Project Report

Surname	ZIM	**First name**	MD GOLAM TANVIR
Student number	EC18477		
Project Letter	B	**Project title**	Variable Speed Stepper Motor Control
GIT Hub Link	https://github.research.its.qmul.ac.uk/ECS642-714-EmbeddedSystems/ec18477ProjectB		

1 System Requirements

Given Requirement	Status	Detailed Behaviour
1. The start position of the motor is its position when the program starts.	Implemented	• The motor was able to get back to its original position throughout the program. Stop button caused the motor to get back its original position using the shortest path. It didn't matter whether the motor has completed any of its move or not. In both the cases of complete move and incomplete move (due to Stop Button Press), the Stop Button caused the motor to get back to its original position.
2. Two buttons control the stepper motor. The first button, the Run Button, causes the motor to move, provided that the motor is in the start position. The system has a number of moves, shown in the table below, in the time shown. On each press of the button, the motor advances to the next move, before returning to move 1.	Implemeted	• The Run Button caused the motor to move when the motor was in start poistion. • When the first move is complete, the Run Button caused the motor to take its second move and so on. • When the final move (move 6) was complete, pressing the Run button caused the motor take move 1. • In all cases, the number of turns, the direction and the time taken by each move were implemented as required.
3. A second button, the 'stop button', behaves as follows: • If the motor is moving, pressing the stop button causes it to stop. • If the motor is not moving, for example because the move has been completed, then pressing the button causes the motor to return to its start position, using the minimum number of steps.	Implemented	• When the motor was moving, pressing the Stop Button caused the motor to stop instantaneously. • If the motor has completed any of its move, the Stop Button caused the motor to return to its start position using minimum number of steps. • If the Stop Button was pressed while the motor was returning to Start Position, the motor was stopped from returning. The second press of Stop Button caused the motor to return to its start position using minimum number of steps.

		• Modulas operator was used to find out how far the motor is positioned from the Start Position. Two things were considered, 1) Was the net steps positive or negative 2) Is the remainder generated from Modulas Operation greater or lesser than 24? Using these two pieces of information, the minimum number of steps and the return direction were determined.
4. The stepper speed must be controlled by the PIT.	Implemented	• The stepper speed for each move was different and they were implemented by PIT. For each different move, different value was set into the timer, in correspondence to the different speed of each move.

2 Design System

The program was designed using Cyclic System which has a cycle of 10ms. The System has twp tasks, these are Poll Input and Control Motor States. In the design there are two input buttons and four GPIO output pins used which control the direction of the motor. The speed of the motor was controlled by PIT (Periodic Interrupt Timer). Start buttons caused the motor to move and the Stop button either caused it to stop (if moving or returning) or return to its Start position (if not moving).

2.1 Peripherals and Pins

The two input pins used in the design are as follows:

Pin No of MCU	Connection with Switch/Button
PTD6	Start button
PTD7	Stop button

These two pins were used to poll the input, i.e. to check if any voltage is present in the pins.
The four outputs which were connected to the motor are as follows:

Pin No of MCU	Connection with Motor
PTE30	Phase A+
PTE29	Phase A-
PTE23	Phase B+
PTE22	Phase B-

2.2 ISR

There is one ISR (Interrupt Service Routine) which is triggered when the Channel 0 of Timer counts upto the loaded value. Within the ISR, there is one single function being called and that is the `updateMotor(MotorId m)` function. This function updates the state of the motor. If the motor has outstanding steps, or is running continuously, calling this function each time causes the motor to take one step. If there are no outstanding steps, then calling this function is harmless: no change occurs.

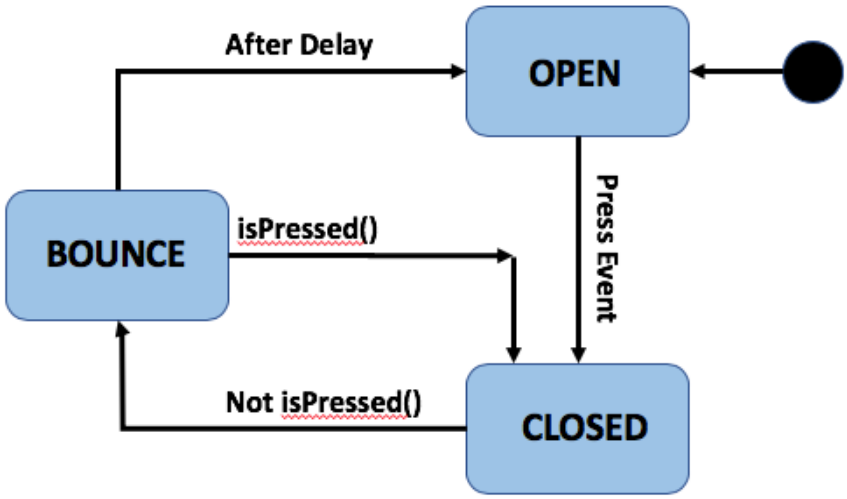
The frequency of calling ISR determines the speed of the motor. This frequency was set using the `setTimer(channel, timeout)` function. The timeout or the Load Value was calculated using the following equation:

$$\textbf{Timer\_Timeout\_value} = \frac{\textbf{Time\_Taken} \times 2^{32}}{\textbf{Number\_of\_Turns} \times 409.6}$$

Each move has different speed, hence difference load values were set in the timer for each move.

2.3 Task 1: Poll Input

This task has three states: *OPEN*, *CLOSED* and *BOUNCE*. In idle time, when the switch is not pressed the state is *OPEN*. When a press event occurs, i.e. the switch is pressed, the state is changed to *CLOSED*. After releasing the switch, the state is changed to *BOUNCE*. This state has been added to debounce the effect of switch debouncing, i.e. some unexpected spikes of voltages. So, in the *BOUNCE* state, a delay is introduced, during that MCU checks if the switch causing any spikes, after certain time, the effect is gone and the state changes from BOUNCE to *OPEN* state.



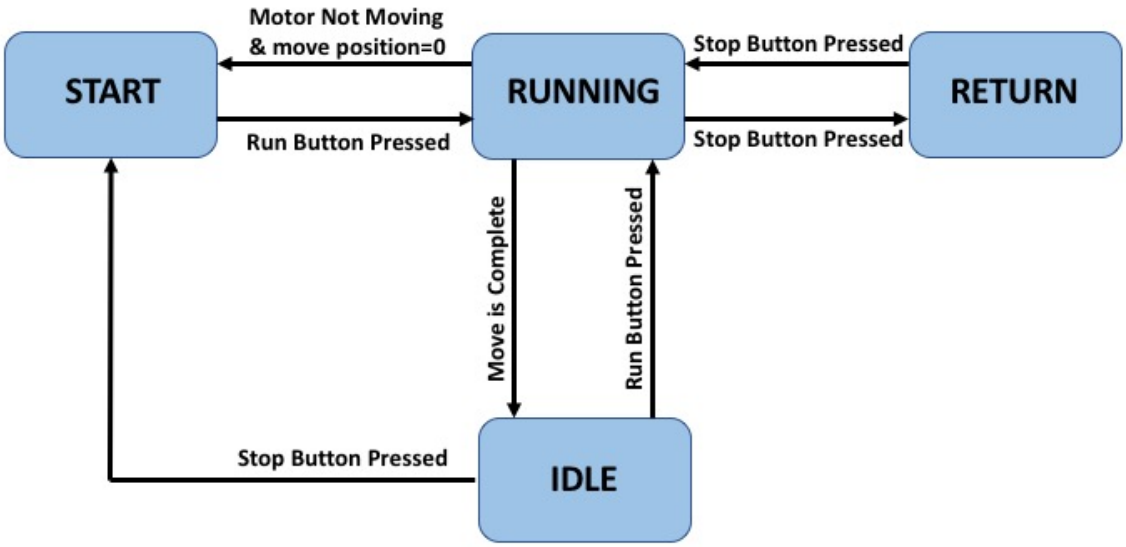
2.4 Task 2: Control Motor States

The following states are used in the design.

System States	Description
STATESTART	This is state of the system when the motor is at starting position.
STATERUNNING	When the motor is moving, whether its making step to make any of the move, or its making step to return – this state has been defined as *STATERUNNING.*
STATEIDLE	When the motor completes a move, the system enters into the *STATEIDLE*.
STATERETURN	When Stop button is pressed to stop the motor, the system enters into *STATERETURN.*

2.5 Communication between Two Tasks

The following State Transition Model shows how the System States change.



When the motor is initialized in the program and the system state is *STATESTART*. As soon as the Run button is pressed, the motor begins to move and the system state is *STATERUNNING*. The motor completes the move, thus the system enters into *STATEIDLE*. Then if the Stop button is pressed, the motor begins to run, thus entering into *STATERUNNING* and when it has returned to the initial position, the system comes back again to *STARTSTATE*.

In another scenario, if the Stop button is pressed, while the motor is moving, the motor is stopped and the sytem enters into *STATERETURN*. And if again Stop button is pressed, the motor begins to move using the shortest path and system state is now *STATERUNNING*. After the return move is complete, the system gets back to *STATESTART*.

In a third scenario, while the motor is returning back to initial position using shortest path, pressing the Stop button will cause the motor to stop and the system enters *STATERETURN*. If again Stop button is pressed, motor begins to move back to initial position and the system enters into and remains in *STATERUNNING*, until the motor gets back to initial position, the system then enters *STATESTART*.