A Realistic Motor Drive System using Pulse-Width Modulation

Comprehensive Software Plan

LE/EECS 3215 M

Nikhil Yates, Hatim Tayaballay[1]

[1] Student IDs and emails can be found in Appendix B

Table of Contents

Motivation

Throughout the course, we have been exposed to different embedded systems concepts. In order to understand many of them, we conducted practical learning in the lab using the HCS12 and Dragon Board. This project is a culmination of the extra work put into this course and ambition to build something that mimics what we see in real life.

The DC motor controller application is an adaptation of an electric sports car motor driver system. We will control the speed of the motor (rpm) and the acceleration of the motor by implementing 2 drive systems: eco and sport. Similar to how eco and sport drive-modes work in many modern ICE cars[2], we will slow how quickly the motor achieves its target rpm in 'eco' mode and reduce time to target speed in 'sport' mode.

Our project is centred around Pulse Width Modulation (PWM) concepts and Real Time Interrupts (RTI).

Overview

Hardware[3]

We will be using the MC9S12DG256CVPE chip that runs on the Dragon12-Light Trainer board. This chip connects with many peripherals such as the H-bridge, Terminal block, and LCD display on the board that are useful for this project.

The motor used in this project is from an unknown source. It is rated for 6V, and runs on the Dragon12. It will be connected to T4 (terminal block 4) by inserting the positive (red) wire into M1 and the negative (black) wire into M2. Unlike the servo motor that was connected to J7 in Lab 4, DC

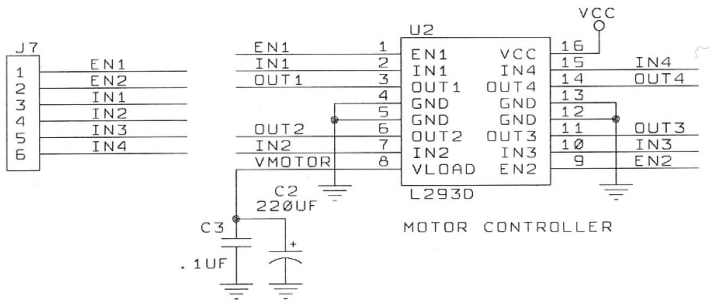


Figure 1. Dragon 12 schematic diagram for motor controller[4]

[2] Reducing power consumption in eco mode restricts performance and decreases acceleration so to optimise fuel consumption. https://autotrends.org/what-does-eco-mode-do/
[3] Refer to Appendix A for further hardware details
[4] From dragon12_schematics.pdf

motors do not have a ground wire. Thus we *must* use the H-bridge to prevent damage to the board. We decided to use VCC (voltage from the board) instead of an external current supply (EXT) because the power on the board is more than sufficient for driving the 6V DC motor. We did this by moving the jumper cap to cover the left 2 pins on jumper J25.

Software

Architecture Overview

Our program will run on a single main.c file within the project package. Asides from the original methods, we implement helper methods for character-integer conversion and time delays. We will keep the majority of processing within the main method. Although our main method is long, our code is traceable and readable.

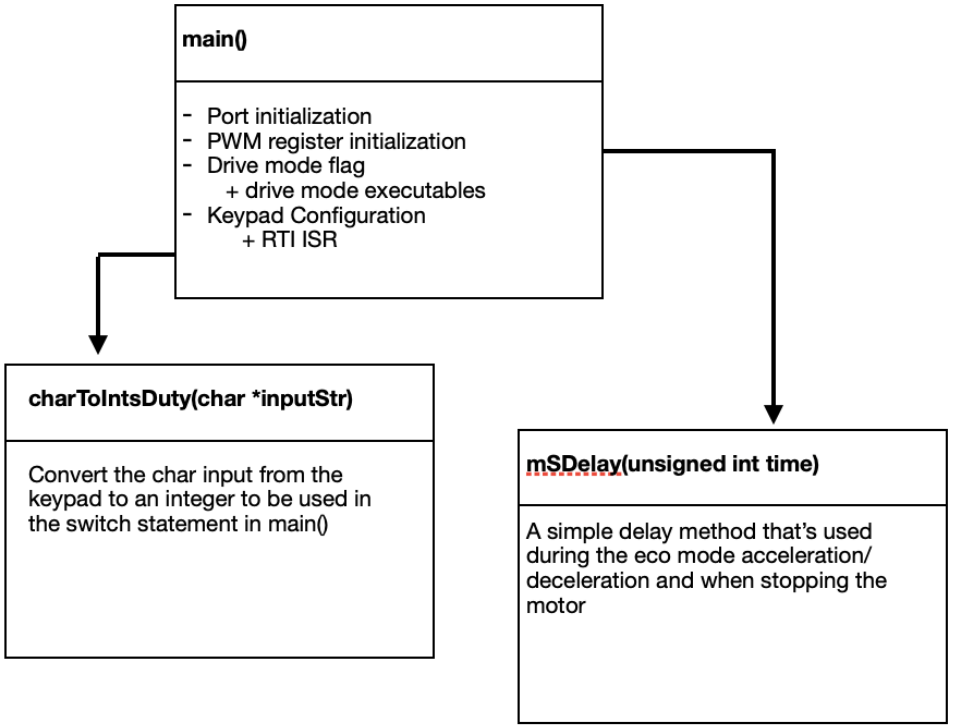


Figure 1. Software framework

The use of comments in development was strongly encouraged. As well, we interface with the LCD screen to display the current drive mode, 'speed' and any 'shifting' errors.

We will use our custom RTI ISR from Lab 4 to enable the keypad and prevent debouncing. Although we use busy loops within main for gear shifting (in eco mode) and stopping, the use of the ISR limits any additional overhead.

Design and Program Analysis

Hardware Configuration

A PWM-controlled DC motor needs to be connected to the H-bridge, Port B (PB0) and Port P(PP0). Below is a visual representation of the HCS12 pins we need to access:

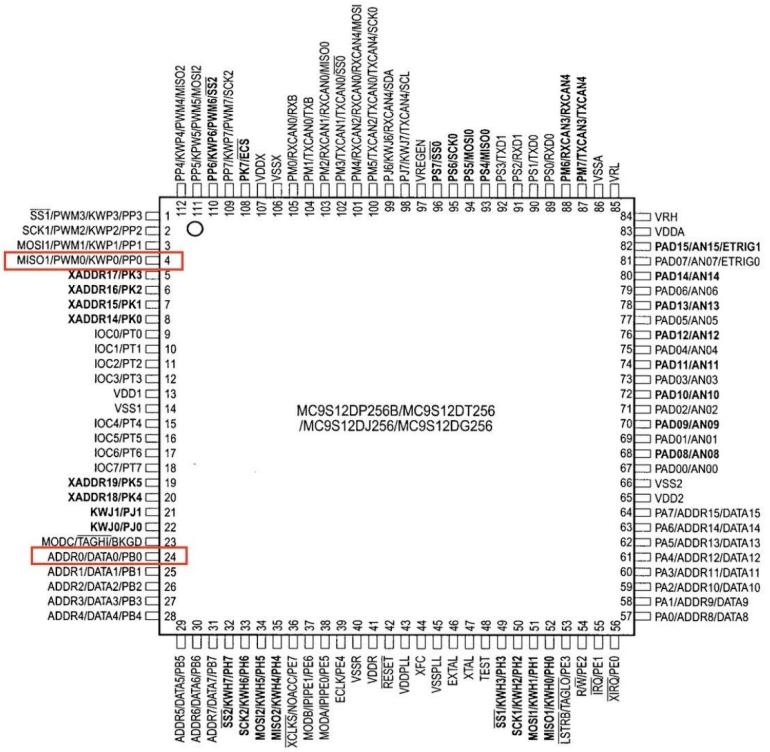


Figure 2. MC9S12 pin specification[5] (desired pins boxed in red) for DC motor PWM control

Port Configuration

We need to use PORTB and PTP to engage the H-bridge motor driver. However, as per the Dragon12 manual, we only need access to PP0 and PB0 for enabling terminal block 4 and routing voltage to M1 (of T4) respectively.

[5] MC9S12 pin assignments from Dragon12 resource manual.

```
DDRB = 0xff; // data direction register of port B is output
PORTB = 0x1; // enabling PB0
DDRP = 0xff; // data direction register of port P is output
PTP = 0x1; // enable PP0
```

Figure 3. Port initialization/assignments

Register Assignments

To enable PWM, we needed to select channel 0 to connect with the H-bridge. Note pin 4 in figure 3 indicates PWM channel 0 is needed to control PWM signals on PP0. The following are PWM register assignments:

```
PWMCLK = 0; // choose clock A
PWMPOL = ~0X10; /* this sets a high signal for channel 0 at the start
                 * of the period
                 */
PWMCTL = 0x0C; // select 8 bit mode[6]
PWMCAE = 0; // left-aligned duty cycle
PWMPER0 = 0x64; // period of 100 (1ms)
PWMPRCLK = 0x22; // setting the prescaler for clock A = 2
PWME = 0x1; // this enables PWM channel 0
```

Figure 4. PWM initialization and register assignments

Software Design

As noted in the software overview, the majority of the functionality of our program comes from our `main()` method.

Similar to our main methods in the labs, we have the bulk of the functionality within an infinite while loop. This means that input is continuously being read from the keypad.

```
DDRH = 0x00;
KeypadInitPort();

unsigned char c = KeypadReadPort
```

Figure 5. Keypad initialization and reading

After we have received a signal that a keypad button has been pressed, we check the input type. For buttons ‘A’, ‘B’, ‘C’, ‘D’, we set/change drive modes (A for eco and B for sport) and stop and start the motor (C and D respectively). For buttons ‘1’ through ‘5’ inclusive, we are able to control the speed of the motor.

We store the button input number (1-5) in an integer `currentSpeed`, and also within a character string. The speed is stored in a character string as well so we can use our method

[6] An 8-bit mode is suitable for running DC motors

`charToIntsDuty()` from lab 4 to convert the character input to an integer for a switch selection statement.

```
currentSpeed = c - '0';
inputStr[0] = c;
inputStr[1] = '\0';
dutyCycle = charToIntsDuty(inputStr)
```

Figure 6. Setting values for speed control variables

Next, we have a conditional statement that checks if the difference between the current speed and last speed, represented by integers `currentSpeed` and `lastSpeed`, is less than or equal to 1. If not, we display an error message on the LCD and deny the shift. Else, we shift.

To determine motor speed correlating to the keypad input, we have a switch statement that assigns the following values to an integer `dutyCycle`:

Numerical Keypad Input	`dutyCycle`
1	20
2	25
3	30
4	40
5	50

After assigning a value to our integer, we then check which drive mode we are in to see how we should accelerate. If we are in *eco* mode, integer `driveMode` (initialised in main) is 0 and 1 in *sport*. In *eco,* a loop delay is implemented as `lastSpeed` gradually increases to `currentSpeed`. Rationale behind the gradual acceleration can be found in *Motivation*. In *sport*, there is no delay between shifts and `PWMDTY0` instantly increases/decreases to the value of `currentDuty = dutyCycle`.

For reference, lines 329 through 350 are for changing the PWM signal based on user input and the condition that the 'gear' jumps are no more than 1.

The helper method `charToIntsDuty` takes a pointer to a character string and converts it to a single digit integer. Its only parameter was not changed, and the method is reused from lab 4.

`mSDelay` is a helper method from lab 1 that creates a time delay. It is used during our slowing and accelerating operations and when we are stopping the motor as mentioned in the software architecture overview.

"How To" and Features

How to use the DC motor program:

1. Load program
2. Start motor by pressing 'D' on the keypad
3. Choose drive mode: 'A' = Eco, 'B' = sport (on keypad)
4. Using keypad buttons '1'-'5', control the speed incrementally/decrementally
5. Press 'C' to slow the motor to a full stop
6. Program repeats starting at step 2

Our program is able to:

- Slow to a stop (regardless of drive mode)
- Prevents bad shifting by disallowing any gear shift larger than 2 (i.e., 1 jumping to 3 is not allowed)
- 2 input startup (pressing any gear or 'D')
- Start after full stop (i.e., 2 input startup after pressing 'C')
- Provide feedback to user about drive mode, current "speed", and if shifts are not allowed by printing to the LCD display

We have compiled an input-meaning table for reference.

Testing

In order to test our code, we tested the input using an oscilloscope before connecting our motor. This allowed us to test our duty cycles which would be used as a measure of the acceleration for our motor. This is both a safer and more efficient method of testing as we would put more stress on the motor if we ran it through the testing stage as well.

Once our code was proven to run, we connected the DC Motor to test the acceleration based on different input speeds (in our case the gears). We tested for outliers, which would show an error and not have an effect on the motor, and we also tested to make sure the gears are shifted in the correct order, not in random order. If inputted incorrectly, it would output an error message.

In terms of physical testing, in order to see increase as duty cycles are increased, we painted one of the endpoints of the connected flag, which shows a hue that can be followed to measure the increase or decrease of speed. Another factor that played in here was the sound

of the motor itself, as it was a clear indication of increased speed when the noise produced by the motor increased. This can be observed in the demo video.

Major Obstacles

1. **Uncertainty surrounding board configuration**

   Description
   Initially, we did not know how a DC motor should be connected to the board. We had first connected the DC motor as we did the servo motor for lab 4, and ended up resetting the board.

   Mitigation
   Referencing the Dragon12 manual, we searched for information surrounding "DC motor". After finding some material in 'Jumper Settings' (section 4.11) and 'On-board hardware features' (1.3) , we were able to do a trace on the dev board for jumper J62. This jumper is hardwired to the H-bridge (labelled U12) on the board. After doing some research about H-bridges and powering a DC motor using a microcontroller and H-bridge, we found the H-bridge was connected to both ports P (PP0, PP1) and B (PB0-PB3). We then referenced the MC9S12 pin assignments diagram (figure 1-3) and discovered PP0 had a PWM channel.

   After a brief testing spell and material referencing, we were able to correctly assign the correct values to Ports P and B and the PWM registers.

2. **Poor motor-board contact**

   Description
   Occasionally during our testing phase, the board would reset when the wires from the DC motor came into anything but perfect contact with M1 and M2 (Terminal Block 4/T4).

   Mitigation
   Using a flat-head screwdriver, we lifted the top of the contact plates (loosened it) of both M1 and M2, inserted the exposed wire completely, and then tightened the top of the contact plates to secure the wires.

3. Poor speed visibility

Description

After successfully changing the duty cycle via the PWM signal we were sending on PWM channel 0 and getting the motor to spin at different speeds, we found that other than the sound of the motor, it was difficult to distinguish a duty cycle change of 10% on camera.

Mitigation[7]

We attached a fan to the end of the drive shaft and painted one of the spokes. This drastically improved visibility. Below is a visual representation of the change we made to make the speed of the motor easier to track:

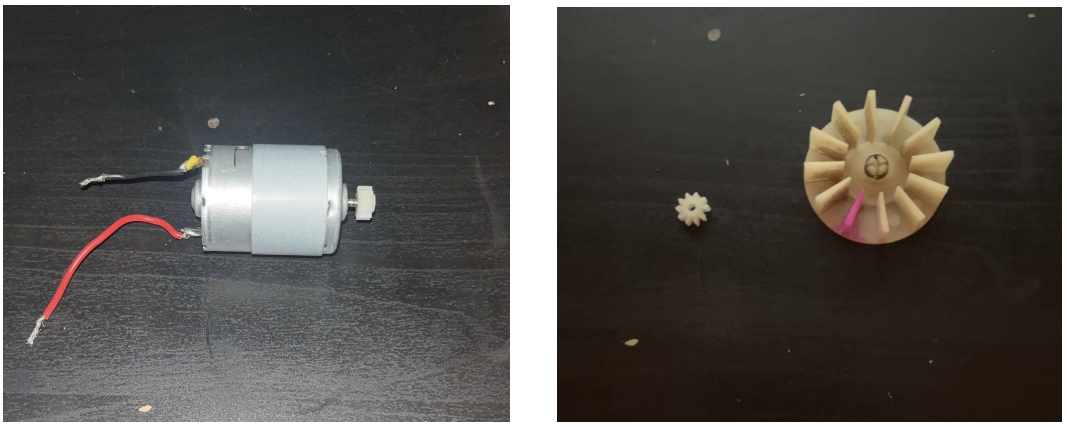


Figure 2. Initial motor 'wheel' vs size comparison of new motor fan with accent spoke[8]

[7] Refer to 'Testing' section for more detail

[8] View Appendix C 'Description of Materials' for final motor design

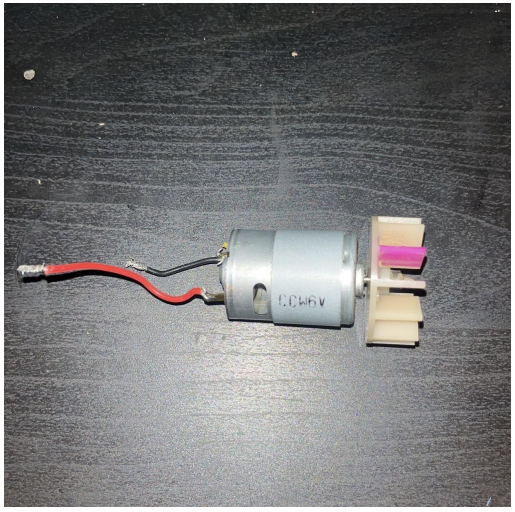
Video Demonstration Link

https://drive.google.com/file/d/1QoFjoBTSGk3C1vZ2eKew8t94kVmMLdiX/view?usp=sharing

Input Meaning Table

Input	Meaning
A	Enter *eco* mode. Acceleration/deceleration is limited.
B	Enter *sport* mode. Instantaneous acceleration, deceleration.
C	Stop the motor. The motor speed decreases gradually until PWMDTY0 = 0
D	Start the motor at PWMDTY0 = 20 or drive level 1.
1-5	Change the speed. Note only incremental/decremental changes are allowed

Appendix A - Hardware Descriptions

Name	Manufacturer	Description	Part Justification (if applicable)	Visual
6V DC Motor	N/A	- DC motors receive power as electrical energy and convert it into kinetic energy (mechanical rotation) via magnetic fields. - When the ‘wheel’ of our motor rotates, the rotor that drives the shaft is being forced to rotate because of mechanical movement. - The motor we used in this project is from an unknown source.	- When any of the Pins on the Dragon12 board are set to “high”, we know that the voltage passing through is rated at 5V. - Although the motor is rated for 6V, we are still able to spin it; the current it draws will simply be lower.	
MC9S12DG256CVPE Chip	Freescale (Motorola)	Refer to manual. Standard chip used throughout the course.	N/A	N/A
Dragon12-Light Trainer	EVB Plus	Refer to manual. Board used during the course.	N/A	N/A

Appendix B - Group Information

Author Name	**York ID**	**Email**
Hatim Tayaballay	217 888 199	hatimeh@my.yorku.ca
Nikhil Yates	218 121 517	nikhilyy@my.yorku.ca

Appendix C - Notes and Documentation

- To control the DC motor using PWM, we need to use the H-Bridge that is integrated on the the Dragon12 board
    - PPO/Pin 4 on the HCS12, PWM channel 0.
    - Use the T4 terminal block to connect the motor
- PWM can be achieved following the same technique as lab 4
- Basic design ideas:
    - Focus on single direction drive:
        - Implement functions for the different drive modes
            - When transitioning from lower to higher rpm, we want a transition that is consistent with the drive mode (i.e., slow transition for eco, fast for sport)
        - PWM duty cycle (PWMDTY0) will control the rpm of the motor and can be left in the same switch statement that we had for Lab4 (change PWM channel from 4 to 0).
- MOT3, MOT4 (T4) to use Vcc vs MOT1, MOT2 (T5) to use Vext

Implementation Notes:

- **Final Configuration:**
    - DDRB = 0xff; PORTB = 0x1; (PB0 set to high)
    - DDRP = 0xff; PTP = 0x1; (PP0 set to high)
        - Reasoning: need PP0 to enable T4 (terminal block 4), PB0 to set M1 to high -> this provides voltage to turn the motor in 1 direction)
    - T4 used: red wire motor connected to T4M1, black connected to T4M2

Appendix D - Resources and References

1. **https://www.pololu.com/docs/0J73/4.1#:~:text=The%20rated%20voltage%20of%20a,as%20its%20speed%20and%20torque.**

   To understand the effects of running a 6V DC motor on a circuit with a lower voltage rating.

2. **https://ie.rs-online.com/web/generalDisplay.html?id=ideas-and-advice/dc-motors-guide#:~:text=DC%20motors%20take%20electrical%20power,fixed%20within%20the%20output%20shaft.**

   General overview of how different types of DC motors work.

3. **dragon12_schematics.pdf**

   Further our understanding of how the H-bridge is connected (jumpers, pinout)

4. **dragon12_light_hcs12_manual_rev_d.pdf**

   Manual that was used to determine the ports of interest, interconnections, MCU pin assignments. Although we used ports B and P in previous labs, they are multipurpose and using this manual, we were able to understand the other I/O usage capabilities (namely H-bridge).

5. **https://autotrends.org/what-does-eco-mode-do/**

   This resource was used to understand how ‘eco’ mode works in ICE cars and its importance (as previously mentioned).

6. **https://www.modularcircuits.com/blog/articles/h-bridge-secrets/h-bridges-the-basics/**

   For understanding the basics of how H-bridge motor drivers work