Laboratory Assignment Number 7 for ME 218b

Pre-Lab Due by 11:59 pm on January 28, 2026
Due by 5:00 pm on January 30, 2026

Goals:

In this lab you will explore the behavior of a DC motor being driven by Pulse Width Modulation (PWM) using a Drive-Brake mode. You will also gain experience implementing closed loop speed control of the DC motor.

Part 0: Pre-Lab

Background:

The pre-lab should be completed after you have read through the entire lab assignment, but before you go into the lab to begin your work there. Completing the pre-lab, which can be done without any special tools or resources, will allow you to be most efficient with your time in the lab. We judge the pre-lab submission based on a reasonable effort in pre-lab document submitted.

Assignment:

- ☐ 0.1) Design a circuit to drive the DC motor bi-directionally using Drive-Brake mode using an L293NE (or SN754410) as the drive stage with a motor voltage of 5V. The schematic should be created in KiCad.
- ☐ 0.2) Figure out how you will measure the electrical time constant of the motor as requested in Part 1.1. Include a circuit diagram of the measurement circuit.
- ☐ 0.3) Design the software that you will use for Part 1. Include pseudo code for program(s).
- ☐ 0.4) Design the software that you will use for Part 2. Include pseudo code for program(s).

In the report:

Include your answers to all of the sections of the pre-lab.

Pseudocode for part 1

Time Constant Measurement Service (Part 1.1)

```
RUN FUNCTION:
  INPUT: ThisEvent

  Switch on CurrentState:

    Case IDLE:
      If ThisEvent is ES_NEW_KEY (e.g., 't' for toggle start):
        Start ES_Timer (e.g., TOGGLE_TIMER) with desired half-period
          (e.g., 1ms for a 500Hz square wave to start)
        Set RA2 HIGH
        Set PinState to HIGH
        Transition to TOGGLING state

    Case TOGGLING:
      If ThisEvent is ES_TIMEOUT from TOGGLE_TIMER:
        Toggle PinState
        Write PinState to RA2
        Restart TOGGLE_TIMER with same period

      If ThisEvent is ES_NEW_KEY (e.g., 's' for stop):
        Stop TOGGLE_TIMER
        Set RA2 LOW
        Transition to IDLE state

      If ThisEvent is ES_NEW_KEY (e.g., '+' or '-'):
        Adjust timer period up or down
        Print new frequency to terminal

  Return ES_NO_EVENT
```

Motor Control Service (Parts 1.2, 1.3)

Init function

```
  // Initialize Direction Control Pin (RB2, IN 1A)
  Disable analog on RB2 (ANSB2 = 0)
  Set RB2 as output (TRISB2 = 0)
  Set RB2 = 0 (default forward direction)

  // Initialize Direction Input Pin (RA1)
  Disable analog on RA1 (ANSA1 = 0)
  Set RA1 as input (TRISA1 = 1)

  // Initialize ADC for potentiometer on AN0
  Call ADC_ConfigAutoScan(BIT0HI)

  // Initialize PWM (OC4 on RB13, using Timer2)
  Call InitPWM() (From previous lab)

  // Initialize Input Capture (IC2 on RB9, using Timer3)
  Call InitInputCapture() (From previous lab)

  // Start periodic timer for ADC reading (10 Hz)
  ES_Timer_InitTimer(MOTOR_TIMER, ADC_UPDATE_TIME)

  // Post ES_INIT
  Post ES_INIT to self
  Return TRUE
```

Run function

```
  INPUT: ThisEvent
  ReturnEvent = ES_NO_EVENT

  Switch on ThisEvent.EventType:

    Case ES_TIMEOUT:
      If ThisEvent.EventParam == MOTOR_TIMER:

        // --- Read Potentiometer ---
        // --- Calculate Duty Cycle ---
        // Map ADC (0-1023) to duty cycle register value (0-PWM_PERIOD)
        DutyCycleReg = (adcResult[0] * PWM_PERIOD) / 1023
```

```
        // Calculate percentage for display
        CurrentDutyCyclePercent = (adcResult[0] * 100) / 1023

        // --- Read Direction Input ---
        DirectionInput = Read PORTAbits.RA1

        // --- Apply Direction and Duty Cycle ---
        If DirectionInput == 0:     // Forward
           Set IN_1A (RB2) = 0
           Set OC4RS = DutyCycleReg
        Else:                       // Reverse
           Set IN_1A (RB2) = 1
           // Invert duty cycle for Drive-Brake with IN1A=1
           Set OC4RS = PWM_PERIOD - DutyCycleReg

        // --- Calculate and Print RPM ---
        // Safely read volatile period (disable interrupts briefly)
        __builtin_disable_interrupts()
        PeriodSnapshot = CurrentPeriod
        __builtin_enable_interrupts()

        If PeriodSnapshot > 0:
           RPM = RPM_NUMERATOR / PeriodSnapshot

           // Print for data collection (CSV format)
           Print "%d, %d\r\n", CurrentDutyCyclePercent, RPM
        Else:
           // Motor stopped or no encoder pulses
           Print "%d, 0\r\n", CurrentDutyCyclePercent

        // Restart timer
        ES_Timer_InitTimer(MOTOR_TIMER, ADC_UPDATE_TIME)

    Default:
       // Ignore other events

Return ReturnEvent
```

Pseudocode for part 2 modification to motor service from part 1

New init function

```
  Save Priority to MyPriority

  // Initialize Direction Control Pin (RB2, IN 1A)



  // Initialize ADC for potentiometer on AN0
  // Initialize PWM (OC4 on RB13, using Timer2)
  Call InitPWM()

  // Initialize Input Capture (IC2 on RB9, using Timer3)
  Call InitInputCapture()

  // Initialize Timing Pin for ISR measurement (Part 2.5)
  Set TIMING_PIN_TRIS = 0 (output)
  Set TIMING_PIN = 0 (initially low)

  // Initialize Control Loop Timer (Timer4)
  Call InitControlTimer()

  // Print startup message

  // Start periodic timer for ADC/reporting (10 Hz)
  ES_Timer_InitTimer(MOTOR_TIMER, ADC_UPDATE_TIME)

  Post ES_INIT to self
  Return TRUE
```

New run function

```
 Switch on event type:

    Case ES_TIMEOUT:
      If ThisEvent.EventParam == MOTOR_TIMER:

        // --- Read Potentiometer for Speed Command ---
        Declare adcResult[1]
        ADC_MultiRead(adcResult)
```

```
        // Map ADC (0-1023) to commanded RPM
        float NewTargetRPM = (float)adcResult[0] * MAX_COMMANDED_RPM / 1023.0

        // Update target RPM (used by control ISR)
        TargetRPM = NewTargetRPM

        // --- Print for monitoring (CSV format) ---
        Print "%.1f, %.1f, %.1f, %.1f\r\n",
            TargetRPM, CurrentRPM, LastRPMError, RequestedDuty

        // Restart timer
        ES_Timer_InitTimer(MOTOR_TIMER, ADC_UPDATE_TIME)

    Default:
      // Ignore other events

  Return ReturnEvent

Control loop isr, priority 5 lower than encoder isr
  // Clear interrupt flag first
  IFS0CLR = _IFS0_T4IF_MASK

  // Raise timing pin on entry (for Part 2.5)
  TIMING_PIN = 1

  // --- Local variables (static for speed) ---
  static float RPMError
  static uint32_t ThisPeriod
  static float Rpm

  // --- Get Current Motor Speed ---
  ThisPeriod = CurrentPeriod

  If ThisPeriod > 0:
    Rpm = PER2RPM / ThisPeriod
  Else:
    Rpm = 0

  // --- Calculate Error ---
  RPMError = TargetRPM - Rpm
```

```
// --- Accumulate Error for Integral ---
SumError += RPMError

// --- Calculate Requested Duty Cycle (PI Control Law) ---
RequestedDuty = (pGain * RPMError) + (iGain * SumError)

// --- Clamp Output and Apply Anti-Windup ---
If RequestedDuty > 100:
   RequestedDuty = 100
   SumError -= RPMError    // Anti-windup: undo the accumulation

Else If RequestedDuty < 0:
   RequestedDuty = 0
   SumError -= RPMError    // Anti-windup: undo the accumulation

// --- Update for display ---
LastRPMError = RPMError
CurrentRPM = Rpm

// --- Apply Duty Cycle to PWM ---
Call SetDuty(RequestedDuty)

// Lower timing pin before exit (Part 2.5)
TIMING_PIN = 0
```