# ECE 315 Computer Interfacing

# Lab #4: Controlling a Stepper Motor Using the Zybo Z7

**Winter 2022**

## Lab Dates, Demo Due Dates, and Report Dates

This lab exercise will be held on March 15 (LAB H21), March 16 (LAB H31), Match 17 (LAB H41), and March 18 (LAB H51). The deadlines for the lab demos and the lab reports will be March 29 (H21), March 30 (H31), March 31 (H41), and April 1 (H51). Your lab report must be uploaded by 11:00 pm on the same day that your lab demonstration is due. Since the end of Winter term is near, the above-mentioned deadlines must be strictly respected.

## Objectives

- To gain experience with 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.
- To gain experience with measuring experimentally the speed and acceleration limits of a small stepper motor.

## Hardware Platform, the Software Environment, and the Stepper Motor

- Once again, the controller platform is the Digilent Zybo Z7 development board. As with all the previous labs, CPU0 in the Zynq-7000 SoC will run the FreeRTOS real-time kernel.
- The fixed Zynq-7000 SoC hardware configuration and the initial stepper motor control software for Exercise 1 must be downloaded from the Lab #4 section of the lab eClass site. The given software contains the initial application code in C that you will be modifying to implement your software designs in the first two exercises.
- One 28BYJ-48 stepper motor with unipolar drive windings, one 5-V DC power supply, one ULN2003-based driver module, one breadboard, one LTV-847 opto-isolator transistor array, four 220-ohm resistors, four 10-Kohm resistors, and wires. **Note: The stepper motor must be powered by the 5-V DC power supply and *not* by the Zybo Z7 board.**

## Documentation

- Digilent Zybo Z7 Reference Manual: https://reference.digilentinc.com/reference/programmable-logic/zybo-z7/reference-manual
- The FreeRTOS Reference Manual: https://www.freertos.org/fr-content-src/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.
- Chapters 5 and 10 of the ECE 315 lecture slides.
- Datasheet for the 28BYJ-48 5-VDC stepper motor: https://download.mikroe.com/documents/datasheets/step-motor-5v-28byj48-datasheet.pdf
- Datasheet for the ST Microelectronics ULN2003 Seven Darlington drivers IC: https://www.digikey.ca/htmldatasheets/production/8883/0/0/1/uln200x-a-d1-.html
- Datasheet for the Lite-On LTV-847 quad opto-isolator IC: https://media.digikey.com/pdf/Data%20Sheets/Lite-On%20PDFs/LTV-817_827_847(M,S,S_TA1).pdf

## Laboratory Exercise Instructions

### Pre-lab work (10%)

The hardware configuration shown in Figure 1 needs to be **assembled** and **verified** by you in the lab session. This is possible to do in well under one hour. There are five connected subsystems: (1) The Zybo Z7 board, which needs to have the PmodSSD card removed so that the four motor control signals and digital ground can be connected to the upper sockets of the JC Pmod header; (2) an opto-isolation circuit for the motor control signals, which needs to be assembled on the breadboard; (3) a ULN2003-based driver board, which needs to be connected between the opto-isolation circuit and the four motor control signal wires; (4) the 28BYJ-48 stepper motor; and (5) an external 5-V DC power supply, which supplies the current to the stepper motor windings. The instructions for each of the five subsystems are given below:

#### (1) Remove the 7-segment LED display Pmod module from the Zybo Z7 board

**Step 1:** Make sure that the power switch (near the micro-USB-JTAG cable connection) is in the OFF position, but keep the cable connected so that the board remains grounded through to the host PC via the cable shield.

**Step 2:** The two-plug PmodSSD module (see Fig. 2) that provided the two-digit 7-segment LED display must be *carefully* removed from the Zybo Z7 board. Be sure to ground your finger by firmly touching a grounded metal object (for example, a metal screw attached to the case of a desktop PC, a heat register connected to house ductwork, or the metal case of an instrument or a kitchen appliance), before touching any electronic components or boards. In addition, be sure to avoid touching the pins on any headers or cables. Removing the PModSSD is a tricky operation because the module's pins tend to jam if the module is not pulled straight out from the JC Pmod header on the board. If you have needle-nose pliers (see Fig. 3), then you can carefully grab and pull back, in several small stages, the two corners of the PmodSSD module (shown as the red circles in Fig. 2). Grasping the two short edges of the module with your fingers is another option for your Zybo Z7 board if you can reach and grab the two short edges of the module.

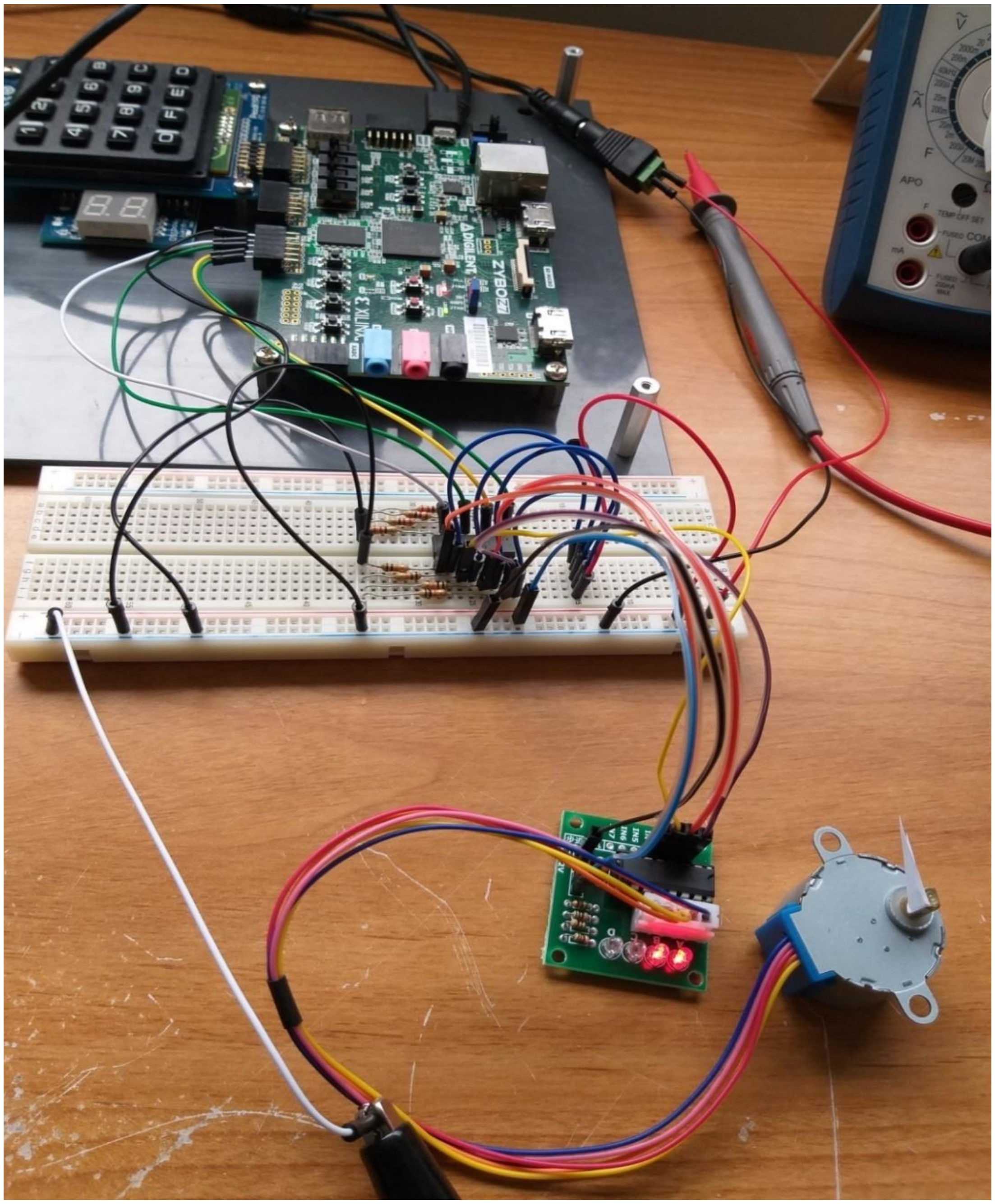

**Fig. 1: The Lab 4 Hardware Configuration**

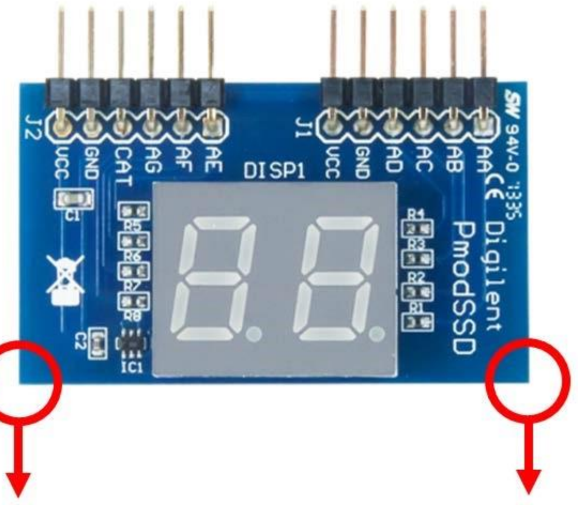

**Fig. 2: PmodSSD Pull Corners**

**Fig. 3: Needle-nose Pliers**

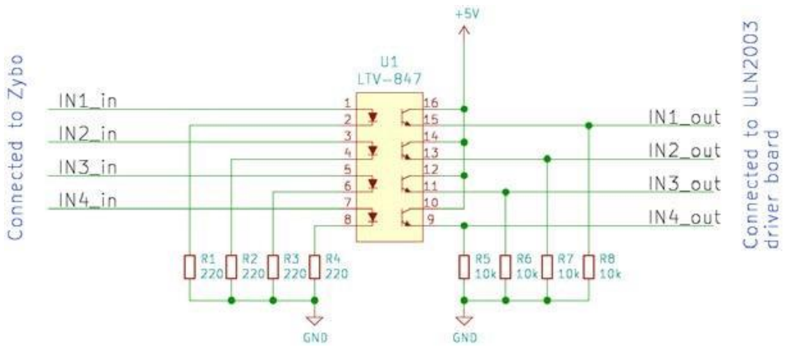

**Fig. 4: Optoisolation Circuit for the Bridge Signals**

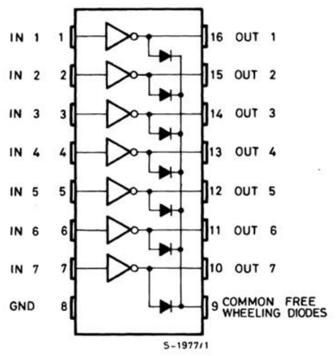

**Fig. 5: ULN2003 Driver Array**

**(2) Assemble the opto-isolation circuit on the breadboard**

**Step 3:** Carefully read the following. When the current flowing through the stepper motor's stator windings is switched on and off, voltage spikes are induced across the windings following Faraday's Law of induction. These voltage spikes can be many volts in magnitude and they can thus damage the sensitive digital components on the Zybo Z7 board, which operate on a 3.3-V DC supply. **Thus it is essential that the electrical motor drive control signals from the Zybo Z7 be isolated electrically from the drive transistors and the stepper motor windings.** This will be done using an opto-isolation circuit (Fig. 4), which you will build on the breadboard (see Fig.1).

**Step 4:** Plug the LTV-847 quad opto-isolator IC into the breadboard, as shown in Fig. 1. The small depression in the IC, indicating pin #1, should be at the top-left corner. Connect pins #2, #4, #6 and #8 through separate 220-ohm transistors (with red-red-black-gold bands) to your ground rail (the one labelled '-') on the breadboard. Connect pins #9, #11, #13 and #15 through separate 10K-ohm resistors (with brown-black-orange-gold bands) to the same breadboard ground rail.

**Step 5:** Connect the four motor control signals and the digital ground rail between the JC Pmod header on the Zybo Z7 board to the breadboard as follows. The JC Pmod header has 12 sockets in two rows, and you will be connecting wires to the *rightmost five sockets on the top row*. The six sockets on the top row (going from left to right when looking into the sockets) will have: (1) no connection, (2) connection to breadboard ground, (3) connection of IN4_in to pin #7 on the LTV-847, (4) connection of IN3_in to pin #5 on the LTV-847, (5) connection of IN2_in to pin #3 on the LTV-847, and (6) connection of IN1_in to pin #1 on the LTV-847.

**Step 6:** Connect pins #10, #12, #14 and #16 to your power rail (labelled "+") on the breadboard.

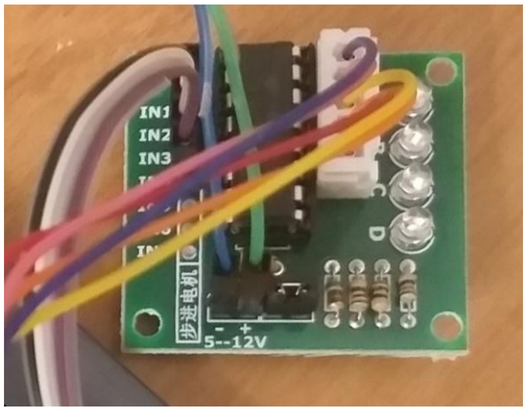

**Fig. 6: ULN2003-based Driver Module**

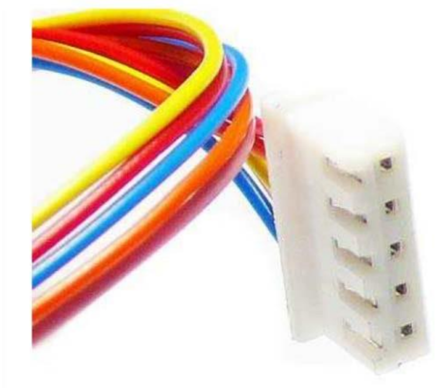
**Fig. 7: Stepper Motor Signal Plug**

**(3) Connect the driver module**

**Step 7:** The driver board (Fig. 6) is based on the ULN2003 IC (see Fig. 5), which contains an array of seven pull-down Darlington transistor switches. We will only be using the first four switches to selectively connect the four outer stepper motor winding terminals to ground. Make the following connections: (1) connect IN1_out from pin #15 on the LTV-847 to socket IN1 on the driver module; (2) connect IN2_out from pin #13 on the LTV-857 to socket IN2 on the driver module; (3) connect IN3_out from pin #11 on the LTV-857 to socket IN3 on the driver module; and (4) connect IN4_out from pin #9 on the LTV-857 to socket IN4 on the driver module.

**(4) Connect the stepper motor**

**Step 8:** Plug the 5-conductor stepper motor signal cable (Fig. 7) into the driver module socket.

**Step 9:** Attach a small tab of tape, about 1.5 cm long, to the axle of the motor.

**Step 10:** Carefully read the following. We will be using the inexpensive and popular 28BYJ-48 stepper motor (Fig. 8), which is designed to operate using an external 5-V DC power supply. This stepper motor uses unipolar windings, as shown in Fig. 9. Terminal **#5 Red** is connected directly to 5-V DC. Note how this connection goes to taps at the centre of both motor windings. At any one time, at most one of terminal **#2 Pink** or **#4 Orange**, but not both, should be driven to ground

(0-V DC). Similarly, at any one time, at most one of **#1 Blue** or **#3 Yellow**, but not both, should be connected to ground. The three standard drive waveforms (wave drive, full step, and half step) that cause the motor's shaft to turn in steps are shown below in Fig. 10. The 28BYJ-48 stepper motor has 2048 full steps per revolution, which implies 4096 half-steps per revolution.

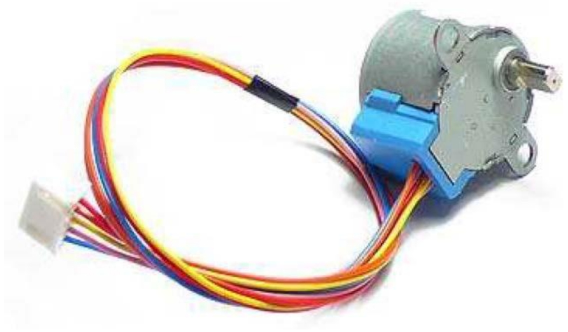

**Fig. 8: 28BYJ-48 5-VDC Stepper Motor**

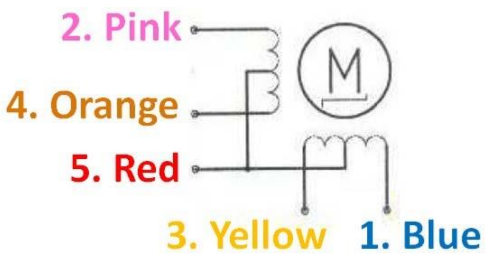

**Fig. 9: Schematic of Unipolar Windings**

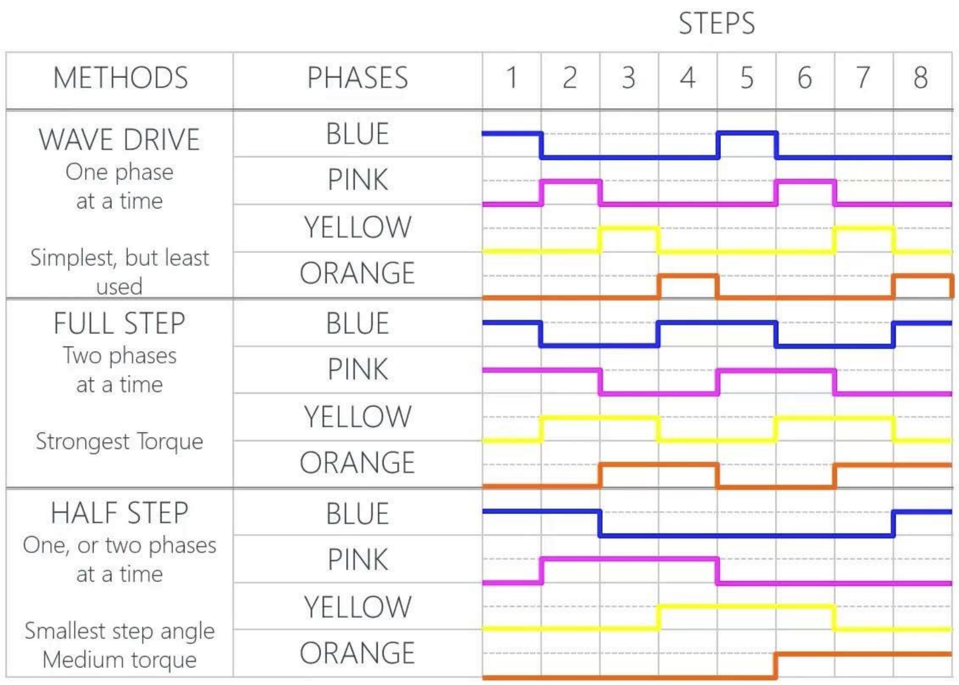

**Fig. 10: Drive Waveforms for the 28BYJ-48 Stepper Motor**

**(5) Connect the 5-V DC power supply**

**Step 11:** Make sure that the 5-V DC power supply is off.

**Step 12:** Connect the *ground terminal* from the power supply to the *ground rail* (labelled "-") on the breadboard. Also connect the *ground terminal* on the driver module to the ground rail on the breadboard.

**Step 13:** Connect the *+5-V DC terminal* from the power supply to the *power supply rail* (labelled "+") on the breadboard. Also connect the *positive supply terminal* on the driver module to the power supply rail on the breadboard.

**Step 14:** Carefully double-check all of the wiring connections.

**Step 15:** Turn on power switch to the Zybo Z7 board. Then turn on the 5-V DC power supply.

**Step 16:** Verify that the stepper motor can be operated using the provided stepper motor software, which is described next.

## The Provided Stepper Motor Control Software

**Necessary Board Support Package (BSP) setting change in SDK:** The tick clock frequency for the Zybo Z7 board needs to be increased from 100 Hz to 1000 Hz as follows:

- In SDK locate the **BSP folder** and expand it. Double-click on the **system.mss** file.
- Click on the **Modify this BSP's settings** option to open the **Board Support Package Settings** window.
- In the **Overview** drop-down menu select **freertos10_xilinx**.
- Expand the configuration options for **kernel behavior**, and then change the **tick_rate** from 100 to 1000.
- Click **OK** to update the BSP settings. The file **FreeRTOSConfig.h** is updated automatically.

**Necessary compiler option change in SDK:** The file stepper.c uses the mathematical function sqrt() and so the –m compiler flag must be present when the code is compiled.

- In SDK **right-click** on your ***project*** folder for Lab 4.
- Select **C/C++ Build -> Settings** to obtain a **Properties for <projectname>** window.
- Make sure that the **Tool Settings** tab is selected in the Settings window.
- Select **ARM v7 gcc linker -> Libraries** to obtain two library sub-windows.
- In the **Libraries (-l)** upper sub-window click on the **Add** icon (the leftmost small icon).
- Type **m** in the box and then click **OK** to specify the mathematics library for C.

- Click on **Apply** and then **OK**. The math library will now be included in C compilations.

## Initial User Interface

The provided initial user interface is produced by a FreeRTOS task that drives the UART. The interface begins by displaying "Stepper motor initialization complete! Operational parameters can be changed below:". Then the user interface will display a series of four parameters, current position, rotational speed, rotational acceleration, and rotational deceleration, with their default values, and then give the user a chance to change them. User can either choose to press <ENTER> to keep the default value or add a positive value from SDK terminal by typing xxx<ENTER>.

Once these four parameters are set, the user is asked to enter multiple destination positions and the dwell time at each position. The user interface will ask the user to enter the destination position. The initial value is 2048 steps (1 rotation clockwise) and user can press <ENTER> to keep this default value. The user can also type xxx<ENTER> to change the destination/target position. Please note that destination value can be positive/negative. A positive value indicates *clockwise* rotation of motor and negative value indicates *counter-clockwise* rotation of the motor. After the destination position is entered, user interface sends another message line asking for a delay at that position. The default delay is 0 milliseconds. User can either press <ENTER> to keep the default value or type xxx<ENTER> to change it. The user can specify a maximum of 10 <position, delay> commands. An example of three <destination, delay> pairs in shown below.

```
Stepper motor Initialization Complete! Operational parameters can be changed below:

Current position of the motor = 0 steps
Press <ENTER> to keep this value, or type a new starting position and then <ENTER>
User chooses to keep the default value of current position = 0 steps

Current maximum speed of the motor = 500.0 steps/sec
Press <ENTER> to keep this value, or type a new maximum speed number and then <ENTER>
User chooses to keep the default value of rotational speed = 500.0 steps/sec

Current maximum acceleration of the motor = 150.0 steps/sec/sec
Press <ENTER> to keep this value, or type a new maximum acceleration and then <ENTER>
User chooses to keep the default value of rotational acceleration = 150.0 steps/sec/sec

Current maximum deceleration of the motor = 150.0 steps/sec/sec
Press <ENTER> to keep this value, or type a new maximum deceleration and then <ENTER>
User chooses to keep the default value of rotational deceleration = 150.0 steps/sec/sec

Destination position of the motor = 2048 steps
Press <ENTER> to keep this value, or type a new destination position and then <ENTER>
User entered the new destination position = 4096 steps

Current dwell time of the motor between movements = 0 ms
Press <ENTER> to keep this value, or type a new dwell time and then <ENTER>
```

```
User entered new dwell time = 1000 ms
*** Pair: 1    , <destination, delay> = <4096, 1000>

Position slots available: 9
Press <ENTER> to stop entering more positions, or enter a new position and then <ENTER>
User entered the new destination position = -2048 steps

Current dwell time of the motor between movements = 0 ms
Press <ENTER> to keep this value, or type a new dwell time and then <ENTER>
User entered new dwell time = 1000 ms
*** Pair: 2    , <destination, delay> = <-2048, 1000>

Position slots available: 8
Press <ENTER> to stop entering more positions, or enter a new position and then <ENTER>
User entered the new destination position = 6144 steps

Current dwell time of the motor between movements = 0 ms
Press <ENTER> to keep this value, or type a new dwell time and then <ENTER>
User entered new dwell time = 5000 ms
*** Pair: 3    , <destination, delay> = <6144, 5000>

Position slots available: 7
Press <ENTER> to stop entering more positions, or enter a new position and then <ENTER>
User chooses to stop inputting sequences, or sequence full.


****************************** MENU ******************************
1. Press m<ENTER> to change the motor parameters again.
2. Press g<ENTER> to start the movement of the motor.
```

Once the user has entered the <destination, delay> pairs, press <ENTER> to stop entering any more values in the interface. The interface will now ask the user to press m<ENTER> to change the motor parameters again or g<ENTER> to start the motor movement. After the last iteration is done, the motor should stop.

## Stepper Motor Driver Functions

A library of stepper motor control functions is provided with the Lab 4 software package. Unzip the Lab 4 files to a convenient directory and study the contents of the files **`stepper.h`** and **`stepper.c`**. As one would expect, the header file contains **`#include`** directives that specify other required include files, as well as **`#define`** definitions of symbolic constants, and prototype headers for the functions that are defined in the accompanying C source file. The stepper motor driver uses the full-step drive waveforms. The full-step waveforms are defined in the header file along with the wave drive and half-step drive waveforms. Variables are provided that give the status of the stepper motor including the **`currentPosition_InSteps`**, the **`targetPosition_InSteps`**, and the **`decelerationDistance_InSteps`**. Calculations are made in real-time using variables of type **`float`** to update the ideal rotor speed, rotor acceleration (or deceleration), and step period. The step period resolution must be reduced to 1 millisecond because the tick clock runs at 1 KHz. In **`stepper.c`** locate the following useful functions:

- `void Stepper_Initialize(); // Called to initialize the stepper motor driver`
- `void Stepper_setCurrentPositionInSteps(long); // Redefines the current position in steps`
- `void Stepper_setSpeedInStepsPerSecond(float); // Set max allowed speed`
- `void Stepper_setAccelerationInStepsPerSecondPerSecond(float); // Set max acceleration`
- `void Stepper_setDecelerationInStepsPerSecondPerSecond(float); // Set max deceleration`
- `void Stepper_moveToPositionInSteps(long); // Move the motor to a new absolute position`
- `_Bool Stepper_motionComplete(); // Is the motor at the target position?`
- `void Stepper_disableMotor(); // Disable the motor`

## Exercise 1: Complete the FreeRTOS task, _Task_Motor(), to rotate the stepper motor (22.5%)

Students are required to complete the _Task_Motor() task that is responsible for calling the necessary stepper functions to move the motor. Please enter your code in the commented section inside this task to produce the correct motor rotations.

## Exercise 2: Provide an emergency stop command for the stepper motor (22.5%)

In some applications of stepper motors, it is essential to have the means to rapidly stop the movement controlled by the motor. In this exercise you are to implement an emergency stop function using one of the Zybo Z7 board's pushbuttons, **BTN0**. In Lab 1 this push-button was read using an XGpio instance and the application interface for the AXI GPIO IP blocks. Please use the same method once again in this lab. The required code is provided for you to use.

**Step 1:** The FreeRTOS task `_Task_Emerg_Stop()` has been created for you. If the BTN0 is pressed for three consecutive polling operations in a row, then call a stepper motor driver function that causes the motor to immediately begin decelerating to a stop. There is a suitable function already present in the stepper.c file.

**Step 2:** When the emergency stop command has been issued, then the **red LED** on the Zybo Z7 board should also be flashed on and off at a frequency of 2 Hz. Implement this operation using the same task that polls the push-button. Control the red LED state using the same method that you used before in Lab 1.

**Step 3:** Verify that the emergency stop condition (stopped motor and flashing red LED) persists until the Zybo Z7 board is reset by pressing the red PS-SRST reset push-button.

## Demonstrations

You will demonstrate to a TA both (1) the functioning of your emergency stop command, and (2) the ability to load and execute a sequence of motor positions and dwelling delays.

## Report Requirements

Your report must include the final versions of the commented source files that you used to implement Exercise 1 and 2. The report must take the form of **one file in pdf format** that is uploaded by the report deadline to eClass. The report guidelines are available on the eClass.

## Marking Scheme

The form and general quality of the report (clarity, organization, tidiness, spelling, grammar, sufficient explanatory comments in the code) will be considered when grading the reports.

The motor hardware connections will be worth 10% of the marks.

Demonstrations for Exercises 1 and 2 will be worth 22.5% and 22.5%, respectively. The report will be worth 45%.