Laboratory Assignment Number 6 for ME 218b

Pre-Lab Due by 23:59 on January 20, 2026
Due by 17:00 on January 22, 2026

Part 0: Pre-Lab

Assignment:

❑ 0.1) Design the circuitry necessary to complete parts 1 & 2 of the lab assignment. Your circuitry should bi-directionally drive the DC motor provided at the lab bench (see Lab 6 Appendix A) using an SN754410 as the drive stage with a motor voltage of 5V, provide an analog voltage (0-3.3V) from the potentiometer in Part 1.1 to an analog input on the PIC32, provide for a digital input capable of input capture to read the encoder and include the drive circuitry for the bar-graph LED in Part 2. The 5V to drive the motor should come from the bench-top power supply, NOT from the breadboard power supply! Include this schematic in your report. The internal kick-back diodes in the SN754410 have proven to be problematic in prior years, so be sure to include external diodes in your design. Please confirm that the H-bridge specified can drive the motor (quote specifications/measurements to back that up)! You should make a measurement on the motor to confirm this.

❑ 0.2) Determine which pins on the PIC32 you need to hook up for the complete schematic from part 0.1. Prepare a table of the pin numbers and port connections that correspond to the pins identified in Part 0.1.

❑ 0.3) Design the event driven software that you will use for Part 1. Include pseudo code for program. Do not worry about getting the specifics of the PWM module ready for the pre-lab. We will cover the PWM subsystem in lecture on Thursday.

❑ 0.4) Design the software that you will use for Part 2. Include pseudo code for the program.

In the report:

Include the schematic diagram of the circuit you used for 0.1, the table of pin assignments from 0.2, pseudo-code from 0.3 & 0.4 and design calculations. Be sure to indicate on the schematic which bits of which ports of the PIC32 you used, and please include pin numbers.

Prelab attached at the beginning of the lab document

Part 1: Driving a DC Motor Using PWM

❑ 1.1) Implement your DC motor driver and potentiometer design from Part 0.1 and an Event Driven software design to read the A/D converter connected to the potentiometer (powered from the 3.3V supply on the breadboard power supply as in Lab 5) and adjust the speed from Part 0.3. The code to read the A/D should be implemented in a service within the framework. Hook it up to the motor and get the motor to run with the speed controlled by the potentiometer.

❑ 1.2) Demonstrate your working software to a coach/TA/Ed and get signed off on the check-out sheet.

In the report:

Include a Highlighted version of the source code to the program. 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 6 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 [only motorservice.c]

```
/****************************************************************************
 Module
   MotorService.c
 Revision
   1.0.0

 Description
   This service implements PWM motor control using OC4 and Timer2.
   Motor speed is controlled by a potentiometer read via ADC at 10Hz.

 Notes
   - PWM output on RB13 (pin 24) via OC4
   - Potentiometer input on AN0/RA0 (pin 2)
   - Direction control on RB2 (1A) and RB3 (2A)
   - PWM frequency approximately 200Hz for part 1

 History
 When           Who     What/Why
 -------------- ---     --------
 01/21/2026     karthi24    started conversion from template file
 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 ----------------------------*/
// PWM Prescaler (1:4 for all frequencies)
#define PWM_PRESCALE        0b010       // 1:4 prescaler
// PBCLK = 20 MHz, Prescaler = 1:4 -> Effective clock = 5 MHz
// PR2 = (5,000,000 / freq) - 1
#define NUM_FREQUENCIES     6

// PWM Period values for each frequency (PR2 values)
// Index:  0       1       2       3       4       5
// Freq:   250Hz   500Hz   1kHz    2kHz    5kHz    10kHz
static const uint16_t PWM_PERIODS[NUM_FREQUENCIES] = {
    19999,  // 250 Hz
    9999,   // 500 Hz
    4999,   // 1 kHz
    2499,   // 2 kHz
    999,    // 5 kHz
    499     // 10 kHz
};

// Frequency labels for printing
static const char* FREQ_LABELS[NUM_FREQUENCIES] = {
    "250 Hz",
    "500 Hz",
    "1 kHz",
    "2 kHz",
    "5 kHz",
    "10 kHz"
};
```

```
// ADC update rate 10 Hz
#define ADC_UPDATE_TIME     100



// Direction pin definitions
#define DIR_1A_TRIS         TRISBbits.TRISB2
#define DIR_1A_ANSEL        ANSELBbits.ANSB2
#define DIR_1A              LATBbits.LATB2

#define DIR_2A_TRIS         TRISBbits.TRISB3
#define DIR_2A_ANSEL        ANSELBbits.ANSB3
#define DIR_2A              LATBbits.LATB3

//pwm pin definition
#define PWM_PIN_TRIS        TRISBbits.TRISB13
#define PWM_PIN_ANSEL       ANSELBbits.ANSB13

// Timer3 prescaler for input capture (1:8)
#define IC_TIMER_PRESCALE   0b011


// Timer3 tick = 0.4 us with 1:8 prescaler at 20MHz PBCLK
#define MIN_PERIOD          900     // Fast speed (tunable parameter)
#define MAX_PERIOD          2000    // Slow speed (tunable parameter)
#define PERIOD_RANGE        (MAX_PERIOD - MIN_PERIOD)

// RPM calculation constant (refer to tablet for proper calculations)
// Output wheel: 512 pulses/rev * 5.9 gearbox = 3020.8 pulses/rev
// Timer tick = 0.4 us, so frequency = 2,500,000 ticks/sec
// RPM = (2,500,000 / Period) / 3020.8 * 60 = 49,656 / Period
#define RPM_NUMERATOR       49656

// Timing pin (RB5) for scope measurement
#define TIMING_PIN_TRIS     TRISBbits.TRISB5
#define TIMING_PIN          LATBbits.LATB5

// Bar graph pin definitions
// Top to bottom: RA1, RA2, RA3, RA4, RB8, RB10, RB11, RB15
#define BAR1_TRIS           TRISAbits.TRISA1
#define BAR1_ANSEL           ANSELAbits.ANSA1
#define BAR1                LATAbits.LATA1

#define BAR2_TRIS           TRISAbits.TRISA2
// A2 doesnt have an ANSEL register
#define BAR2                LATAbits.LATA2

#define BAR3_TRIS           TRISAbits.TRISA3
// A3 doesnt have an ANSEL
#define BAR3                LATAbits.LATA3

#define BAR4_TRIS           TRISAbits.TRISA4
// A4 doesnt have an ANSEL
#define BAR4                LATAbits.LATA4

#define BAR5_TRIS           TRISBbits.TRISB8
// B8 does not have an ANSEL
#define BAR5                LATBbits.LATB8
```

```
#define BAR6_TRIS           TRISBbits.TRISB10
// B10 does not have an ANSEL
#define BAR6                LATBbits.LATB10

#define BAR7_TRIS           TRISBbits.TRISB11
//B11 does not have an ANSEL
#define BAR7                LATBbits.LATB11

#define BAR8_TRIS           TRISBbits.TRISB15
#define BAR8_ANSEL          ANSELBbits.ANSB15   // RB15 is AN9
#define BAR8                LATBbits.LATB15
/*---------------------------- 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 InitDirectionPWMPins(void);
static void SetDutyCycle(uint32_t dutyCycle);
static void InitInputCapture(void);
static void InitBarGraphTimingPins(void);
static void UpdateBarGraph(uint16_t period);
static void SetPWMFrequency(uint8_t freqIndex);


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

// Shared with ISR (these variables need to be of the volatile datatype)
static volatile uint16_t CurrentPeriod = 0;
static volatile uint16_t LastCapture = 0;

static uint8_t CurrentFreqIndex = 0;            // Start at 250 Hz
static uint16_t CurrentPWMPeriod = 19999;       // PR2 value for current frequency

static uint32_t LastADCValue = 0;



/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitMotorService

 Parameters
     uint8_t : the priorty of this service

 Returns
     bool, false if error in initialization, true otherwise

 Description
     Saves away the priority, and Initializes PWM, ADC, direction pins, and starts the
periodic timer
 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 direction control pins
    InitDirectionPWMPins();


    ADC_ConfigAutoScan(BIT0HI);

    // Initialize PWM using OC4 and Timer2
    InitPWM();

    // Initialize bar graph output pins
    InitBarGraphTimingPins();

    // Initialize input capture for encoder
    InitInputCapture();

    DB_printf("Press '1'-'6' to change PWM frequency:\r\n");
    DB_printf("  1=250Hz  2=500Hz  3=1kHz\r\n");
    DB_printf("  4=2kHz   5=5kHz   6=10kHz\r\n");
    DB_printf("Current: %s\r\n\n", FREQ_LABELS[CurrentFreqIndex]);


    // Start the periodic ADC read timer (10 Hz)
    ES_Timer_InitTimer(MOTOR_TIMER, ADC_UPDATE_TIME);

  // 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
   add your description here
 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_NEW_KEY:
            // Part 3: Handle frequency change via keyboard
            {
                char key = (char)ThisEvent.EventParam;

                // Check for keys '1' through '6'
                if (key >= '1' && key <= '6')
                {
                    uint8_t newIndex = key - '1';  // Convert '1'-'6' to 0-5
                    SetPWMFrequency(newIndex);

                    DB_printf("\r\n>>> PWM Frequency changed to: %s <<<\r\n\n",
                              FREQ_LABELS[CurrentFreqIndex]);
                }
            }
            break;



        case ES_TIMEOUT:
            if (ThisEvent.EventParam == MOTOR_TIMER)
            {
                // Read potentiometer value from ADC
                uint32_t adcResult[1];
                ADC_MultiRead(adcResult);
                LastADCValue = adcResult[0];
                // Scale ADC value (0-1023) to duty cycle (0-PWM_PERIOD)
                uint32_t newDutyCycle = (adcResult[0] * CurrentPWMPeriod) / 1023;

                // Update PWM duty cycle
                SetDutyCycle(newDutyCycle);
                // Update bar graph
                __builtin_disable_interrupts();
                uint16_t periodSnapshot = CurrentPeriod;
                 __builtin_enable_interrupts();

                // Update bar graph with the snapshot
                UpdateBarGraph(periodSnapshot);

                // Print period for debugging
//                DB_printf("Period: %d\r\n", periodSnapshot);

                // Raise timing pin before RPM calculation
//                TIMING_PIN = 1;

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

                    // Lower timing pin after calculation ---> step 2.5
//                    TIMING_PIN = 0;

                    // Raise timing pin before printf (Part 2.6)
//                    TIMING_PIN = 1;
                    DB_printf("\r[%s] RPM: %d  Period: %d  DC: %d%%",
                              FREQ_LABELS[CurrentFreqIndex],
                              wheelRPM,
                              periodSnapshot,
                              (adcResult[0] * 100) / 1023);


                    // Lower timing pin after printf (Part 2.6)
//                    TIMING_PIN = 0;
                }
                else
                {
//                    TIMING_PIN = 0;
                    DB_printf("\r[%s] RPM: ----  Period: -----  DC: %d%%",
                              FREQ_LABELS[CurrentFreqIndex],
                              (adcResult[0] * 100) / 1023);
                }
                // Restart timer for next reading

                ES_Timer_InitTimer(MOTOR_TIMER, ADC_UPDATE_TIME);
            }

            break;

        case ES_INIT:
            // Initialization complete - nothing special needed
            break;

        default:
            break;
        }
    return ReturnEvent;
}
/***************************************************************************
 ISR's
 ***************************************************************************/
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
    // ICBNE = 1 means buffer is not empty
    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 AFTER reading buffer
    IFS0CLR = _IFS0_IC2IF_MASK;

    // Post event to service to trigger bar graph update
//    DB_printf("%d\r\n",LastCapture);
//    ES_Event_t NewEvent;
//    NewEvent.EventType = ES_NEW_EDGE;
//    PostMotorService(NewEvent);

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

static void SetPWMFrequency(uint8_t freqIndex)
{
    if (freqIndex >= NUM_FREQUENCIES)
    {
        return;  // Invalid index
    }

    // Update tracking variables
    CurrentFreqIndex = freqIndex;
    CurrentPWMPeriod = PWM_PERIODS[freqIndex];

    // Briefly disable Timer2 while changing period
    T2CONbits.ON = 0;

    // Update period register
    PR2 = CurrentPWMPeriod;

    // Clear timer to avoid glitches
    TMR2 = 0;

    // Recalculate duty cycle to maintain same percentage
    uint32_t newDutyCycle = (LastADCValue * CurrentPWMPeriod) / 1023;
    OC4RS = newDutyCycle;
    OC4R = newDutyCycle;

    // Re-enable Timer2
    T2CONbits.ON = 1;
}

static void InitDirectionPWMPins(void)
{
    // Disable analog function on RB2 and RB3 and RB13 (they are AN4 and AN5)
    DIR_1A_ANSEL = 0;
    DIR_2A_ANSEL = 0;
    PWM_PIN_ANSEL = 0;

    // Set RB2 and RB3 and RB13 as outputs
    DIR_1A_TRIS = 0;
    DIR_2A_TRIS = 0;
    PWM_PIN_TRIS = 0;

    // Set initial direction
    DIR_1A = 1;
    DIR_2A = 0;
}

static void InitPWM(void)
{
    // ===== Timer2 Setup =====

    T2CONbits.ON = 0;

    T2CONbits.TCS = 0;

    // Select prescaler (1:4)
    T2CONbits.TCKPS = PWM_PRESCALE;

    TMR2 = 0;

    PR2 = CurrentPWMPeriod;

//    IFS0CLR = _IFS0_T2IF_MASK;
    //Enable Timer2
    T2CONbits.ON = 1;
    // ===== Output Compare 4 Setup =====

    OC4CONbits.ON = 0;

    // Set initial duty cycle (0% - motor stopped)
    OC4R = 0;
    OC4RS = 0;

    // Map OC4 output to RB13 (pin 24)


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

    RPB13R = 0b0101;
    // Enable OC4
    OC4CONbits.ON = 1;
}

static void InitBarGraphTimingPins(void)
{
    // RB5 is not an analog pin, no ANSEL needed
    TIMING_PIN_TRIS = 0;     // Set as output
    TIMING_PIN = 0;          // Initialize low

    // Disable analog
    BAR1_ANSEL = 0;
    BAR8_ANSEL = 0;

    // Set all bar graph pins as outputs
    BAR1_TRIS = 0;
    BAR2_TRIS = 0;
    BAR3_TRIS = 0;
    BAR4_TRIS = 0;
    BAR5_TRIS = 0;
    BAR6_TRIS = 0;
    BAR7_TRIS = 0;
    BAR8_TRIS = 0;

    // Initialize all LEDs off
    BAR1 = 0;
    BAR2 = 0;
    BAR3 = 0;
    BAR4 = 0;
    BAR5 = 0;
    BAR6 = 0;
    BAR7 = 0;
    BAR8 = 0;
}

static void SetDutyCycle(uint32_t dutyCycle)
{
    // Clamp duty cycle to valid range
    if (dutyCycle > CurrentPWMPeriod)
    {
        dutyCycle = CurrentPWMPeriod;
    }

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

static void InitInputCapture(void)
{
    // ===== Timer3 Setup (free-running for Input Capture) =====

    // Disable timer
    T3CONbits.ON = 0;

    //Select internal PBCLK source
    T3CONbits.TCS = 0;

    //  Select prescaler (1:8)
    T3CONbits.TCKPS = IC_TIMER_PRESCALE;

    // Clear timer register
    TMR3 = 0;

    //  Load period register (max for free-running)
    PR3 = 0xFFFF;

    //Clear T3IF interrupt flag
    IFS0CLR = _IFS0_T3IF_MASK;

    //  Enable Timer3
    T3CONbits.ON = 1;

    // ===== Input Capture 2 Setup =====

    // Map IC2 input to RB9
    IC2R = 0b0100;

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

    // Configure IC2
    IC2CONbits.ICTMR = 0;        // Use Timer3 as source (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;
}

static void UpdateBarGraph(uint16_t period)
{
    uint8_t numBars;

    // Raise timing pin before calculation (Part 2.3)
//    TIMING_PIN = 1;

    // Determine number of bars to light
    if (period <= MIN_PERIOD)
    {
        numBars = 8;  // Fast = more bars (or 1 if you prefer opposite)
    }
    else if (period >= MAX_PERIOD)
    {
        numBars = 1;  // Slow = fewer bars
    }
    else
    {
        // Linear interpolation: 8 - 7 * (period - MIN) / RANGE
        numBars = 8 - (7 * (period - MIN_PERIOD)) / PERIOD_RANGE;
    }
    // Update each LED directly based on numBars (no flicker)
    BAR1 = (numBars >= 1);
    BAR2 = (numBars >= 2);
    BAR3 = (numBars >= 3);
    BAR4 = (numBars >= 4);
    BAR5 = (numBars >= 5);
    BAR6 = (numBars >= 6);
    BAR7 = (numBars >= 7);
    BAR8 = (numBars >= 8);

    // Lower timing pin after writing to LEDs (Part 2.3)
//    TIMING_PIN = 0;
}
/*------------------------------- Footnotes -------------------------------*/
/*------------------------------ End of file ------------------------------*/
```

Part 2: Measuring Motor Speed

❑ 2.1) Using your code from Part 1 as a base, add an Input Capture interrupt response routine to measure the period of the encoder signal. Hook up a single channel of the encoder output to the appropriate input line of the PIC32. The 5V supply for the encoder and display should come from the breadboard power supply. Be sure to map the input capture to a 5V tolerant pin on the PIC32!

❑ 2.2) Add code to the mainline (non-interrupt-driven) code that will write a scaled version of the current value of the encoder count to 8 of the 10 LEDs (implement an 8-segment bar graph with the length of the bar proportional to the value of the interval between encoder edges). You will need to use a 74ACT244 (or equivalent, e.g. 'VHCT, 'HCT, etc.) to drive the LEDs. Your bar graph should show 1 bar lit at max speed and progressively more bars as the speed drops. Scale it so that it reaches 8 bars just above the slowest possible speed. You should only update the display if there has been a new encoder edge since the last update.
Compile and download this program. Run the program. When this is working, get a coach/TA/Ed to sign off on it

❑ 2.3) Add a few instructions to your code from Part 2.2 to raise an I/O line when you begin the calculation of the scaled value and lower the line after you write the scaled value to the LEDs. How long did the calculation take? Hint: use a 'scope or logic analyzer to measure the high time of the pulse.

❑ 2.4) Add code to a service that will write the speed of the output wheel, as an integer RPM, to the TeraTerm window. Update (re-write) this speed at a rate of 10 times per second (you may use the framework timer library for this).
Compile and download this program. Run the program. When this is working, get a coach/TA/Ed to sign off on it.

❑ 2.5) Add a few instructions to your code from Part 2.4 to raise an I/O line when you begin the calculation of the RPM and lower the line after you complete the calculation. How long did the calculation take?

❑ 2.6) Add a few instructions to your code from Part 2.4 to raise an I/O line immediately before writing the RPM and lower it immediately after writing the RPM. How long did it take to write the RPM to the screen?

In the report:

In addition to the answers from 2.3, 2.5 & 2.6, include a Highlighted version of the source code to the programs that you wrote for Part 2. 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 6 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 the submission of raw C code to the last step. Please make sure that your files are named such that we can tell which files go with which parts of the lab.

Answer 2.3

3.1 microseconds

Answer 2.5

900 nanoseconds

Answer 2.6

199 – 208 microseconds

Source code [already added in part 1 so I am not pasting it here again]

Part 3: Mapping a DC Motor

- ❑ 3.1) Using your code from Part 2 as a base, adjust the PWM frequency to approximately 250Hz. Step through drive duty cycles from 0 to 100% in 10% increments and record the speed of the motor at each duty cycle. Be sure to wait until the motor speed stabilizes before recording the speed. For regions where there is a large change in speed between two adjacent 10% duty cycle points, go back and add 2 more duty cycle points between them. Create a plot of Speed (Y) vs Duty Cycle (X)
- ❑ 3.2) Repeat Part 3.1 with the PWM Frequency set to approximately 500Hz. (add to the same graph)
- ❑ 3.3) Repeat Part 3.1 with the PWM Frequency set to approximately 1000Hz. (add to the same graph)
- ❑ 3.4) Repeat Part 3.1 with the PWM Frequency set to approximately 2000Hz. (add to the same graph)
- ❑ 3.5) Repeat Part 3.1 with the PWM Frequency set to approximately 10000Hz. (add to the same graph)

In the report:

Include a separate table for of duty cycle and corresponding motor speed for each of the sub-parts. Include a single plot of RPM (Y) as a function of Duty Cycle (X) using the data from each of the sub-parts (label the lines with the corresponding frequency).

250 hz

duty cycle	speed
0	0
10	0
20	0
30	0
35	14
36	11
38	13
40	20
50	28
53	34
56	36
60	38
70	43
80	47
90	50
100	55

500 hz duty cycle	speed
0	0
10	0
20	0
30	0
40	0
43	13
46	18
50	23
54	25
57	30
60	32
70	38
80	43
90	49
100	54

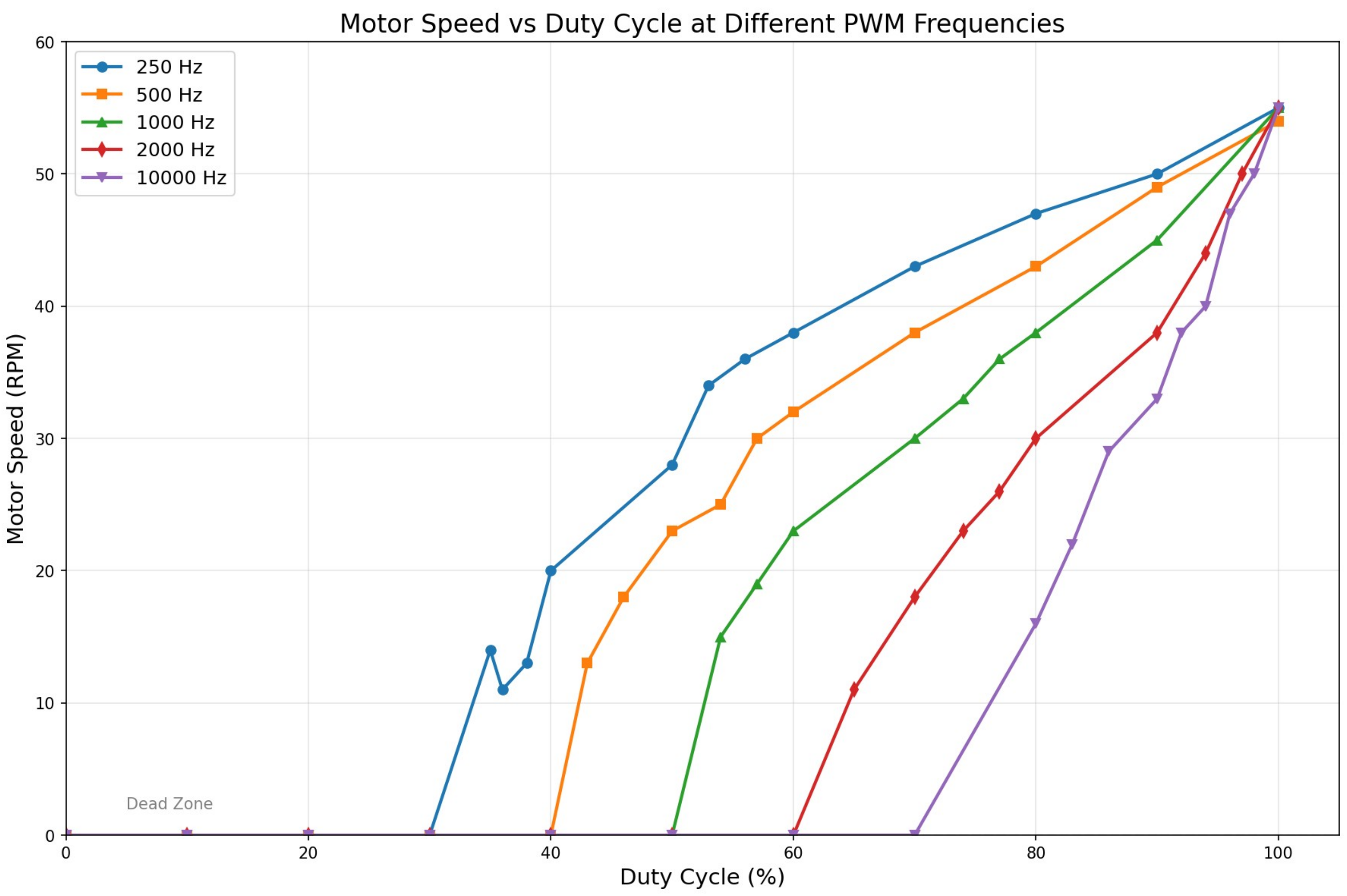
1000 hz duty cycle	speed
0	0
10	0
20	0
30	0
40	0
50	0
54	15
57	19
60	23
70	30
74	33
77	36
80	38
90	45
100	55

2000 hz duty cycle	speed
0	0
10	0
20	0
30	0
40	0
50	0

60	0
65	11
70	18
74	23
77	26
80	30
90	38
94	44
97	50
100	55

10000 hz

duty cycle	speed
0	0
10	0
20	0
30	0
40	0
50	0
60	0
70	0
80	16
83	22
86	29
90	33
92	38
94	40
96	47
98	50
100	55


Motor Speed vs Duty Cycle at Different PWM Frequencies
250 Hz
500 Hz
1000 Hz
2000 Hz
10000 Hz
Dead Zone
Motor Speed (RPM)
Duty Cycle (%)
0
10
20
30
40
50
60
0
20
40
60
80
100