Project Report

Surname	ZIM	**First name**	MD GOLAM TANVIR
Student number	EC18477		
Project Letter	D	**Project title**	DC Motor Speed
GIT Hub Link	https://github.research.its.qmul.ac.uk/ECS642-714-EmbeddedSystems/ec18477ProjectD/		

1 System Requirements

Given Requirement	Status	Detailed Behaviour
1. The system should be able to vary the speed of the motor using PWM.	Implemented	• The DC motor speed was controlled by varying the duty cycle of PWM module of the MCU. TPM0 module was configured for generating PWM in Channel 1.
2. Suggest a method to set the speed of the motor, with at least five different speeds (with no load). One of the speeds should be 'stopped'– that is, it should be possible to stop the motor.	Implemeted	• The speed of the motor was set and controlled using the Speed Button. Initially, after turning on the MCU, the motor is idle, i.e. not moving. Pressing the speed button caused it move at a set speed. Further pressing of the Speed Button increase the speed and so on. Five different speed was set, with each press the speed was increased to the next high level. • One of the speed of the motor was 'stopped', i.e. zero. • The motor has five different speed, each press causes the motor to run at next speed. When the motor is in highest speed, pressing the Speed button cause the motor return to Speed 1, i.e. 'stopped' or zero.
3. The system should display the (approximate) speed of the motor. It is ok to display the speed in bands (for exampleusing a number of LEDs). The speed in RPM should be calculated but need only be shown in the debugger. The actual speed must be shown, not the set speed, so that when the wheel is braked and the motor slows, the display changes.	Implemented	• The on board RGB led was used to show the speed of the motor in bands. Five different color was used to show five different speed. They are as follows: ✓ At Speed 01: Color was red. ✓ At Speed 02: Color was yellow. ✓ At Speed 03: Color was white. ✓ At Speed 04: Color was cyan. ✓ At Speed 05: Color was magenta. • The RPM speed was calculated and shown in the debugger and it matched with the actual speed of the motor which was measured manually. • While the motor was being braked, the motor speed slowed down and the

		speed shown in the debugger changed and also the LED band indicator.
4. The system should be able to change the direction of rotation and the direction be displayed.	Implemented	• There was a second button, Direction Button was used to change the direction of the motor. Initially the motor is set as to move counterclockwise direction, however, pressing the Direction Button causes the motor to alter its direction. • The direction of the motor was sensed using the pulse train coming from Sensor A and Sensor and analyzing their phase shift, the actual direction was determined and shown using an external LED. ✓ If the direciton is clockwise, the LED was ON/OFF. ✓ If the direciton is counterclockwise, the LED was ON/OFF

2 Design System

The program was designed using RTOS. The system has four tasks, two of them are for sensing the button position, one of them is frequency calculation based and the last one is for reversing the direction of the DC motor.

The two buttons are for two different purposes, the first one is Speed Button, which was used to control the speed of the DC motor. Each press of the button caused the increment of the speed of the motor until it reaches to the highest, then return to zero and increase again. The Direction Button was used to alter the direction, i.e. clockwise to counter-clockwise and vice versa. The DC motor has two outputs: Sensor A and Sensor B which give total 360 pulses for each rotation of the Geared Shaft.

2.1 Peripherals and Pins

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

Pin No of MCU	**Connection**
PTD6	Speed button
PTD7	Direction button
PTD5	*PTA13 (TPM1 capture channel 1)*

These two pins were used to poll the input, i.e. to check if any voltage is present in the pins. The following connections are between MCU and DC motor, described as follows:

Pin No of MCU and configuration type	**Connection with DC Motor**
PTB0 (GPIO Output)	IP1
PTB1 (GPIO Output)	IP2
PTA4 (TPM0 PWM output)	EN
PTA12 (TPM1 capture channel 0)	Sensor A
PTA13 (TPM1 capture channel 1)	Sensor B

PTB2 (GPIO Output)	Direction Indicator LED

2.2 ISR

There is one ISR (Interrupt Service Routine) which is triggered when there occurs a channel interrupt or overflow interrupt. There are two channels configured in TPM Input Capture Module, Channel 0 and Channel 1. When there is a rising edge occurred in either Sensor A or Sensor B, the ISR is called. Also when the TPM module counter overflows and resets, it triggers the ISR function `TPM1_IRQHandler()`.

Even though the ISR is triggered in above three instanes, CPU does take action only there is a rising pulse from Sensor A. It ignores interrupt generated due to rising edge pulse in Sensor B or Counter Overflow.

When there is a rising edge in Sensor A, CPU captures the Channel value, which is the value of Counter at the time of pulse arrives. Number of pulses arrive and CPU calculate the difference between the current timer count and the previous one: this is the difference that gives the times between pulses, which are called interval between pulses. When there are 8 such values are stored, they are added up and sent to a thread called `freqTask()` via `osMessageQueuePut()` function. The speed measurement is calculated on that thread.

Another task of this ISR is to show the direction of the motor by the external LED. It follows the following algorithm to determine the direction of the motor.

While there is a rising pulse in Sensor A:
If Sensor B = HIGH, then DC motor direction = CounterClockwise.
If Sensor B = LOW, then DC motor direction = Clockwise.

The volage level of Sensor B is sensed by connecting the Sensor B to the GPIO Input pin PTD5.

2.3 Thread 1: Poll Input for Speed Control – `buttonTask()`

This task has three states: *BUTTONUP*, *BUTTONDOWN*, *BUTTONBOUNCE*. In idle time, when the switch is not pressed the state is *BUTTONUP*. When a press event occurs, i.e. the switch is pressed, the state is changed to *BUTTONDOWN*. After releasing the switch, the state is changed to *BUTTONBOUNCE*. This state has been added to debounce the effect of switch debouncing, i.e. some unexpected spikes of voltages. So, in the *BUTTONOUNCE* 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 *BUTTONOUNCE* to *BUTTONUP* state.

When the Speed Control button is pressed, it changes the PWM duty cycle. There are five different PWM duty cycle values used (0%, 55%, 70%, 80%, 100%) in order to having five different speed for motor and these are used in an array. Whenever the Speed Button press occurs, PWM duty cycle is changed to the next value until it reaches the fifth value and comes back again to first one.

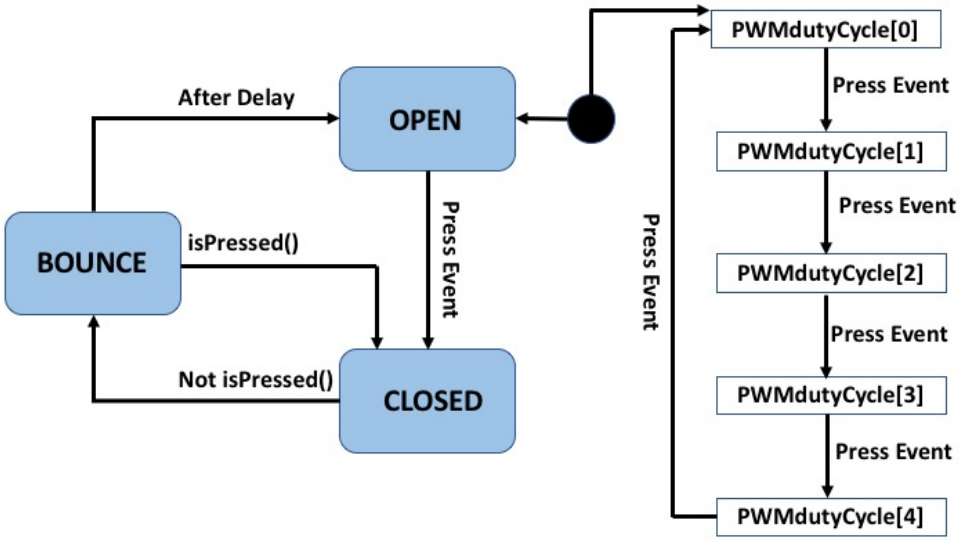

Figure 1: Task 1 STM

2.4 Thread 2: Poll Input for Altering Direction – `button2Task ()`

This task has exactly same three states as of thread 1. The only difference is whenever the Direction Control button is pressed, this thread sets the flag `MASK(PRESS_EVT)` in the event object `evtFlags` using the `osEventFlagsSet` function.

2.5 Thread 3: Changing the Direction of the DC Motor - `directionTask()`

This thread sets the direction of the DC motor. During startup, the initial direction is set as counterclockwise. When Direction control button is pressed in thread 2, this flag is set and thread 3 change alter the direction of the DC motor.

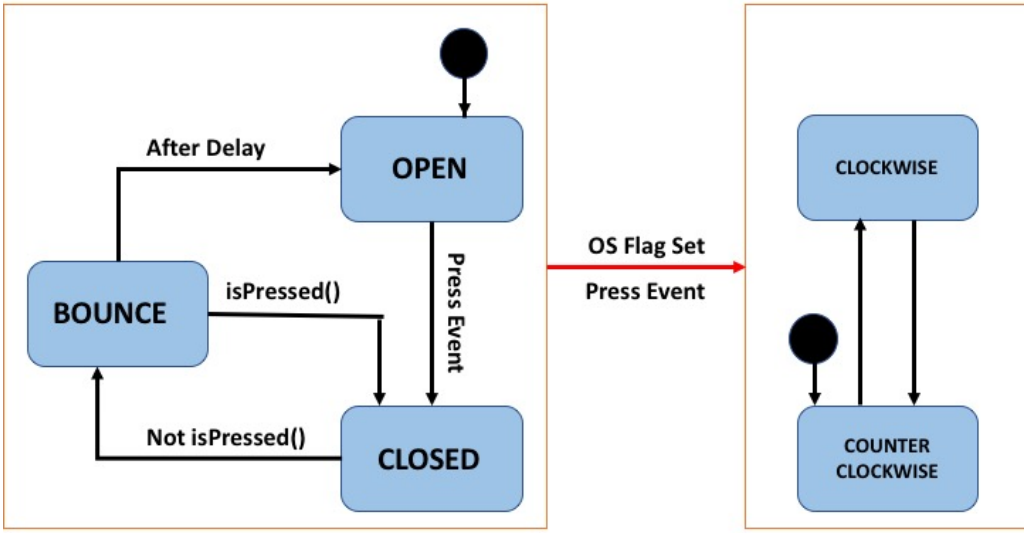

Figure 2: Task 2 and Task 3 STM & intercommunication

2.6 Thread 4: Calculating the Frequency and displaying the running Speed – `freqTask()`

This thread's task is to calculate the speed of the DC motor in RPM unit and show the speed in one of the five bands using RGB LED as described above. The following equation was derived and used to calculate the RPM speed of DC motor.

$$\textbf{DC Motor Speed (in RPM)} = \frac{\textbf{No of Intervals} \times 60}{\textbf{Intervals} \times \textbf{Pulses per Geared Shaft rotation} \times \textbf{Each Count Period}}$$

Here,

No of Intervals (between pulses) taken into consideration = 8.

Number of Pulses per Geared Shaft Rotation from one Sensor (A or B) = 3x60 = 180.

Time Period for each count of TPM Counter = 200ms/(2^{16}) = 3.05x10^{-6} seconds.

Intervals = Summation of 8 intervals (in terms of Counter value), this varies with the speed.

The speed was divided in 5 bands, Red, Yellow, White, Cyan and Magenta was used to show different speeds.

2.7 Communication between Two Threads using Flag

There is a communication between thread 2 and thread 3. As described before, in Thread 2, when the Direction Button is pressed, the flag `MASK(PRESS_EVT)` in the event object `evtFlags` is set. . Thread 3 enters into BLOCKED state unless the flag `MASK(PRESS_EVT)` in event object `evtFlags` is set. When the flag is set, this thread changes the direction either from Clockwise to CounterClockwise or CounterClockwise to Clockwise.

2.8 Communication using Message Passing

There is a message passed from ISR function `TPM1_IRQHandler()` to thread 4 `freqTask()`. When there are 8 interval values are stored in the ISR funtion by the `updateInterval()` function, a message is put into the message que identified by `interval_q_id`. using `osMessageQueuePut()` function. The actual message being sent from this function is the summation of 8 intervals.

The freqTask() thread waits to retrieves a message from the above message queue. Until the message arrives, this thread remains in BLOCKED state. When it retrives, it performs the speed calculation described above and based on that it changes the color combincation of on-board RGB LED to display the actual running speed.

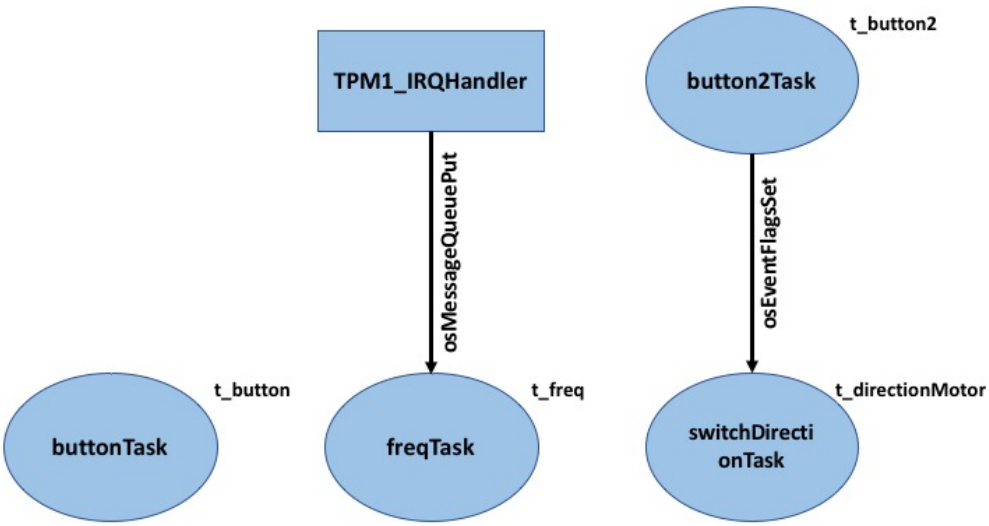

Figure 3: Communication between Tasks and ISR