# Brushed DC Motor Controller

## Introduction

This lab will design a proportional-integral-derivative (PID) controller for a brushed DC motor.

## Procedure

For simplicity, we will design a non-reversible motor.

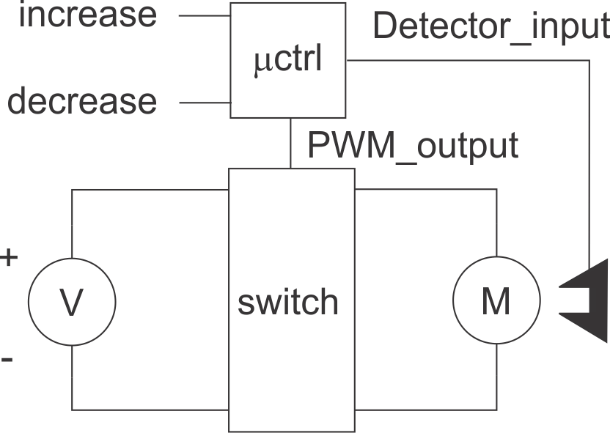


Figure 1: Motor controller system.

Figure 1 shows the basic organization of the motor controller. The microcontroller provides a pulse-width modulated signal PWM\_output to a switch. The voltage source attached to the switch has sufficient drive to operate the motor. The shaft encoder produces Detector\_input to count transitions on the shaft encoder. The user can adjust the speed setting with two buttons increase and decrease.

The switch and shaft encoder are built as separate circuits. Most of the interface is built within the PSoC 5LP.

This design uses the Microchip TC4423 MOSFET driver to drive the motor. The motor power supply is a 6V wall-wart style AC/DC converter.



Figure 2: The PSoC 5LP motor controller board.

Figure 2 shows the PSoC 5LP motor controller. The prototyping area includes a CMOS power switch. Its control input comes from the pulse width modulator. It switches the output of a DC power supply used to drive the motor. The prototyping area also includes pullup resistors for the shaft encoder illumination source and detector. The shaft encoder target is attached to the motor shaft.

Q1: Choose appropriate resistor values for the shaft encoder’s photodiode and photodetector.

Open the PSoC Creator project PWM\_BasicTest01 and open the TopDesign.cysch schematic.

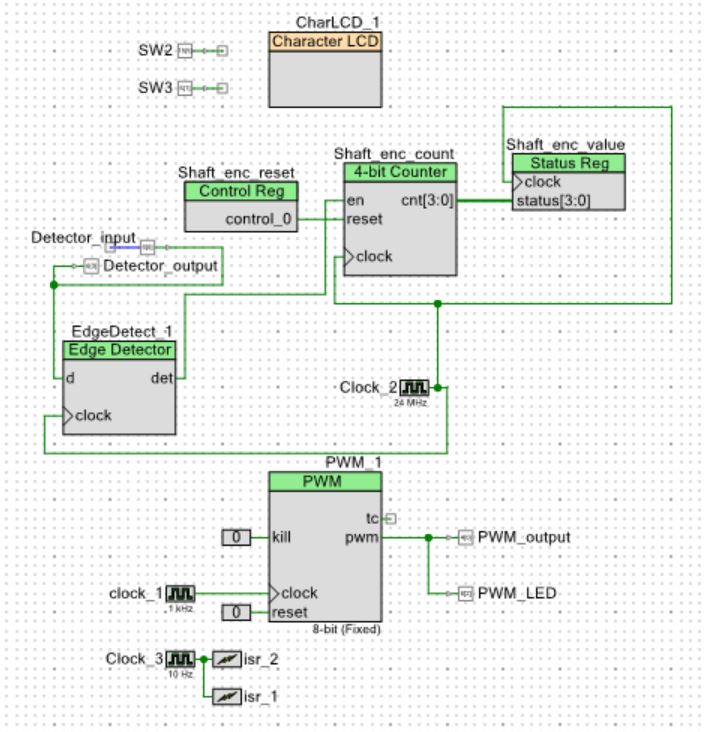


Figure 3: PSOC 5LP schematic for the motor controller.

Figure 3 shows the schematic for the internal PSOC 5LP section of the interface:

* The pulse width modulator PWM\_1 generates the pulse width modulated control for the motor duty cycle. Its period is determined by clock\_1. The PWM unit’s compare value is the comparison value used to determine the duty cycle.
* An interrupt request isr1 is used to update the motor control. Its rate is determined by Clock\_3.
* The shaft encoder generates the Detector\_input value, which is processed by an edge detector before being used to increment a counter. The Shaft\_enc\_value register is used to read the counter value. The Shaft\_enc\_reset register is used to write the counter’s reset signal.
* A separate interrupt, isr\_2, is used to process the shaft encoder data. Its rate is also determined by Clock\_3.
* Switches SW2 and SW3 are used by the user to increase and decrease the PWM’s compare value and thus control the duty cycle.
* The LCD displays the value of several internal variables.

The PWM Period register controls its period. It can be read using PWM\_1\_ReadPeriod(). The Compare register provides the comparison value to determine the end of the duty interval. It can be read using PWM\_1\_ReadCompare() and written using PWM\_1\_WriteCompare().

Open main.c:

* The update\_rot() function computes the shaft speed from the shaft encoder count.
* The open\_loop\_brushed() function provides open-loop control that simply copies the command value into Compare.
* The closed\_loop\_brushed() function computes a PID control function to generate the command value that is loaded into the PWM Compare register.
* The main() function first initializes the system components. It then executes an infinite loop to update the LCD with three values: PWM period, PWM compare, and shaft speed.

Open isr1.c. This file may be regenerated by the PSoC Creator tools. Any code you add should be put between special comments like this:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Place your includes, defines and code here

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* `#START isr\_1\_intc` \*/

#include "Status\_Reg\_2.h"

#include "Control\_Reg\_2.h"

extern void open\_loop\_brushed();

extern void closed\_loop\_brushed();

/\* `#END` \*/

These declarations introduce registers and the two control functions. The CY\_ISR(isr\_1\_Interrupt) function is the interrupt service routine. We add code to call the appropriate control function, in this case the closed loop version:

/\* `#START isr\_1\_Interrupt` \*/

closed\_loop\_brushed();

/\* `#END` \*/

Open isr2.h. This file has a similar structure. We add a declaration for update\_rot() and call that function within CY\_ISR.