Experiment 9: Controlling a DC Motor

0.1 Objectives

The purpose of this experiment is to learn how to generate a proportional output using the output compare resource on the PIC32MX processor so that you can implement Digital-to-Analog Conversion (DAC) with pulse width modulation (PWM). The proportional output will be used to control the speed of a DC motor.

Equipment List

- ChipKIT^{TM} Pro MX7 processor board with USB cable
- Microchip MPLAB ® X IDE
- MPLAB ® XC32++ Compiler
- Digilent DC Motor
- Digilent PmodDHB1

Overview

The information Digital I/O works well for generating and detecting discrete events. But the real world is continuous; thus, for microprocessors to have value, they need to have the capability to input and output analog signals. A digital-to-analog converter (DAC) converts a digital, or binary, value to an analog voltage. An analog-to-digital converter (ADC) converts a voltage in a specified range to a binary value that represents the magnitude of the signal.

DC Motor

A Direct Current (DC) motor is a two-wire (power and ground) continuous rotation motor which converts DC electrical energy to a mechanical energy. The principle of a DC motor is that whenever a current-carrying conductor is placed in a magnetic field, it experiences a mechanical force (as shown in Figure 23). When you supply power, a DC motor will start spinning until that power is removed. Most DC motors run at a high RPM (revolutions per minute).

Pulse Width Modulation (PWM)

There are many different ways to control the speed of motors, but one very simple and easy way is to use **Pulse Width Modulation (PWM)**. As its name suggests, pulse width modulation speed control works by driving the motor with a series of "ON-OFF" pulses and varying the duty cycle, the fraction of time that the output voltage is "ON" compared to when it is "OFF", of the pulses while keeping the frequency

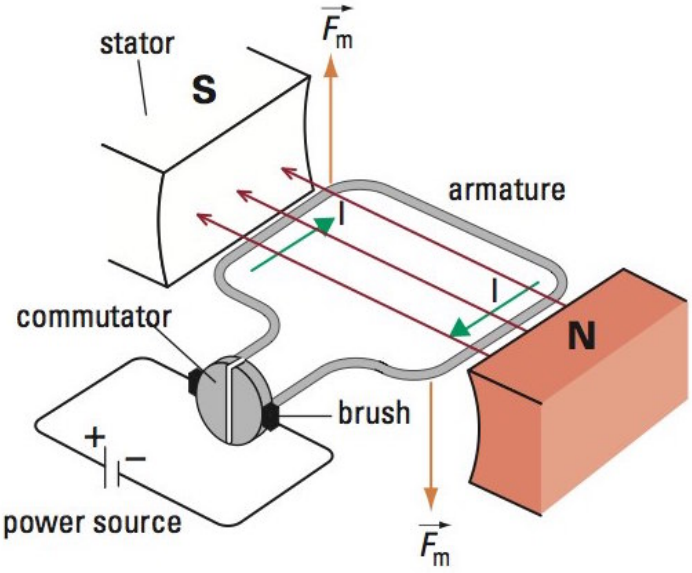

Figure 23: A simple DC motor.

constant. The power applied to the motor can be controlled by varying the width of these applied pulses and thereby varying the average DC voltage applied to the motors terminals. By changing or modulating the timing of these pulses the speed of the motor can be controlled, i.e., **the longer the pulse is "ON", the faster the motor will rotate and likewise, the shorter the pulse is "ON" the slower the motor will rotate**.

In other words, the wider the pulse width, the more average voltage applied to the motor terminals, the stronger the magnetic flux inside the armature windings and the faster the motor will rotate as it is shown in Figure 24.

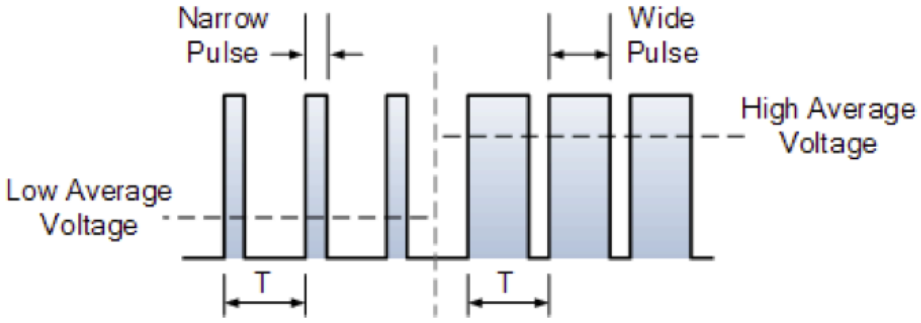

Figure 24: Pulse Width Modulated Waveform.

The Digilent PmodDHB1 (Figure 25) is a dual H-Bridge motor driver that is capable of driving two DC motors, a bipolar stepper motor, and other devices with inductive loads

The DC motor driver circuit provided by the PmodDHB1 allows control of the motor speed by connecting the Output Enable pin to the PIC32 PWM output as well as motor direction control by connection to another PIC32 I/O pin.

The Pulse Width Modulation (PWM) is controlled by the PIC32MX processor **Output Compare (OC)** module [8]. For this experiment, we will use Timer 2 to generate the PWM cycle frequency. The DC motor is connected to **JD** on the ChipKIT^{TM} Pro MX7 processor board. This connection provides access to output compare channel 2 (OC2) that we will use to generate the PWM output. It is important also to connect J4 header (motor voltage) of PmodDHB1 to the motor, where pin 1 is the motor power and pin 2 is power

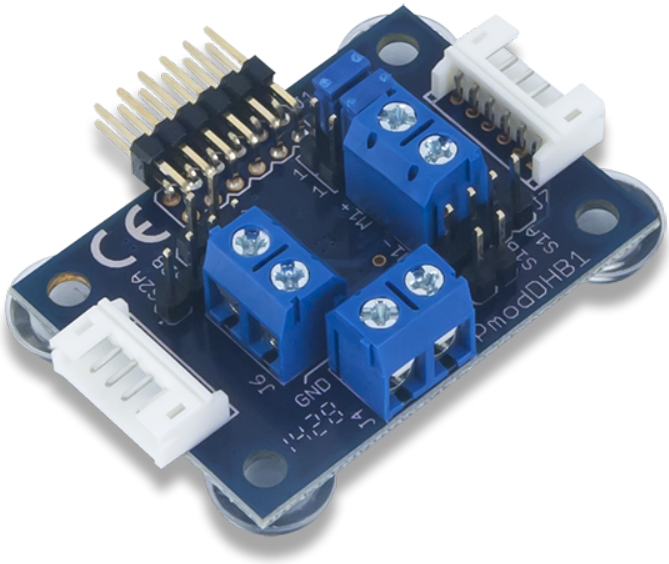

Figure 25: Dual H-Bridge PmodDHB1.

supply ground.

Output Compare Peripheral Library

The Output Compare module is primarily used to generate a single pulse or a series of pulses in response to selected time base events. Output Compare module operation essentially requires a timer and one or two compare values. Users have an option to select either a 32-bit timer or a 16-bit timer. The timer can be programmed to count at a desired frequency and count up to a desired period value.

The Output Compare module compares the value of the timer with the compare values depending on the selected operating mode. When a compare match occurs between a timer value and compare values, the Output Compare module outputs a change of state on its output pins in accordance with the chosen Output Compare compare mode logic. Either a single pulse or a sequence of pulses may be generated.

One of the key operating modes of OC module is the Pulse-Width Modulation (PWM) mode. In this mode, output goes high when a compare match occurs between timer and the pulse width value (duty cycle). The output is reset back to its initial state when the timer resets after attaining its maximum count.

To use the OC driver, you can use the MPLAB Harmony Configurator (MHC). Make sure to enable the **interrupt mode** so as to generate the required ISR function for event handlers. Then in the drop menu of **OC Driver Instance 0**, choose an OC mode (for **PWM mode with edge aligned mode**[6]) and Timer 2 (16 bit) for an OC **Timebase timer**. **Pulse-width values** are used only in Dual compare and PWM modes. In PWM mode, the buffer value provides the initial duty cycle for the first time period, while all later time cycles employ the 'pulse-width' value as the duty cycle value. The calculation of the pulse-width value of an OC depends on the Timer period value. For example, if Timer period is set to 78, the pulse-width value should be 39 so as to generate a 50% duty cycle.

[6]In Edge-aligned PWM mode, the Output Compare module output is driven 'High' whenever the timer resets, and is driven 'Low' on a compare match with the pulse-width value.

PWM Period

The PWM period (PR_{PWM}) is specified by writing to the *Timer y* Period register, PR_y. The PWM period can be calculated using the following formula:

$$\mathrm{PR}_{PWM} = \frac{(1 + PR_y) * \mathrm{Prescale}_y}{\mathrm{PBCLK}} \tag{2}$$

where $PR_{PWM} = \frac{1}{freq_{PWM}}$.

For example, if the desired PWM frequency is 52.08KHz, and PBCLK = 10MHz and timer prescale is 1:1, then timer's period will be equal to 191 (based on Equation 2).

In order to start the OC module, you can use the following function:

```
DRV_OCx_Start();
```

where x is the OC instance number.

Also, in order to set a 16-bit pulse width for Output Compare module output, you can use the following command:

```
void DRV_OCx_PulseWidthSet(uint16_t pulseWidth);
```

Project Tasks

The objective of this experiment is to implement an open-loop motor speed control. For this experiment, the motor speed control will be set to two tasks.

Task 1

Implement an ON/OFF system for a DC motor, where ON means a duty cycle of 95% when BTN1 is pressed and OFF means 5% when BTN2 is pressed.

Task 2

Then, implement a speed control system for a DC motor to four fixed PWM duty cycle values as shown in Table 9.

BTN1	**BTN2**	**PWM**
OFF	OFF	25%
ON	OFF	50%
OFF	ON	75%
ON	ON	95%

Table 9: Task 2- Button-controlled PWM.

Project Requirements

- Detect button uses the change-notice interrupt similar to the experiment 6 implementation. The CN interrupt is to use a 20 ms software delay for switch debouncing.
- Configure Timer 2 to generate an interrupt each millisecond.
- Configure PWM output compare to use 16-bit Timer 2.
- Button Detect ISR (refer to Experiment 7).

  - Sets LED2 on entry and clears LED2 on exit.
  - Removes button contact bounce with 20ms software delay.
  - Reads button state.
  - Decodes buttons and sets PWM in accordance with Table 9.
  - Sets the motor PWM.
  - Clears CN interrupt flag.
- Timer2 ISR.
  - Toggles LED1.
  - Clears T2 interrupt flag.