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 1: Driving a DC Motor Using PWM

- ☐ 1.1) To get the best performance out of the motor, it is useful to know the electrical time constant. For minimum current ripple, you will want to set the PWM frequency to at least a frequency of one over the electrical time constant. Implement the necessary software to help you to measure the electrical time constant of the motor on the lab bench. What is the time constant?
- ☐ 1.2) Implement your DC motor driver design from Part 0.1 and Event Driven software design to read the A/D converter connected to the potentiometer and adjust the speed. Hook it up to the motor and get the motor to run with the speed controlled by the potentiometer and the direction set by the digital input pin. Set the PWM frequency based on the time constant that you determined in part 1.1
- ☐ 1.3) Map the Duty Cycle vs RPM transfer function for the motor/driver combination. Create a graph to visualize the data.
- ☐ 1.4) Demonstrate your working software to a TA/Karl and get signed off on the check-out sheet.

In the report:

Include

- The schematic diagram of the circuit you used, design calculations confirming the suitability of all components used. Be sure to indicate on the schematic which bits of which ports of the PIC32 you used, and please include pin numbers.
- A description of your measurement technique and results from your time constant measurement in Part 1.1,
- A graph of the DC vs. Speed data from Part 1.3.
- A Highlighted version of the source code to the program.
- Be sure to be signed-off by a TA.

Circuit diagram included as a separate pdf.

Design calculation's part 1 attached as a separate pdf,

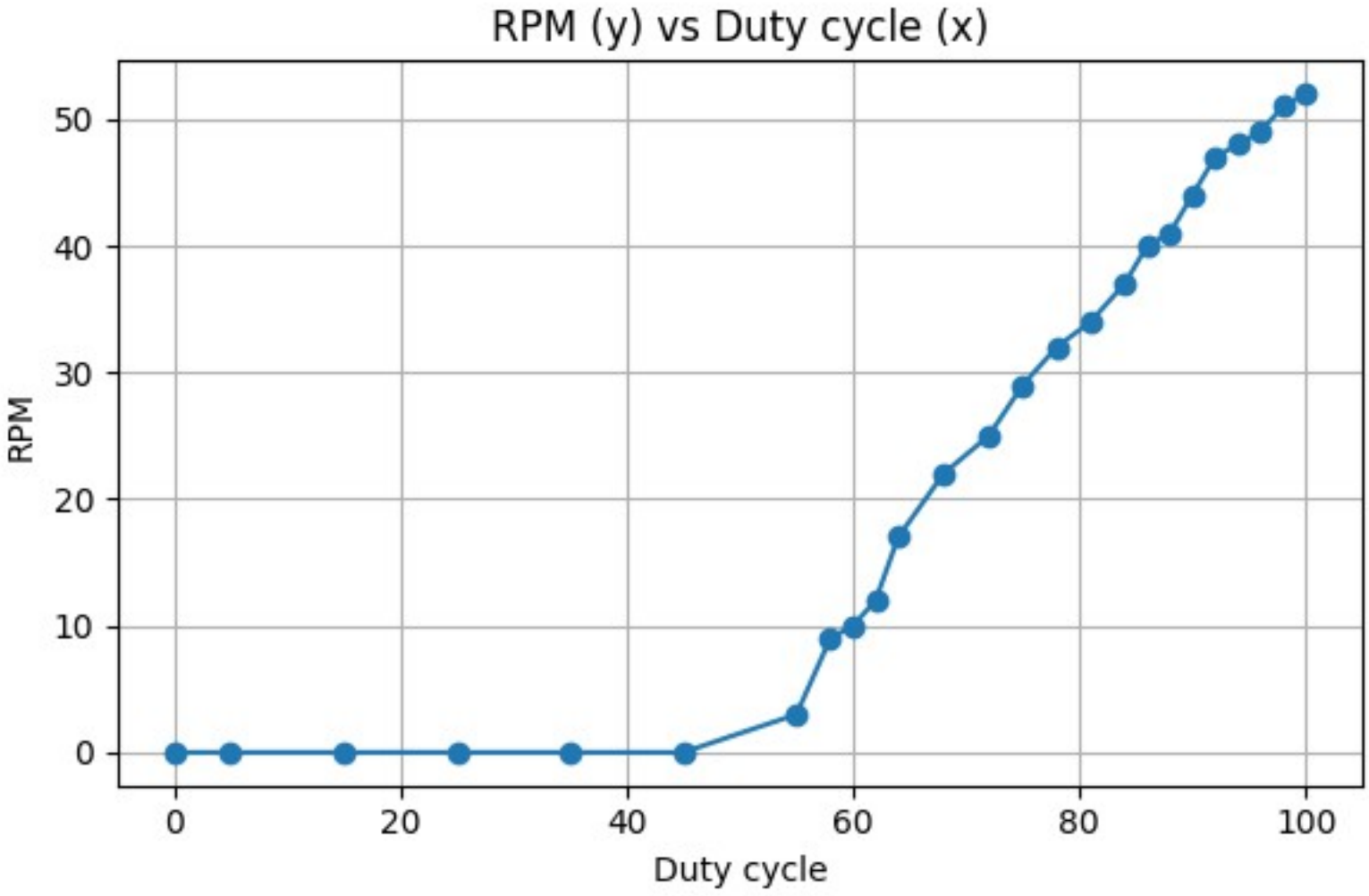
Design calculation's part 2

Maximum current drawn by the motor is the current at stall, which is 5/12.45 = 0.401 Ampere

H bridge has a maximum current output rating of 1 A so it will be able to drive the motor

Table of which bits and which ports is located in the Kicad

Description of measurement technique and time constant values attached as a separate document



Source code to the programs in part 1

```
/*******************************************************************************
 Module
   TimeConstantService.c

 Revision
   1.0.2

 Description
    This service generates a square wave on RA2 (pin 10) to help measure
   the motor's electrical time constant.

 Notes
   - Controls:
        't' - Start toggling
        's' - Stop toggling
 History
 When           Who     What/Why
 -------------- ---     --------
 01/28/2026     karthi24    Started coding service to find time constant
 01/16/12 09:58 jec      began conversion from TemplateFSM.c
*******************************************************************************/
/*---------------------------- Include Files ----------------------------*/
/* include header files for this state machine as well as any machines at the
   next lower level in the hierarchy that are sub-machines to this machine
*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "TimeConstantService.h"
#include "dbprintf.h"

/*---------------------------- Module Defines ----------------------------*/
// Pin definitions for RA2 (pin 10)
#define SQUARE_WAVE_TRIS    TRISAbits.TRISA2
#define SQUARE_WAVE_LAT     LATAbits.LATA2

// Timer settings
#define INITIAL_HALF_PERIOD_MS  500       // Start with 100ms half-period (5 Hz square wave)


/*---------------------------- Module Functions ----------------------------*/
/* prototypes for private functions for this service.They should be functions
   relevant to the behavior of this service
*/

/*---------------------------- Module Variables ----------------------------*/
// with the introduction of Gen2, we need a module level Priority variable
static uint8_t MyPriority;
// States for the state machine
typedef enum {
    IDLE,
    TOGGLING
} ServiceState_t;

static ServiceState_t CurrentState;
static uint8_t PinState;
static uint16_t HalfPeriodMS;

/*---------------------------- Module Code ----------------------------*/
/*******************************************************************************
```

```
 Function
     InitTimeConstantService

 Parameters
     uint8_t : the priorty of this service

 Returns
     bool, false if error in initialization, true otherwise

 Description
     Saves away the priority, and does any
     other required initialization for this service
 Notes

 Author
     J. Edward Carryer, 01/16/12, 10:00
****************************************************************************/
bool InitTimeConstantService(uint8_t Priority)
{
  ES_Event_t ThisEvent;

  MyPriority = Priority;
  /********************************************
   in here you write your initialization code
   ********************************************/
    CurrentState = IDLE;
    PinState = 0;
    HalfPeriodMS = INITIAL_HALF_PERIOD_MS;

  // Configure RA2 as digital output
    SQUARE_WAVE_TRIS = 0;   // Set as output
    SQUARE_WAVE_LAT = 0;    // Initialize low

    DB_printf("\r\n=== Time Constant Measurement Service ===\r\n");
    DB_printf("Commands:\r\n");
    DB_printf("  't' - Start square wave\r\n");
    DB_printf("  's' - Stop square wave\r\n");
    DB_printf("Current half-period: %d ms (%.1f Hz)\r\n",
              HalfPeriodMS, 1000.0 / (2.0 * HalfPeriodMS));
    DB_printf("========================================\r\n\n");

  // post the initial transition event
  ThisEvent.EventType = ES_INIT;
  if (ES_PostToService(MyPriority, ThisEvent) == true)
  {
    return true;
  }
  else
  {
    return false;
  }
}

/****************************************************************************
 Function
     PostTimeConstantService

 Parameters
     EF_Event_t ThisEvent ,the event to post to the queue
```

```
 Returns
     bool false if the Enqueue operation failed, true otherwise

 Description
     Posts an event to this state machine's queue
 Notes

 Author
     J. Edward Carryer, 10/23/11, 19:25
****************************************************************************/
bool PostTimeConstantService(ES_Event_t ThisEvent)
{
  return ES_PostToService(MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunTimeConstantService

 Parameters
   ES_Event_t : the event to process

 Returns
   ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise

 Description
   Implements the state machine for square wave generation
 Notes

 Author
   J. Edward Carryer, 01/15/12, 15:23
****************************************************************************/
ES_Event_t RunTimeConstantService(ES_Event_t ThisEvent)
{
  ES_Event_t ReturnEvent;
  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors
  /********************************************
   in here you write your service code
   ********************************************/
  switch (CurrentState){
        case IDLE:
              if (ThisEvent.EventType == ES_NEW_KEY)
              {
                  char key = (char)ThisEvent.EventParam;

                  if (key == 't' || key == 'T')
                  {
                      // Start toggling
                      DB_printf("Starting square wave...\r\n");

                      // Set pin high and start timer
                      PinState = 1;
                      SQUARE_WAVE_LAT = 1;
                      ES_Timer_InitTimer(TIME_CONST_TIMER, HalfPeriodMS);

                      CurrentState = TOGGLING;
                  }
              }
```

```
            break;

        case TOGGLING:
            if (ThisEvent.EventType == ES_TIMEOUT &&
                ThisEvent.EventParam == TIME_CONST_TIMER)
            {
                // Toggle the pin
                if (PinState == 1)
                {
                    PinState = 0;
                    SQUARE_WAVE_LAT = 0;
                }
                else
                {
                    PinState = 1;
                    SQUARE_WAVE_LAT = 1;
                }

                // Restart timer for next toggle
                ES_Timer_InitTimer(TIME_CONST_TIMER, HalfPeriodMS);
            }
            else if (ThisEvent.EventType == ES_NEW_KEY)
            {
                char key = (char)ThisEvent.EventParam;

                if (key == 's' || key == 'S')
                {
                    // Stop toggling
                    DB_printf("Stopping square wave.\r\n");

                    // Set pin low and stop timer
                    SQUARE_WAVE_LAT = 0;
                    PinState = 0;
                    ES_Timer_StopTimer(TIME_CONST_TIMER);

                    CurrentState = IDLE;
                }
            }
            break;
    }
    return ReturnEvent;
}

/***************************************************************************
 private functions
 ***************************************************************************/

/*------------------------------ Footnotes -------------------------------*/
/*------------------------------ End of file -------------------------------*/
```

```
/*******************************************************************************
 Module
   TemplateService.c

 Revision
   1.0.2

 Description
   This service implements PWM motor control with encoder feedback for Lab 7
   Part 1.

 Notes
 *
 * Drive-Brake Mode:
   - Forward: IN 1A = 0, IN 2A = PWM
   - Reverse: IN 1A = 1, IN 2A = inverted PWM (use 100% - duty cycle)

 History
 When           Who     What/Why
 -------------- ---     --------
 01/28/2026     karthi24    started coding up part 1 of lab 7
 01/16/12 09:58 jec      began conversion from TemplateFSM.c
*******************************************************************************/
/*---------------------------- Include Files ----------------------------*/
/* include header files for this state machine as well as any machines at the
   next lower level in the hierarchy that are sub-machines to this machine
*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "MotorService.h"
#include "PIC32_AD_Lib.h"
#include "dbprintf.h"
#include <xc.h>
#include <sys/attribs.h>
/*---------------------------- Module Defines ----------------------------*/
#define PWM_PRESCALE        0b010       // 1:4 prescaler
#define PWM_PERIOD          1249        // For 4000 Hz PWM

#define IC_TIMER_PRESCALE   0b011       // 1:8 prescaler

#define RPM_NUMERATOR       49656

#define ADC_UPDATE_TIME_MS      100     // 10 Hz ADC reading
#define PRINT_UPDATE_TIME_MS    500     // 2 Hz terminal printing

// Direction Control Pin to the motor driver (RB2, pin 6) - IN 1A
#define DIR_1A_TRIS         TRISBbits.TRISB2
#define DIR_1A_ANSEL        ANSELBbits.ANSB2
#define DIR_1A_LAT          LATBbits.LATB2

// Direction Input Pin changed by user (RA1, pin 3)
#define DIR_INPUT_TRIS      TRISAbits.TRISA1
#define DIR_INPUT_ANSEL     ANSELAbits.ANSA1
#define DIR_INPUT_PORT      PORTAbits.RA1

// PWM Output Pin (RB13, pin 24) - IN 2A via OC4
#define PWM_PIN_TRIS        TRISBbits.TRISB13
#define PWM_PIN_ANSEL       ANSELBbits.ANSB13
/*---------------------------- Module Functions ----------------------------*/
```

```
/* prototypes for private functions for this service.They should be functions
   relevant to the behavior of this service
*/
static void InitPWM(void);
static void InitDirectionPins(void);
static void InitInputCapture(void);
static void SetDutyCycle(uint8_t dutyPercent);

/*---------------------------- Module Variables ----------------------------*/
// with the introduction of Gen2, we need a module level Priority variable
static uint8_t MyPriority;

// Shared with IC ISR (volatile)
static volatile uint16_t CurrentPeriod = 0;
static volatile uint16_t LastCapture = 0;

static uint8_t CurrentDutyCyclePercent = 0;


/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitTemplateService

 Parameters
     uint8_t : the priorty of this service

 Returns
     bool, false if error in initialization, true otherwise

 Description
     Saves away the priority, and does any
     other required initialization for this service
 Notes

 Author
     J. Edward Carryer, 01/16/12, 10:00
****************************************************************************/
bool InitMotorService(uint8_t Priority)
{
  ES_Event_t ThisEvent;

  MyPriority = Priority;
  /********************************************
   in here you write your initialization code
   ********************************************/
  InitDirectionPins();

  ADC_ConfigAutoScan(BIT0HI);

  InitPWM();

  InitInputCapture();

  // Print startup message
    DB_printf("\r\n=== Motor Control Service (For part 1) ===\r\n");
    DB_printf("RA1 input controls direction\r\n");
    DB_printf("====================================\r\n");
    DB_printf("\r\nDuty%%, RPM\r\n");  // CSV header for data collection
```

```
    ES_Timer_InitTimer(MOTOR_ADC_TIMER, ADC_UPDATE_TIME_MS);

    // Start periodic timer for terminal printing (2 Hz)
    ES_Timer_InitTimer(MOTOR_PRINT_TIMER, PRINT_UPDATE_TIME_MS);


  // post the initial transition event
  ThisEvent.EventType = ES_INIT;
  if (ES_PostToService(MyPriority, ThisEvent) == true)
  {
    return true;
  }
  else
  {
    return false;
  }
}

/****************************************************************************
 Function
     PostMotorService

 Parameters
     EF_Event_t ThisEvent ,the event to post to the queue

 Returns
     bool false if the Enqueue operation failed, true otherwise

 Description
     Posts an event to this state machine's queue
 Notes

 Author
     J. Edward Carryer, 10/23/11, 19:25
****************************************************************************/
bool PostMotorService(ES_Event_t ThisEvent)
{
  return ES_PostToService(MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunTemplateService

 Parameters
   ES_Event_t : the event to process

 Returns
   ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise

 Description
   Handles ADC reading, direction control, and RPM reporting
 Notes

 Author
   J. Edward Carryer, 01/15/12, 15:23
****************************************************************************/
ES_Event_t RunMotorService(ES_Event_t ThisEvent)
```

```
{
  ES_Event_t ReturnEvent;
  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors
  /*****************************************
   in here you write your service code
   *****************************************/
  switch(ThisEvent.EventType){
      case ES_TIMEOUT:
            if (ThisEvent.EventParam == MOTOR_ADC_TIMER)
            {
                // --- Read Potentiometer ---
                uint32_t adcResult[1];
                ADC_MultiRead(adcResult);

                // Calculate duty cycle percentage (0-100)
                CurrentDutyCyclePercent = (adcResult[0] * 100) / 1023;

                // --- Read Direction Input ---
                uint8_t directionInput = DIR_INPUT_PORT;

                // --- Apply Direction and Duty Cycle ---
                if (directionInput == 0)    // Forward
                {
                    DIR_1A_LAT = 0;
                    SetDutyCycle(CurrentDutyCyclePercent);
                }
                else                        // Reverse
                {
                    DIR_1A_LAT = 1;
                    // Invert duty cycle for Drive-Brake with IN1A = 1
                    SetDutyCycle(100 - CurrentDutyCyclePercent);
                }

                // Restart ADC timer
                ES_Timer_InitTimer(MOTOR_ADC_TIMER, ADC_UPDATE_TIME_MS);
            }
            else if (ThisEvent.EventParam == MOTOR_PRINT_TIMER)
            {
                // --- Calculate and Print RPM ---
                // Safely read volatile period
                __builtin_disable_interrupts();
                uint16_t periodSnapshot = CurrentPeriod;
                __builtin_enable_interrupts();

                if (periodSnapshot > 0)
                {
                    uint32_t rpm = RPM_NUMERATOR / periodSnapshot;

                    // Print in CSV format for easy data collection
                    DB_printf("%d, %d\r\n", CurrentDutyCyclePercent, rpm);
                }
                else
                {
                    // Motor stopped or no encoder pulses
                    DB_printf("%d, 0\r\n", CurrentDutyCyclePercent);
                }

                // Restart print timer
                ES_Timer_InitTimer(MOTOR_PRINT_TIMER, PRINT_UPDATE_TIME_MS);
```

```
            }
            break;
  }
  return ReturnEvent;
}

/****************************************************************************
 ISR: Input Capture 2
 ****************************************************************************/
void __ISR(_INPUT_CAPTURE_2_VECTOR, IPL6AUTO) IC2_ISR(void)
{
    uint16_t ThisCapture;

    // Drain the FIFO - read all buffered captures, keep only the last one
    while (IC2CONbits.ICBNE)
    {
        ThisCapture = IC2BUF;
    }

    // Calculate period (unsigned subtraction handles 16-bit rollover)
    CurrentPeriod = ThisCapture - LastCapture;

    // Save for next edge
    LastCapture = ThisCapture;

    // Clear interrupt flag
    IFS0CLR = _IFS0_IC2IF_MASK;
}

/****************************************************************************
 private functions
 ****************************************************************************/
/****************************************************************************
 Function
     InitPWM

 Description
     Initializes Timer2 and OC4 for PWM output on RB13 at 4000 Hz
****************************************************************************/
static void InitPWM(void)
{
    // ===== Timer2 Setup =====
    T2CONbits.ON = 0;           // Disable Timer2
    T2CONbits.TCS = 0;          // Select internal PBCLK
    T2CONbits.TCKPS = PWM_PRESCALE;  // 1:4 prescaler
    TMR2 = 0;                   // Clear timer register
    PR2 = PWM_PERIOD;           // Set period for 4000 Hz
    // No need to clear interrupt flags since we are not using them
    T2CONbits.ON = 1;           // Enable Timer2

    // ===== OC4 Setup =====
    OC4CONbits.ON = 0;          // Disable OC4 during setup

    // Set initial duty cycle to 0%
    OC4R = 0;
    OC4RS = 0;

    // Disable analog on PWM pin
    PWM_PIN_ANSEL = 0;
```

```
    // Map OC4 to RB13 (pin 24)
    RPB13R = 0b0101;

    // Configure OC4 for PWM mode
    OC4CONbits.OCTSEL = 0;      // Use Timer2 as clock source
    OC4CONbits.OCM = 0b110;     // PWM mode, fault pin disabled

    // Enable OC4
    OC4CONbits.ON = 1;
}

/*****************************************************************************
 Function
     InitDirectionPins

 Description
     Initializes the direction control output (RB2) and direction input (RA1)
*****************************************************************************/
static void InitDirectionPins(void)
{
    // --- Direction Control Output (RB2, IN 1A) ---
    DIR_1A_ANSEL = 0;       // Disable analog
    DIR_1A_TRIS = 0;        // Set as output
    DIR_1A_LAT = 0;         // Initialize low (forward direction)

    // --- Direction Input (RA1) ---
    DIR_INPUT_ANSEL = 0;    // Disable analog
    DIR_INPUT_TRIS = 1;     // Set as input
}

/*****************************************************************************
 Function
     InitInputCapture

 Description
     Initializes Timer3 (maximum period) and IC2 for encoder period measurement
*****************************************************************************/
static void InitInputCapture(void)
{
    // ===== Timer3 Setup (free-running for Input Capture) =====
    T3CONbits.ON = 0;           // Disable Timer3
    T3CONbits.TCS = 0;          // Select internal PBCLK
    T3CONbits.TCKPS = IC_TIMER_PRESCALE;  // 1:8 prescaler
    TMR3 = 0;                   // Clear timer register
    PR3 = 0xFFFF;               // Max period
    IFS0CLR = _IFS0_T3IF_MASK;  // Clear Timer3 interrupt flag
    T3CONbits.ON = 1;           // Enable Timer3

    // ===== Input Capture 2 Setup =====
    // Map IC2 input to RB9 (pin 18)
    IC2R = 0b0100;

    // Disable IC2 during setup
    IC2CONbits.ON = 0;

    // Configure IC2
    IC2CONbits.ICTMR = 0;       // Use Timer3 (ICTMR=0 means Timer3)
    IC2CONbits.ICI = 0b00;      // Interrupt on every capture event
```

```
    IC2CONbits.ICM = 0b011;     // Capture on every rising edge

    // Configure IC2 interrupt
    IFS0CLR = _IFS0_IC2IF_MASK; // Clear interrupt flag
    IPC2bits.IC2IP = 6;         // Priority 6
    IPC2bits.IC2IS = 0;         // Subpriority 0
    IEC0SET = _IEC0_IC2IE_MASK; // Enable IC2 interrupt

    // Enable Input Capture module
    IC2CONbits.ON = 1;
}

/*****************************************************************************
 Function
     SetDutyCycle

 Parameters
     uint8_t dutyPercent - duty cycle as percentage (0-100)

 Description
     Converts percentage to PWM register value and updates OC4RS
*****************************************************************************/
static void SetDutyCycle(uint8_t dutyPercent)
{
    // Clamp to valid range
    if (dutyPercent > 100)
    {
        dutyPercent = 100;
    }

    // Convert percentage to PWM register value
    // dutyValue = (dutyPercent * PWM_PERIOD) / 100
    uint32_t dutyValue = ((uint32_t)dutyPercent * PWM_PERIOD) / 100;

    // Write to OC4RS (double-buffered, updates on next period match)
    OC4RS = dutyValue;
}

/*------------------------------ Footnotes -------------------------------*/
/*------------------------------ End of file -------------------------------*/
```

Part 2: Implementing Closed Loop Speed Control

- ☐ 2.1) Using your code from Part 1 as a base, add a periodic interrupt response routine set to fire at 2 ms intervals. In the interrupt response routine, implement a minimal proportional plus integral control law with anti-windup to calculate and program a new duty cycle every 2 ms. Be sure that you structure your code so that it does not cause you to lose an input capture interrupt (hint: make the control law interrupt a lower priority than the encoder interrupt). For calculating your control law, it is OK to use floats.
- ☐ 2.2) Tune the gains on your system to provide a quick and stable startup and maintain speed in the presence of an external disturbance (dragging a finger on the wheel attached to the motor). See Appendix B for suggestions on how to gather data to implement Ziegler-Nichols tuning.
- ☐ 2.3) Add code to your program that will take the A/D converter input as a speed command.
- ☐ 2.4) Explore the ability of the system to maintain a constant speed in the face of disturbances across the speed range.
- ☐ 2.5) Add a few instructions to your code from Part 2.4 to raise an I/O line immediately upon entry to the ISR and lower it immediately before exiting. How long did it spend in the ISR? What fraction of the execution time does that represent?
- ☐ 2.6) Demonstrate the working system to a TA/Coach/Ed. They will want to see the ability to vary the commanded speed and to maintain that speed in the presence of a disturbance.

In the report:

Include a Highlighted version of the source code to the program that you wrote for Part 2.1 and the measurements from 2.5. Be sure to be signed-off by a coach or TA. This quarter, we will be using Gradescope for your code submissions as well. Look for a Gradescope assignment entitled Lab 7 Code and submit the raw .C files (not Highlighted). Note, you will complete 1 code submission (including the files for all parts) when you submit, so save this step to the last. Please make sure that your files are named such that we can tell which files go with which parts of the lab.

Source code for part 2

```
/*******************************************************************************
 Module
   TemplateService.c

 Revision
   1.0.2

 Description
   This service implements closed-loop PI speed control of a DC motor.
   The potentiometer sets the commanded RPM, and a PI controller running
   at 2ms intervals adjusts the duty cycle to maintain the target speed.

 Notes
 No more direction control via input pin.

 History
 When           Who     What/Why
 -------------- ---     --------
 01/29/2026     karthi24    started coding for part 2
 01/28/2026     karthi24    started coding up part 1 of lab 7
 01/16/12 09:58 jec      began conversion from TemplateFSM.c
*******************************************************************************/
/*----------------------------- Include Files -----------------------------*/
/* include header files for this state machine as well as any machines at the
   next lower level in the hierarchy that are sub-machines to this machine
*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "MotorService.h"
#include "PIC32_AD_Lib.h"
#include "dbprintf.h"
#include <xc.h>
#include <sys/attribs.h>
/*----------------------------- Module Defines ----------------------------*/
#define PWM_PRESCALE        0b010       // 1:4 prescaler
#define PWM_PERIOD          1249        // For 4000 Hz PWM

#define IC_TIMER_PRESCALE   0b011       // 1:8 prescaler

// Control Loop Timer Configuration
// Timer4 with 1:4 prescaler -> 5 MHz tick rate
#define CONTROL_TIMER_PRESCALE  0b010   // 1:4 prescaler
#define CONTROL_TIMER_PERIOD    9999    // For 2ms control loop

#define MAX_COMMANDED_RPM   50 // Used in antiwindup


#define PER2RPM       49656

#define ADC_UPDATE_TIME_MS      100     // 10 Hz ADC reading
#define PRINT_UPDATE_TIME_MS    700     // 1.42 Hz terminal printing

// (RB2, pin 6) - IN 1A - previously used for direction control. only set once for
unidirectional PI control
#define DIR_1A_TRIS         TRISBbits.TRISB2
#define DIR_1A_ANSEL        ANSELBbits.ANSB2
#define DIR_1A_LAT          LATBbits.LATB2
```

```
// PWM Output Pin (RB13, pin 24) - IN 2A via OC4
#define PWM_PIN_TRIS        TRISBbits.TRISB13
#define PWM_PIN_ANSEL       ANSELBbits.ANSB13

// Timing Pin (RA2, pin 10) - for ISR timing measurement
#define TIMING_PIN_TRIS     TRISAbits.TRISA2
#define TIMING_PIN_LAT      LATAbits.LATA2
/*---------------------------- Module Functions ----------------------------*/
/* prototypes for private functions for this service.They should be functions
   relevant to the behavior of this service
*/
static void InitPWM(void);
static void InitDirectionPins(void);
static void InitInputCapture(void);
static void InitControlTimer(void);
static void SetDuty(float dutyPercent);
/*---------------------------- Module Variables ----------------------------*/
// with the introduction of Gen2, we need a module level Priority variable
static uint8_t MyPriority;

// Shared with IC ISR (volatile)
static volatile uint16_t CurrentPeriod = 0;
static volatile uint16_t LastCapture = 0;

// PI Controller Gains (adjustable via keyboard)
static float pGain = 0.5;               // Proportional gain - start conservative
static float iGain = 0.01;              // Integral gain - start small

// PI Controller State (used by Control ISR)
static volatile float SumError = 0.0;           // Integral accumulator
static volatile float TargetRPM = 0.0;          // Commanded speed from ADC

// For display (updated by Control ISR)
static volatile float CurrentRPM = 0.0;
static volatile float LastRPMError = 0.0;
static volatile float RequestedDuty = 0.0;


/*---------------------------- Module Code ----------------------------*/
/****************************************************************************
 Function
     InitTemplateService

 Parameters
     uint8_t : the priorty of this service

 Returns
     bool, false if error in initialization, true otherwise

 Description
     Saves away the priority, and does any
     other required initialization for this service
 Notes

 Author
     J. Edward Carryer, 01/16/12, 10:00
****************************************************************************/
bool InitMotorService(uint8_t Priority)
```

```
{
  ES_Event_t ThisEvent;

  MyPriority = Priority;
  /********************************************
   in here you write your initialization code
   ********************************************/

  // Initialize the Timing pin
  // RA2 does not have an ANSEL register (not analog-capable)
    TIMING_PIN_TRIS = 0;            // Set as output
    TIMING_PIN_LAT = 0;             // Initialize low


  InitDirectionPins();

  ADC_ConfigAutoScan(BIT0HI);

  InitPWM();

  InitInputCapture();

    // Initialize control loop timer (Timer4, 2ms)
  InitControlTimer();

  // Print startup message
    DB_printf("\r\n=== Motor Control Service (PI controller with anti-windup)===\r\n");
    DB_printf("Potentiometer sets commanded RPM (0-%d)\r\n", MAX_COMMANDED_RPM);
    DB_printf("Initial gains: pGain = %d, iGain = %d\r\n", (int)(100 * pGain), (int)(100 *
iGain));
    DB_printf("\r\nKeyboard Controls:\r\n");
    DB_printf("  'P'/'p' - increase/decrease pGain\r\n");
    DB_printf("  'I'/'i' - increase/decrease iGain\r\n");
//    DB_printf("  'r'     - reset integral term\r\n");
    DB_printf("  '?'     - show current gains\r\n");
    DB_printf("============================================\r\n");
    DB_printf("\r\nTargetRPM, ActualRPM, Error, Duty%%\r\n");

    ES_Timer_InitTimer(MOTOR_ADC_TIMER, ADC_UPDATE_TIME_MS);

    // Start periodic timer for terminal printing
    ES_Timer_InitTimer(MOTOR_PRINT_TIMER, PRINT_UPDATE_TIME_MS);


  // post the initial transition event
  ThisEvent.EventType = ES_INIT;
  if (ES_PostToService(MyPriority, ThisEvent) == true)
  {
    return true;
  }
  else
  {
    return false;
  }
}

/****************************************************************************
 Function
     PostMotorService
```

```
 Parameters
     EF_Event_t ThisEvent ,the event to post to the queue

 Returns
     bool false if the Enqueue operation failed, true otherwise

 Description
     Posts an event to this state machine's queue
 Notes

 Author
     J. Edward Carryer, 10/23/11, 19:25
****************************************************************************/
bool PostMotorService(ES_Event_t ThisEvent)
{
  return ES_PostToService(MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunMotorService

 Parameters
   ES_Event_t : the event to process

 Returns
   ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise

 Description
   Handles ADC reading for speed command, terminal printing, and gain tuning
 Notes

 Author
   J. Edward Carryer, 01/15/12, 15:23
****************************************************************************/
ES_Event_t RunMotorService(ES_Event_t ThisEvent)
{
  ES_Event_t ReturnEvent;
  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors
  /********************************************
   in here you write your service code
   ********************************************/
  switch(ThisEvent.EventType){
      case ES_TIMEOUT:
            if (ThisEvent.EventParam == MOTOR_ADC_TIMER)
            {
                // --- Read Potentiometer for Speed Command ---
                uint32_t adcResult[1];
                ADC_MultiRead(adcResult);

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

                // Restart ADC timer
                ES_Timer_InitTimer(MOTOR_ADC_TIMER, ADC_UPDATE_TIME_MS);
            }
            else if (ThisEvent.EventParam == MOTOR_PRINT_TIMER)
            {
```

```
                    // --- Print current state for monitoring ---
                    // Read volatile variables safely
                    __builtin_disable_interrupts();
                    float displayRPM = CurrentRPM;
                    float displayError = LastRPMError;
                    float displayDuty = RequestedDuty;
                    __builtin_enable_interrupts();

                    // Print in CSV format

                    DB_printf("%d, %d, %d, %d\r\n",
                              (int)(TargetRPM), (int)(displayRPM), (int)(displayError),
(int)(displayDuty));

                    // Restart print timer
                    ES_Timer_InitTimer(MOTOR_PRINT_TIMER, PRINT_UPDATE_TIME_MS);
                }
                break;

            case ES_NEW_KEY:
            {
                char key = (char)ThisEvent.EventParam;

                switch (key)
                {
                    case 'P':   // Increase pGain
                        pGain += 0.1;
                        DB_printf("pGain = %d\r\n", (int)(100 * pGain));
                        break;

                    case 'p':   // Decrease pGain
                        pGain -= 0.1;
                        if (pGain < 0) pGain = 0;
                        DB_printf("pGain = %d\r\n", (int)(100 * pGain));
                        break;

                    case 'I':   // Increase iGain
                        iGain += 0.005;
                        DB_printf("iGain = %d\r\n", (int)(100 * iGain));
                        break;

                    case 'i':   // Decrease iGain
                        iGain -= 0.005;
                        if (iGain < 0) iGain = 0;
                        DB_printf("iGain = %d\r\n", (int)(100 * iGain));
                        break;

    //                case 'r':   // Reset integral term
    //                case 'R':
    //                    SumError = 0;
    //                    DB_printf("Integral term reset (SumError = 0)\r\n");
    //                    break;

                    case '?':   // Show current gains
                        DB_printf("Current gains: pGain = %d, iGain = %d\r\n",
                                  (int)(100 * pGain), (int)(100 * iGain));
                        DB_printf("SumError = %d\r\n", (int)(SumError));
                        DB_printf("\r\nTargetRPM, ActualRPM, Error, Duty%%\r\n");
                        break;
```

```
                default:
                    break;
            }
        }break;

  }
  return ReturnEvent;
}

/*****************************************************************************
 ISR: Control Loop (Timer4) - Priority 5
 Runs every 2ms to calculate and apply PI control law
 *****************************************************************************/
void __ISR(_TIMER_4_VECTOR, IPL5AUTO) Timer4_ISR(void)
{
    // Clear interrupt flag first
    IFS0CLR = _IFS0_T4IF_MASK;

    // Raise timing pin on entry
    TIMING_PIN_LAT = 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 = (float)PER2RPM / (float)ThisPeriod;
    }
    else
    {
        Rpm = 0;
    }

    // --- Calculate Error ---
    RPMError = TargetRPM - Rpm;
    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
    }
    else if (RequestedDuty < 0)
    {
        RequestedDuty = 0;
        SumError -= RPMError;   // Anti-windup
    }

    // --- Update values for display ---
```

```
    LastRPMError = RPMError;
    CurrentRPM = Rpm;

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

    // Lower timing pin before exit
    TIMING_PIN_LAT = 0;
}

/*****************************************************************************
 ISR: Input Capture 2
 *****************************************************************************/
void __ISR(_INPUT_CAPTURE_2_VECTOR, IPL6AUTO) IC2_ISR(void)
{
    uint16_t ThisCapture;

    // Drain the FIFO - read all buffered captures, keep only the last one
    while (IC2CONbits.ICBNE)
    {
        ThisCapture = IC2BUF;
    }

    // Calculate period (unsigned subtraction handles 16-bit rollover)
    CurrentPeriod = ThisCapture - LastCapture;

    // Save for next edge
    LastCapture = ThisCapture;

    // Clear interrupt flag
    IFS0CLR = _IFS0_IC2IF_MASK;
}

/*****************************************************************************
 private functions
 *****************************************************************************/
/*****************************************************************************
 Function
     InitPWM

 Description
     Initializes Timer2 and OC4 for PWM output on RB13 at 4000 Hz
*****************************************************************************/
static void InitPWM(void)
{
    // ===== Timer2 Setup =====
    T2CONbits.ON = 0;           // Disable Timer2
    T2CONbits.TCS = 0;          // Select internal PBCLK
    T2CONbits.TCKPS = PWM_PRESCALE;  // 1:4 prescaler
    TMR2 = 0;                   // Clear timer register
    PR2 = PWM_PERIOD;           // Set period for 4000 Hz
    // No need to clear interrupt flags since we are not using them
    T2CONbits.ON = 1;           // Enable Timer2

    // ===== OC4 Setup =====
    OC4CONbits.ON = 0;          // Disable OC4 during setup

    // Set initial duty cycle to 0%
    OC4R = 0;
```

```
    OC4RS = 0;

    // Disable analog on PWM pin
    PWM_PIN_ANSEL = 0;

    // Map OC4 to RB13 (pin 24)
    RPB13R = 0b0101;

    // Configure OC4 for PWM mode
    OC4CONbits.OCTSEL = 0;      // Use Timer2 as clock source
    OC4CONbits.OCM = 0b110;     // PWM mode, fault pin disabled

    // Enable OC4
    OC4CONbits.ON = 1;
}

/*****************************************************************************
 Function
     InitDirectionPins

 Description
     Initializes the direction control output (RB2) and direction input (RA1)
*****************************************************************************/
static void InitDirectionPins(void)
{
    // --- Direction Control Output (RB2, IN 1A) ---
    DIR_1A_ANSEL = 0;       // Disable analog
    DIR_1A_TRIS = 0;        // Set as output
    DIR_1A_LAT = 0;         // Initialize low (forward direction)


}

/*****************************************************************************
 Function
     InitInputCapture

 Description
     Initializes Timer3 (maximum period) and IC2 for encoder period measurement
*****************************************************************************/
static void InitInputCapture(void)
{
    // ===== Timer3 Setup (free-running for Input Capture) =====
    T3CONbits.ON = 0;           // Disable Timer3
    T3CONbits.TCS = 0;          // Select internal PBCLK
    T3CONbits.TCKPS = IC_TIMER_PRESCALE;  // 1:8 prescaler
    TMR3 = 0;                   // Clear timer register
    PR3 = 0xFFFF;               // Max period
    IFS0CLR = _IFS0_T3IF_MASK;  // Clear Timer3 interrupt flag
    T3CONbits.ON = 1;           // Enable Timer3

    // ===== Input Capture 2 Setup =====
    // Map IC2 input to RB9 (pin 18)
    IC2R = 0b0100;

    // Disable IC2 during setup
    IC2CONbits.ON = 0;

    // Configure IC2
```

```
    IC2CONbits.ICTMR = 0;       // Use Timer3 (ICTMR=0 means Timer3)
    IC2CONbits.ICI = 0b00;      // Interrupt on every capture event
    IC2CONbits.ICM = 0b011;     // Capture on every rising edge

    // Configure IC2 interrupt
    IFS0CLR = _IFS0_IC2IF_MASK; // Clear interrupt flag
    IPC2bits.IC2IP = 6;         // Priority 6
    IPC2bits.IC2IS = 0;         // Subpriority 0
    IEC0SET = _IEC0_IC2IE_MASK; // Enable IC2 interrupt

    // Enable Input Capture module
    IC2CONbits.ON = 1;
}

/*****************************************************************************
 Function
     InitControlTimer

 Description
     Initializes Timer4 for 2ms control loop interrupt
*****************************************************************************/
static void InitControlTimer(void)
{
    // Disable Timer4
    T4CONbits.ON = 0;

    // Select internal PBCLK
    T4CONbits.TCS = 0;

    // Set prescaler (1:4)
    T4CONbits.TCKPS = CONTROL_TIMER_PRESCALE;

    // Clear timer register
    TMR4 = 0;

    // Set period for 2ms
    // PBCLK = 20MHz, Prescaler = 1:4 -> 5MHz tick rate
    // 2ms = 0.002s * 5,000,000 = 10,000 ticks
    // PR4 = 10000 - 1 = 9999
    PR4 = CONTROL_TIMER_PERIOD;

    // Configure Timer4 interrupt - Priority 5 (LOWER than Input Capture)
    IFS0CLR = _IFS0_T4IF_MASK;      // Clear interrupt flag
    IPC4bits.T4IP = 5;              // Priority 5 lower than IC interrupt priority
    IPC4bits.T4IS = 0;              // Subpriority 0
    IEC0SET = _IEC0_T4IE_MASK;      // Enable Timer4 interrupt

    // Enable Timer4
    T4CONbits.ON = 1;
}
/*****************************************************************************
 Function
     SetDutyCycle

 Parameters
     uint8_t dutyPercent - duty cycle as percentage (0-100)

 Description
     Converts percentage to PWM register value and updates OC4RS
```

```
*****************************************************************************/
static void SetDuty(float dutyPercent)
{
    // Clamp to valid range
    if (dutyPercent > 100)
    {
        dutyPercent = 100;
    }
    else if (dutyPercent < 0)
    {
        dutyPercent = 0;
    }

    // Convert percentage to PWM register value
    uint32_t dutyValue = (uint32_t)((dutyPercent * PWM_PERIOD) / 100.0);

    // Write to OC4RS (double-buffered, updates on next period match)
    OC4RS = dutyValue;
}

/*------------------------------ Footnotes -------------------------------*/
/*------------------------------ End of file -------------------------------*/
```

Measurements from part 2.5

Time spent in ISR = 27.4 microseconds

0.0137 or 1.37% of the total execution time

math

Nachiketh Karthik

January 2026

Motor Electrical Characterization

The motor is modeled electrically as a series resistance and inductance,

$$R_{\text{eq}} = R_{\text{motor}} + R_{\text{sense}}, \quad L = L_{\text{motor}}.$$

Motor Resistance Measurement

A DC voltage of 5 V was applied to the motor in series with a known sense resistor R_{sense}. The voltage drop across the sense resistor was measured as V_{drop}.

The stall current is given by

$$i_{\text{stall}} = \frac{V_{\text{drop}}}{R_{\text{sense}}}.$$

The voltage across the motor resistance is

$$V_{\text{motor}} = 5 \text{ V} - V_{\text{drop}}.$$

Thus, the motor resistance is

$$R_{\text{motor}} = \frac{V_{\text{motor}}}{i_{\text{stall}}} = \frac{5 - V_{\text{drop}}}{V_{\text{drop}}/R_{\text{sense}}}.$$

Using the measured values

$$V_{\text{drop}} = 2.26 \text{ V},$$

the motor resistance was calculated to be

$$R_{\text{motor}} \approx 12.45 \ \Omega.$$

Electrical Time Constant Measurement

The motor was driven with a step voltage using PWM, and the current rise was observed on an oscilloscope. For a first-order $R-L$ system, the current reaches 63.2% of its steady-state value at one time constant:

$$i(\tau) = 0.632\, i_{\text{final}}.$$

From the oscilloscope trace, the time constant was measured as

$$\tau \approx 225\ \mu\text{s}.$$

The electrical time constant of the motor is given by

$$\tau = \frac{L_{\text{motor}}}{R_{\text{motor}} + R_{\text{sense}}}.$$

Solving for the motor inductance,

$$L_{\text{motor}} = \tau \left(R_{\text{motor}} + R_{\text{sense}} \right).$$

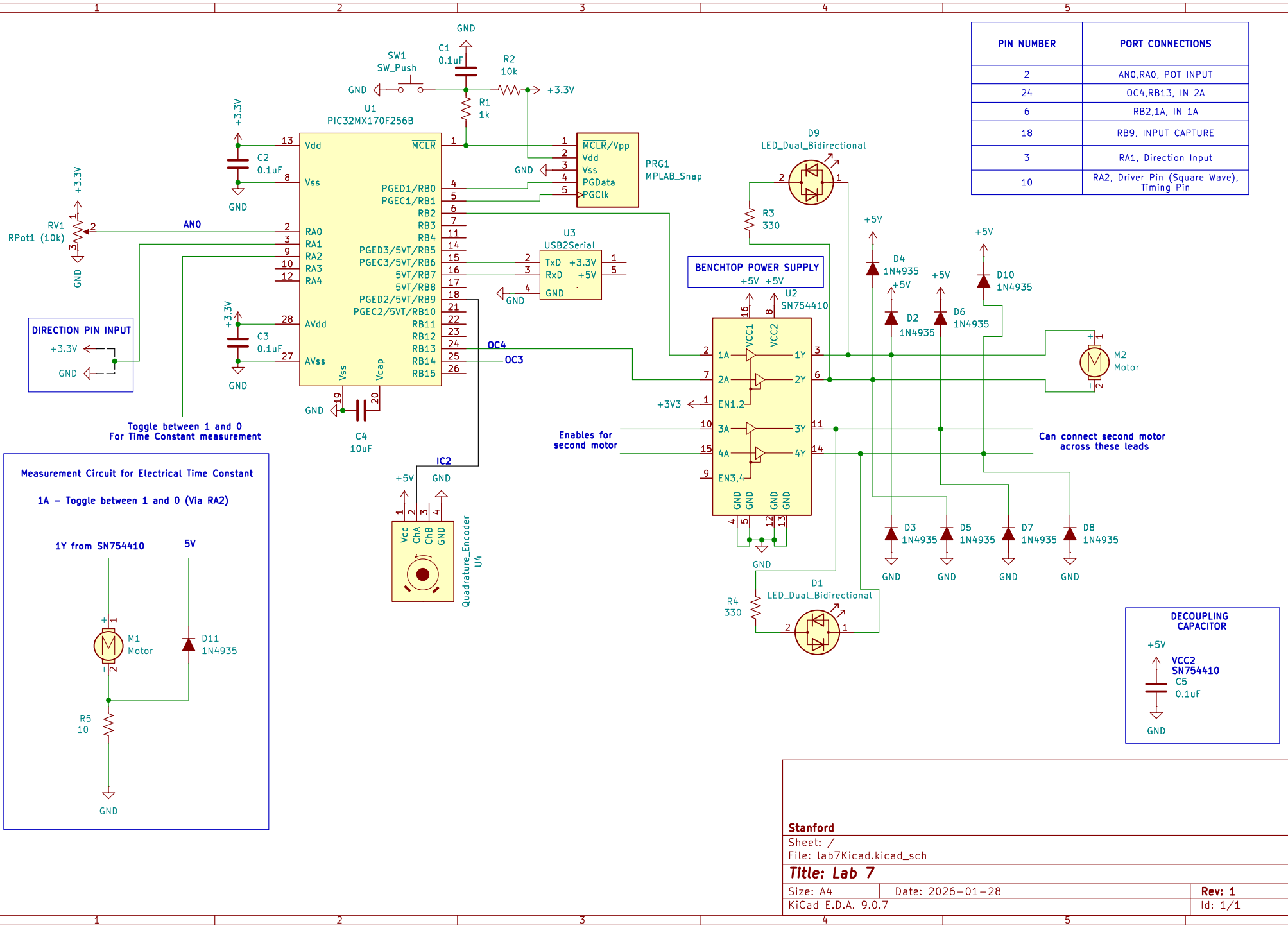
Using the measured values, the motor inductance was estimated to be

$$L_{\text{motor}} \approx 5.05\ \text{mH}.$$

Motor-Only Electrical Time Constant

After estimating the motor inductance, the motor-only electrical time constant is

$$\tau_{\text{motor}} = \frac{L_{\text{motor}}}{R_{\text{motor}}} = \frac{5.05 \times 10^{-3}}{12.45}\ \text{s} = 4.06 \times 10^{-4}\ \text{s} \approx 0.406\ \text{ms} \approx 406\ \mu\text{s}.$$


PIN NUMBER
PORT CONNECTIONS
2
AN0,RA0, POT INPUT
24
OC4,RB13, IN 2A
6
RB2,1A, IN 1A
18
RB9, INPUT CAPTURE
3
RA1, Direction Input
10
RA2, Driver Pin (Square Wave), Timing Pin
U1
PIC32MX170F256B
SW1
SW_Push
C1
0.1uF
R1
1k
R2
10k
C2
0.1uF
C3
0.1uF
C4
10uF
RV1
RPot1 (10k)
AN0
DIRECTION PIN INPUT
Toggle between 1 and 0
For Time Constant measurement
PRG1
MPLAB_Snap
U3
USB2Serial
OC4
OC3
IC2
U4
Quadrature_Encoder
BENCHTOP POWER SUPPLY
+5V +5V
U2
SN754410
+3V3
Enables for
second motor
Can connect second motor
across these leads
M2
Motor
D9
LED_Dual_Bidirectional
R3
330
D1
LED_Dual_Bidirectional
R4
330
D2
1N4935
D3
1N4935
D4
1N4935
D5
1N4935
D6
1N4935
D7
1N4935
D8
1N4935
D10
1N4935
Measurement Circuit for Electrical Time Constant
1A – Toggle between 1 and 0 (Via RA2)
1Y from SN754410
5V
M1
Motor
D11
1N4935
R5
10
DECOUPLING
CAPACITOR
VCC2
SN754410
C5
0.1uF
Stanford
Sheet: /
File: lab7Kicad.kicad_sch
Title: Lab 7
Size: A4
Date: 2026-01-28
Rev: 1
KiCad E.D.A. 9.0.7
Id: 1/1