ECE 315 Lab 4 🧀

For Ahmed and Shyama 🎁💯🙏

🚙 Lora Ma
🌎 Benjamin Kong

ECE 315 Lab Section H41

March 31, 2022

Contents

1 Abstract

The purpose 😂 of this lab 😐 was to

- gain experience 💰 using a microcontroller, running the FreeRTOS real-time kernel, to control the operation of a small stepper motor
- to enhance 💰 the user interface of a stepper motor control application on the Zybo Z7.
- gain experience 💰 with measuring experimentally the speed and acceleration limits of a small stepper motor.

We will be using the 🚒 Zybo Z7 development board by 🚙 Digilent. The board is built around the 🚗 Xilinx 🚕 Zynq-7010 System-on-Chip silicon chip and contains two 667 MHz ARM Cortex A9 32-bit CPUs. For this lab, we will be using CPU0 to run the FreeRTOS real-time kernel. We will also be using one 28BYJ-48 stepper motor with unipolar drive windings, one 5-V DV power supply, one ULN2003-based driver module, one breadboard, one LTV-847 opto-isolator transistor array, four 220-ohm resistors, four 😤 10 kΩ 😤 resistors, and various wires. 🍎

In this lab, we will be completing two exercises. In exercise 1, we will be completing the FreeRTOS task `_Task_Motor()` to rotate the stepper motor. This task is responsible for calling the necessary stepper functions to move the motor.

In exercise 2, we will be providing an emergency stop command for the stepper motor. We are implementing this functionality using one of the Zybo Z7 board's pushbuttons (BTN0) as an emergency stop button.

2 Design

2.1 Hardware Design

We first got the hardware set up. We first set up the wire connections from the 🎱 Zybo to our breadboard according to the schematics in the lab handout. We connected the appropriate wires and resistors to the slots of the chip. We then connected the motor/motor control to the breadboard

as well. It is important to note that the motor is powered by an external 5V connection, not by the Zybo. We verified that 5V was running through the system using a voltmeter.

2.2 Exercise 1 Design

We first completed the `_Task_Motor` function in `lab4_main.c`. Inside of the `while` loop, we wait until we read some motor parameters from FIFO1. We read this into a struct containing various motor parameters such as current position, rotational speed, acceleration, and deceleration. After reading these parameters from FIFO1, we call the appropriate stepper functions in the driver files:

- `Stepper_setCurrentPositionInSteps` - to set the current position of the stepper motor,
- `Stepper_setSpeedInStepsPerSecond` - to set the speed in steps per second of the stepper motor,
- `Stepper_setAccelerationInStepsPerSecondPerSecond` - to set the acceleration in steps per second of the stepper motor, and
- `Stepper_setDecelerationInStepsPerSecondPerSecond` - to set the deceleration in steps per second of the stepper motor.

After setting the parameters, we begin actually moving the motor. Recall that the user input results in a list of destination/delay pairs. We begin iterating through each of these starting at the first one. We first set the `target` variable to be the position of the first destination/delay pair. We then call the stepper function provided in the driver file to move the motor to the desired destination. We then disable the motor, again using the function provided in the driver file. Lastly, we call `vTaskDelay` using a delay provided in the destination/delay pair. After this, we repeat with the next destination/delay pair until there are no more destination/delay pairs.

2.3 Exercise 2 Design

To implement the emergency stop, we first read the button value using `XGpio_DiscreteRead`. If the button is indeed pressed, then we initiate the emergency stop sequence. How the emergency stop is implemented is discussed below.

To implement the 🚒 emergency stop, we first set the position of the motor to the current target. We then set `sequenceIndex` to 0 and disable the stepper motor (using the methods provided in the driver file). We then delete the `xMotortask` and `xUarttask` to terminate those tasks so that the entire system is stopped and so that the emergency stop cannot be overridden by another task. The motor is now off and no other tasks exist that can restart the movement of the motor.

We next want to flash a 🚒 red LED at 2Hz. In order to do this, we simply create an infinite loop. Similarly to a previous lab, we turn the LED on for half the delay time, turn it off for half the delay time, and repeat this forever. At this point, the emergency sequence is complete and only resetting the system can stop the emergency state.

3 Testing

(yeet 🚗 vroom 🚕 vroom 🚙 car go 💨 brrrrr)

Table 1: Tests for exercise 1 and 2.

Description	**Expected**	**Pass**
Enter nothing	Motor should not move	Yes
Enter default motor position, speed, acceleration, and deceleration. Then enter destination/delay pairs of 2048/1000 and -2048/1000	Motor turns clockwise, then after an observable delay, begins to turn counterclockwise. At the end, the motor stops moving	Yes
Repeat the above steps, but press BTN0 while the motor is turning clockwise	Motor should stop and red LED should flash at 2Hz	Yes
Repeat the above steps, but press BTN0 while the motor is turning counterclockwise	Motor should stop and red LED should flash at 2Hz	Yes
Enter default motor position, speed, acceleration, and deceleration. Then enter destination/delay pairs of 2048/1000, -2048/1000, and 2048/1000	Motor turns clockwise, counterclockwise and then clockwise again before stopping	Yes

4 Conclusion

The purpose 💵 of this lab was to

- gain experience 🚀 using a microcontroller, running the FreeRTOS real-time kernel, to control the operation of a small stepper motor
- to enhance 🚀 the user interface of a stepper motor control application on the Zybo Z7.
- gain experience 🚀 with measuring experimentally the speed and acceleration limits of a small stepper motor.

We believe we have fully completed the objectives of this lab. 🎯

In exercise 1 and 2, we gained experience using a microcontroller, running the FreeRTOS real-time kernel, and controlling the operation of a small stepper motor. We did this through setting up the microcontroller and stepper motor. We also wrote the `_Task_motor()` task which was responsible for calling the necessary stepper functions to move the motor. We also gained experience enhancing the user interface of a stepper motor control application on the Zybo Z7. We also gained experience in exercise 2 with measuring experimentally the speed and acceleration limits of a small stepper motor. We also set BTN0 on the Zybo Z7 board's pushbuttons to initiate an emergency stop. 🏆

5 Appendix

5.1 Exercise 1 Source Code

The source code is in the file `lab4_main.c` that was submitted along with this report. 🚰

5.2 Exercise 2 Source Code

The source code is in the file `lab4_main.c` that was submitted along with this report. 🧀