ENGT5259 EMBEDDED SYSTEMS
Laboratory guidance notes
Part 5: Open Loop Control

Author: M A Oliver, J A Gow *v1.0*

Table of Contents

1 Aims and objectives....2
1.1Open Loop Motor Speed Control....2
2 Integrating the template into your current project....3
2.1Adding the files....3
2.2Creating new message IDs....3
3 pinchange module....4
3.1Initialization of the pinchange module....4
3.2Renaming the Encoder ISR....4
4 pwm module....4
4.1Initialization of the pwm module....4
4.2Duty cycle....4
5 revcount module....5
5.1Initialization of the revcount module....5
5.2Interrupt handler....5
5.3Calculating the speed....5
6 Test code....6
6.1Setting a new RPM value....6
6.2Displaying the actual RPM value....6

Version record

Version	Notes	Date	By
1.0	Initial version	30/03/22	MAO/JAG

PLEASE READ THIS CAREFULLY

This is the fifth of a series of guidance documents to get you started with your embedded systems project. They are aligned with the key aspects of the coursework specification, and thus this document should be read in conjunction with the coursework specification document. These documents will start off with some fairly close guidance - however as you develop your skills the notes will provide less and less detailed guidance and become more of an overview.

Note that the implementation is yours, and **you are encouraged to experiment**, or to try different approaches. As long as (a) you clearly understand the code you have written and

(b) your result complies with the specification, you will pass. There is no 'right answers' in this module. The benchmarks are: (a) have you hit the learning outcomes and (b) does your final project code comply with the specification.

Note that this module is not a C/C++ programming course. Most embedded system development is carried out in C/C++ and you should have had exposure to these languages as part of your undergraduate studies. If you are rusty, or are not familiar with this language there are plenty of tutorials available online (some links are available on the Blackboard shell) that can guide you. It is possible to learn C/C++ alongside this project. They are not especially 'difficult' languages to learn when you are using them to solve a problem and there are some 'templates' available to help.

You are advised to read this worksheet in its entity before commencing, as it contains useful information throughout.

1 Aims and objectives

This document covers an introduction to the motor drive and rps count implementation of the coursework specification. This should take you about 1 laboratory session to work through. It also details how the provided template files can be integrated into your existing project.

1.1 Open Loop Motor Speed Control

This work focusses on the development of firmware for reading and interpreting the rotary encoder on the ATmega328P Microcontroller Development Boards that are located in Q1-01 / Q2-01.

The motor will be driven via a power transistor using Pulse Width Module (PWM). PWM drive is an efficient method of driving a transistor as the transistor will be driven either hard-on (high collector current and low collector-emitter voltage) or hard-off (zero collector current and high collector-emitter voltage). This results in minimal heat dissipation. If the transistor was to be driven in a linear fashion, there would be a substantial collector current in combination with a substantial collector-emitter voltage resulting in significant heat dissipation.

The speed of the motor is obtained using a beam-breaker. An infra-red beam is fired at an infra-red sensitive photo transistor. A 3-position fan header is mounted to the motor. This breaks the beam three times per revolution. If we can count the number of times the beam is broken over a given time interval then divide by three, then the motor speed (in terms of revolutions per second) can be calculated.

The schematic for the motor drive and beam-breaker is shown in Figure 1. Note that the motor requires an external 12V supply.

At this point of the project, we are not looking at closed loop control. This will appear in the final worksheet.

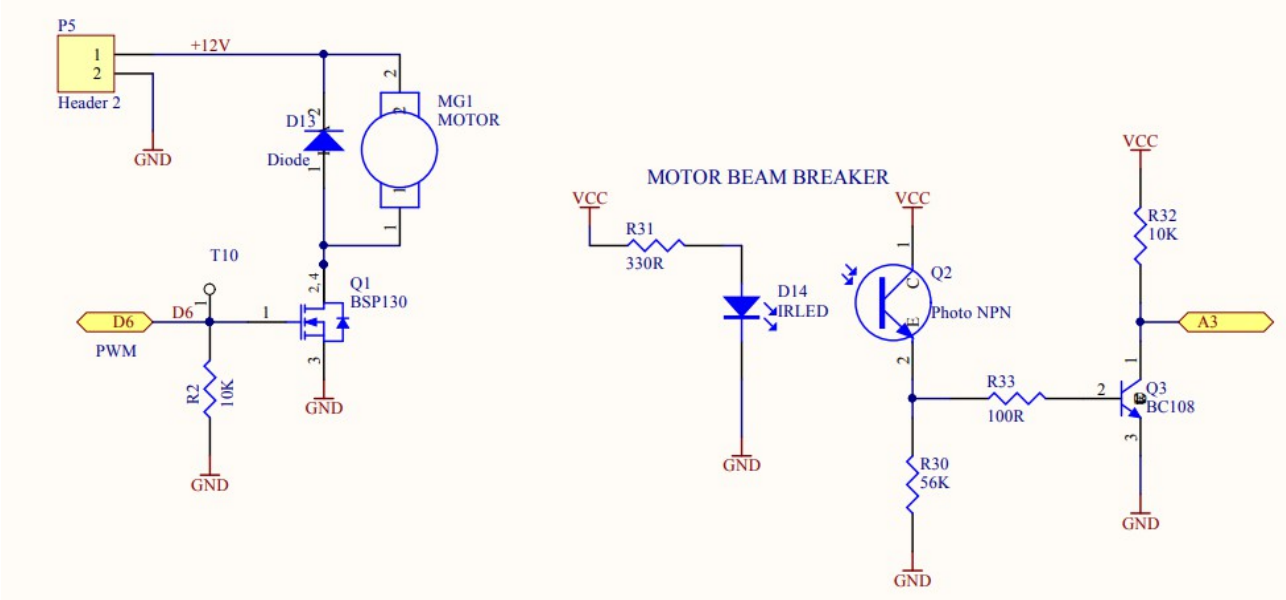

Figure 1: Motor Drive and Motor Beam Breaker.

2 Integrating the template into your current project.

The following steps are useful for integrating the required module files from the template into your current project.

2.1 Adding the files

Before embarking on this part of the process, please ensure that you have closed down your integrated development environment (most likely the Arduino IDE). ***You would be advised to back up your existing work at this stage before proceeding.***

The following files should now be copied into your containing directory (folder):-

pinchange.h

pinchange.cpp

pwm.h

pwm.cpp

revcount.h

revcount.cpp

For those using the Arduino IDE, when you restart the application, these files should now appear in your workspace.

2.2 Creating new message IDs

No new message IDs are required for this stage of the project.

3 pinchange module

Unfortunately due to the design of the development boards, the beam-breaker and the rotary encoder share the same interrupt. This causes a problem that is addressed by this module.

3.1 Initialization of the pinchange module

Identify the function `UserInit()` in your .ino file. You will need to add a single function call

```
PINInitialize();
```

to initialize the pinchange functionality.

Please note that the order in which functions are called within `UserInit()` is important.

Also, remember to `#include` the pinchange module into this file otherwise you will run into compilation errors.

3.2 Renaming the Encoder ISR

In your encoder.cpp module, you have an interrupt service routine called

```
ISR(PCINT1_vect)
```

You will need to rename this to

```
void ENCInterruptHandler(void)
```

Also you will need to provide an export prototype for this function in encoder.h.

4 pwm module

This module controls the PWM drive into the motor. You are encouraged to study the code in this module in conjunction with the ATmega328P data sheet.

4.1 Initialization of the pwm module

Identify the function `UserInit()` in your .ino file. You will need to add a single function call

```
PWMInitialize();
```

to initialize the pwm functionality.

Please note that the order in which functions are called within `UserInit()` is important.

Also, remember to `#include` the pwm module into this file otherwise you will run into compilation errors.

4.2 Duty cycle

The function

```
PWMSetDuty()
```

is responsible for setting the duty cycle of the pulse-width modulation drive. It accepts arguments ranging from 0 (or 0x00 in hexadecimal) for zero duty cycle, up to 255 (or 0xFF in hexadecimal) for maximum duty cycle.

5 revcount module

This module is used to determine the revolutions per second of the motor.

5.1 Initialization of the revcount module

Identify the function `UserInit()` in your .ino file. You will need to add a single function call

```
REVInitialize();
```

to initialize the rev counter functionality.

Please note that the order in which functions are called within `UserInit()` is important.

Also, remember to `#include` the revcount module into this file otherwise you will run into compilation errors.

5.2 Interrupt handler

The function

```
REVInterruptHandler()
```

is called by the `ISR(PCINT1_vect)` interrupt service routine if it recognizes that the interrupt has been raised by the IR beam-breaker.

Within this function, you have a simple TO-DO action item. That is to increment the value of `rpscount` by 1.

5.3 Calculating the speed

The function

```
REVGetRevsPerSec()
```

is required to calculate the revolutions per second. You have a TO-DO action item. That is to perform the calculation.

A variable called `rps` (standing for revolutions per second) has been declared for you. The function returns this value. You need to perform the calculation.

The variable `currpscount` counts the number of times the beam has been broken over a 0.26s interval.

If this value is multiplied by 100 it would yield the number of times the beam has been broken over a 26 second interval.

Now, it that result is divided by 26, it would yield the number of times the beam has been broken over a 1 second interval.

Remember that the beam is broken 3 times per revolution. Dividing your cumulative result by 3 will yield the number of revolutions per second.

6 Test code

In this update, you have skeleton code in control.cpp that can be used to test the PWM drive and IR beam-breaker. This can now be copied into your build.

6.1 Setting a new RPM value

The function

```
CTRLNewRPM()
```

was introduced in Laboratory worksheet 4. If you study this function you will see an extra line of code added to this that writes the Demanded RPS value to the PWM drive.

The function `CTRLWriteRPS()` performs this task.

6.2 Displaying the actual RPM value

There has been a modification to the

```
ControlTask()
```

function. Previously an emulated value has been used for the actual RPS value. Now, a true RPS value has been obtained using the `REVGetRevsPerSec()` function.

It will be a worthwhile task studying the code within this module.

Once you have implemented these tasks, you should be able to use the keypad and rotary encoder to drive the motor and see the reported speed of the motor.