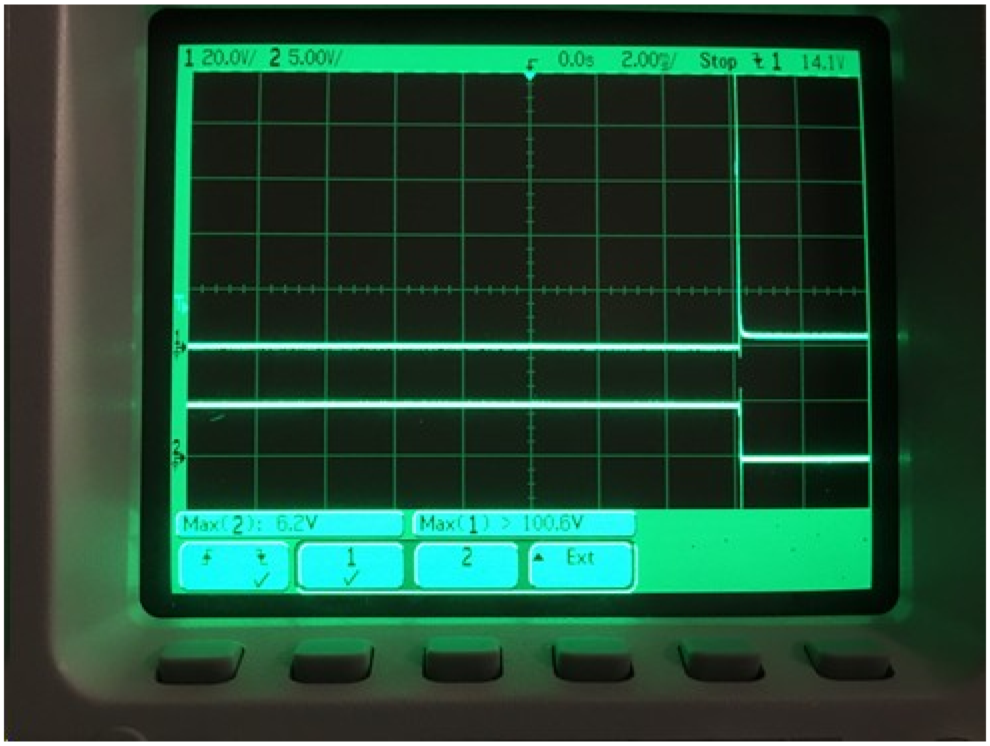
8. **<u>State Number Requirement:</u>**
   There should be approximately 20 to 50 states with a Moore finite state machine. Usually students with less than 20 states did not move the motor very far or did not properly

handle flashing the LED and stepping the motor at the same time. Counters and variables violate the "no conditional branch" requirement. If your machine has more than 50 states you have made it more complicated than we had in mind and you should consider simplifying your design.

9. **How to get the motor to return to home**:
   Start with the output pattern needed to wipe once stopping at home. Consider how you would get it to wipe continuously when the wiper switch is active. Put the two thoughts together so it returns to home when you release the wiper switch.

10. **Making the water pump LED flash:**
    Think about the sequence of outputs required for the LED to flash. Embed that sequence as output values for sequential states.

Tips and Tricks Section:
You can get over 100V on your protoboard if you forget to connect pin 9 to +5V. Here is a scope trace showing over 100 V on the collector pin of the motor circuit running without pin 9 connected!



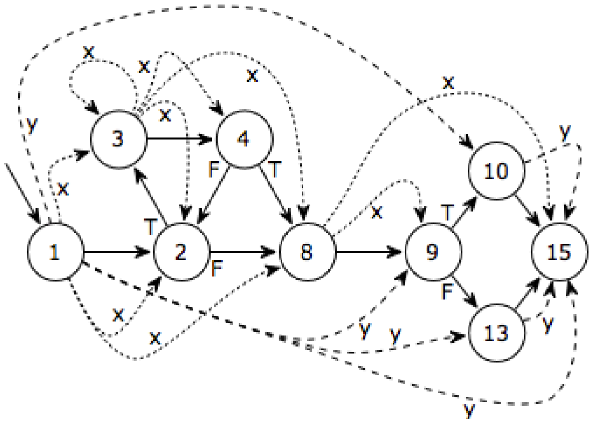
For example, the voltage current relationship of an inductor is

$$V = L\ dI/dt$$

If L=1mH, I=200mA, and the switch time of the ULN2003 is 1us, you can get 200V
200V = 1 mH*0.2A/1us

- There is no single, "best" way to implement your system. However, your scheme must use a linked data structure stored in ROM. There should be a 1-1 mapping from the FSM states and the linked elements. An example solution will have about 40 states in the finite state machine, and provides for input dependence.
- Try not to focus on the automotive engineering issues. I.e., the machine does not have to maximize speed. On the other hand the time delays need to be long enough so the motor moves.

Aside: If your FSM starts to look like the image below, you may want to clean it up.



Drawing your FSM

- Recall the FSM example in the class. Because we have two inputs, there will be 4 next state arrows. One way to draw the FSM graph to make it easier to read is to use X to signify don't care. For example, compare the Figure 5.2.4 in the book to the FSM graph in Figure 5.3 below. Drawing two arrows labeled **01** and **11** is the same as drawing one arrow with the label **X1**. When we implement the data structure, we will expand the shorthand and explicitly list all possible next states.

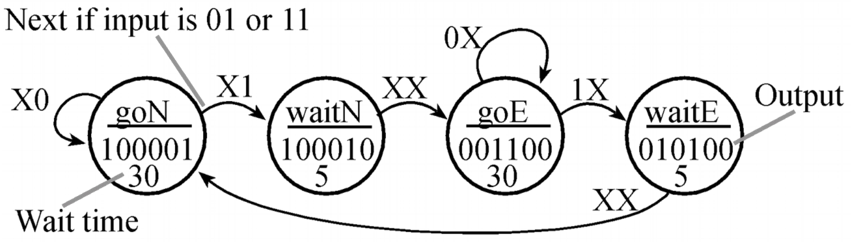
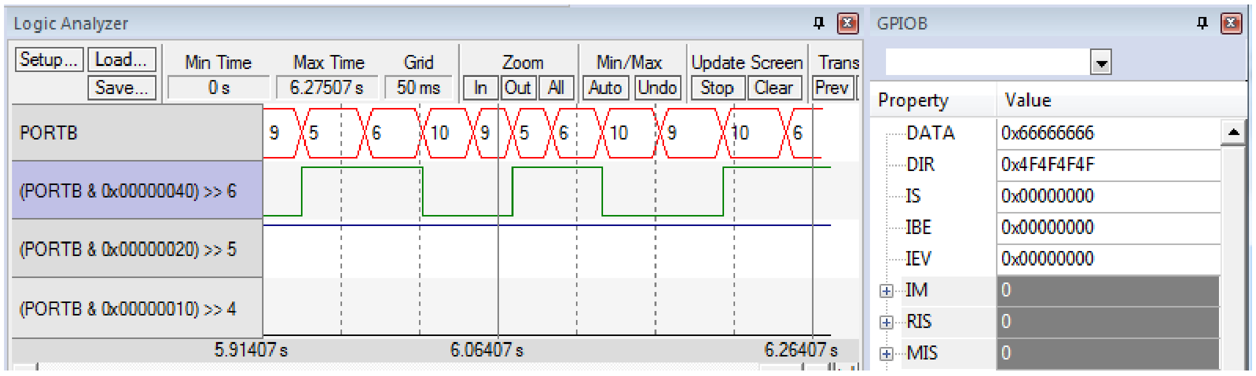


Figure 5.3. FSM from the book Figure 5.2.4 redrawn with a shorthand format.

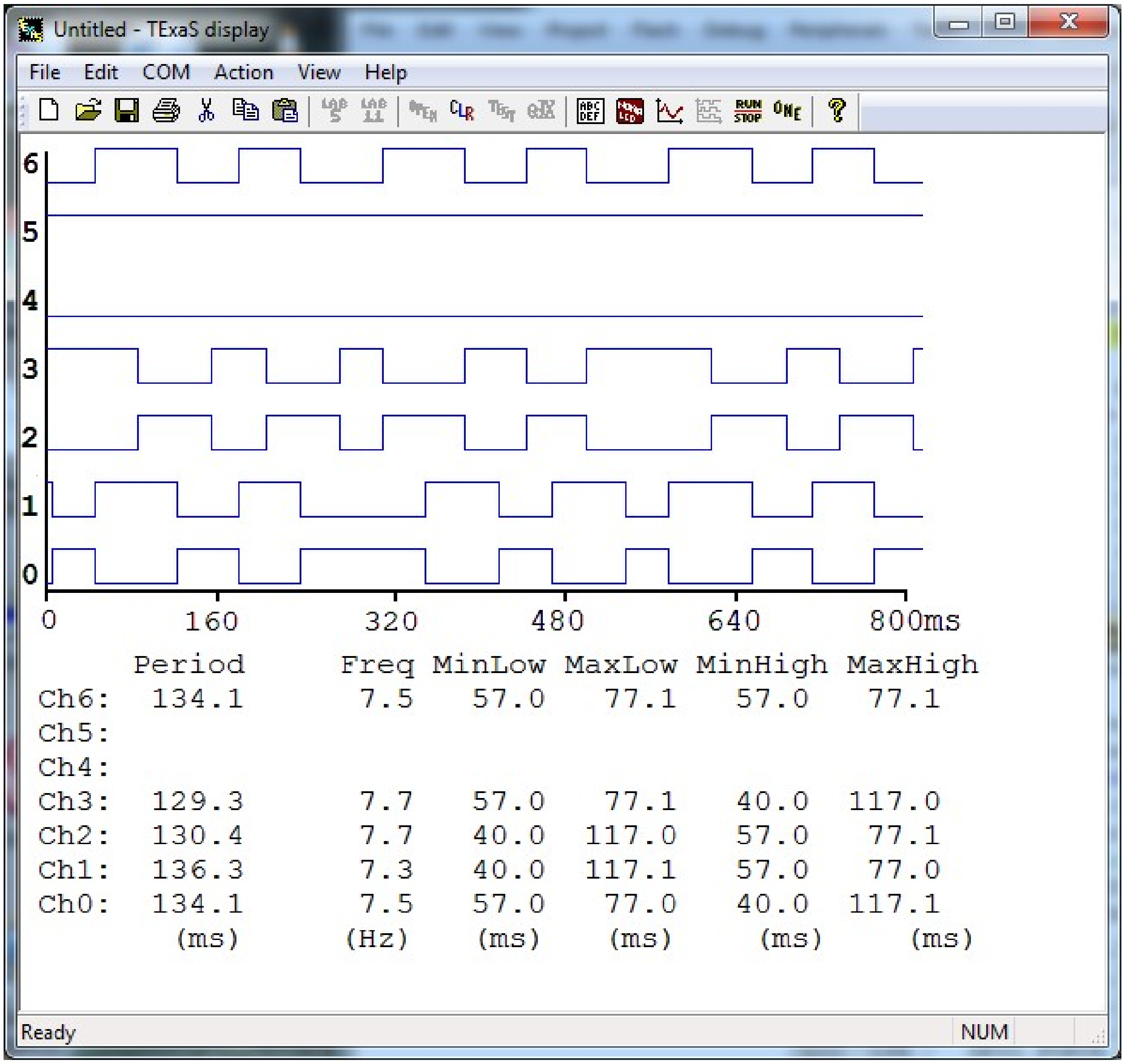
Example Logic Analyzer Window.

Your I/O window may look slightly different from these examples (e.g., you are free to assign signals to different port bits.)



Property	Value
DATA	0x66666666
DIR	0x4F4F4F4F
IS	0x00000000
IBE	0x00000000
IEV	0x00000000
IM	0
RIS	0
MIS	0

Simulation showing washing and wiping



```
         Period      Freq  MinLow  MaxLow  MinHigh  MaxHigh
Ch6:     134.1        7.5    57.0    77.1     57.0     77.1
Ch5:
Ch4:
Ch3:     129.3        7.7    57.0    77.1     40.0    117.0
Ch2:     130.4        7.7    40.0   117.0     57.0     77.1
Ch1:     136.3        7.3    40.0   117.1     57.0     77.0
Ch0:     134.1        7.5    57.0    77.0     40.0    117.1
          (ms)       (Hz)    (ms)    (ms)     (ms)     (ms)
```

TExaSdisplay logic analyzer showing washing and wiping
(this FSM slows down as it approaches the ends of travel where it needs to reverse direction)

FAQ: Frequently Asked Questions

1. **Does clockwise and counterclockwise matter?**
   No, just go one way and then go the other way.

2. **We are getting an error when we compile, saying that the file TM4C123GH6PM.h cannot be opened because it cannot be found. Where can we find this file?**
   If you still have your EE319kware, you could copy paste it into your folder that lists includes your lab 5.

3. **Should we use the ULN2003 IC used in Lab 3 in this lab too? Or is that not necessary?**
   You must use the driver. Each motor coil needs about 100 mA and the microcontroller can only source or sink up to 8 mA.

4. **The lab manual mentions "SysTick_Wait" and "SysTick_Wait10ms" subroutines. Are we supposed to program these, or are they included somewhere?**
   These files are found in the SysTick_4c123 folder of the 319kware. You can copy and paste the SysTick.c and .h files into your working directory, then add them to your project by right clicking the source folder in the Keil project explorer on the left side and selecting "Add existing files...". Make sure to include the #include "SysTick.h" in your main.c to get the function prototypes for SysTick_Init and SysTick_Wait. SysTick.c will be in C, so no worries of assembly there. You can call assembly functions from C, but it would be easier to just stick with the C file in this case.

5. **Is #define STEPPER_OUT (*((volatile uint32_t *)0x4000507C)) the correct code to set bits 4,3,2,1,0 of Port B so that I can change the just the stepper? I got this number by adding 0x40, 0x20, 0x10, 0x08 and 0x04 to 0x4000.5000 which is the start memory map in peripherals in one of the data sheets.**
   Yes, that address will allow you to read/write to PB4-PB0 without affecting the other pins on Port B

6. **Are there limitations to SysTick_Wait that we should know about? It seems that when we call this subroutine to wait for 2 seconds, it gets stuck in an infinite loop**
   Make sure to call SysTick_Init. Remember to write the functions in the SysTick.c file

7. **What is a possible source of error to the warning "_____ macro redefined"? This occurs whenever I try to do a**
   ```
   #define GPIO_PORTE_DIR_R    (*((volatile unsigned long *)0x40024400))
   ```
   This macro is defined in the "tm4c123gh6pm.h" file. There is no need to redefine this macro since you should be including this file in your lab 5 source codee.

8. **How do we do a NOP properly in C? We keep getting an error saying that Port A doesn't have a clock set.**
   Make sure none of your code uses the old clock registers like SYSCTL_RCGC2_R. Rather use SYSCTL_RCGCGPIO_R to set clock.

9. **I don’t know why there should be 4 next pointers? I don't see how there could be more than 4 states.**

   Since you have two different inputs buttons, you have 4 different combinations of inputs that you need to account for. There are many states where you will progress to another state regardless of the status of the switches.

10. **What is the purpose of pin 9 of the ULN2003.**
    As the controller switches states (e.g., 1 to 2), one coil is deactivated and another coil is activated. This causes a discontinuity as the current goes from 100 mA to 0. Because the coil is wound in a spiral, the coil has inductance L. The voltage across an inductor is V = L dI/dt. The inductance L and dI/dt generate large voltages (up to 100 V). This diode on pin 9 safely discharges this spike into the +5V power line. Forgetting this diode connection will result in damage to your microcontroller.

11. **How would you show the LaunchPad LED lights on PCBartist? Do you have to show it at all?**
    The circuit is already embedded in the board, so I would only worry about showing pinouts and external hardware interfaces for the ULN2003 driver, motor, led, switches, etc.

12. **I know we can't use conditional branches, but switches were not listed as something we weren't allowed to use in the lab manual. Will points be deducted if we implement a switch?**
    The reason we prohibit branches is because the logic of which next state to go to for each state should be specified in your FSM data, not in your engine. The only thing your engine should do is find the next state in the data, using the current state as an index or pointer (whichever you used in your design). This way, you should have no need to use switches or conditional branches.

13. **Do we need pull down resistors for the LEDs on the board?**
    You only need to use the PDR and PUR registers when configuring the onboard switches. For the LEDs, you don't need to worry about them. For off board positive logic switches you can use the explicit 10k resistor or the PDR register.

14. **I am not able to have lasting color changes on their logic analyzers?**
    Select a color to change the pin to, click ok to close the palette, then click "close" on the setup analyzer window. Repeat for all pins. It's tedious that you have to do it for each pin individually, but it worked for me.

15. **Why are we disabling and then enabling interrupts?**
    Typically global interrupts rather than individual functions are disabled during "critical sections". Consider what would happen if you were initializing a timer or some other function when another interrupt fired? Or if you were reading or writing to a piece of data when an interrupt fired that modified the data at the same address? When you start using

interrupts more in your labs in the upcoming weeks it'll become more of something to think about. The TExaS logic analyzer uses a periodic timer interrupt to send data from your system to the PC 10,000 times/sec or every 100 us

Extra thoughts that do not need to be implemented:

- Let Θ be the angle of the motor. As you know this angle increments or decrements by 4degrees after each output to the motor. The rotational speed of the motor would be $d\Theta/dt$. The rotational acceleration is $d^2\Theta/dt^2$. A very interesting parameter of the motor is jerk, $d^3\Theta/dt^3$. A very large jerk occurs when the software attempts to reverse direction. From a motor controller design perspective we can optimate motor performance (speed and accuracy) by minimizing jerk. Basically, one puts short time delays while it is moving in the middle, and longer time delays as it slows down to reverse direction on the ends of the travel.
- The sequence 1,2,4,8,16 is called full stepping. Notice that each change flips one pair of bits. There is another sequence called half-stepping. For half-stepping, there will be 10 valid output patterns and each output causes a ½ step (5 degrees for our motor). For example going from 1 to 2 cause bit 0 to flip off and bit 1 to flip on. The new pattern, placed between the existing 5 patterns, is created by first turning one bit on then turning the other bit off . For example going from 1 to 2 is replaced by 1 to 3 to 2. The complete 8 pattern output is 1,3,2,6,4,12,8,24,16,17. For this sequence the motor angle changes by 5 degrees after each output.