BLDC Motor Drive Using Six-Step Trapezoidal Voltage Control

ECEN 442/742 – DSP-Based Motion Control (Fall 2024)

Matthew McLean **Max Kotas**

January 5, 2025

Abstract

This report presents the design and implementation of a Brushless DC (BLDC) motor control system based on six-step trapezoidal voltage modulation. Hall sensor feedback is used to determine the rotor position, and the motor is driven via a three-phase inverter controlled by pulse-width modulation (PWM). Both open-loop and closed-loop strategies were investigated, with the closed-loop approach leveraging a proportional-integral (PI) controller for robust speed regulation. Key experimental results include speed measurement validation, current waveform analysis, and successful direction reversal. This document details the methodology, findings, and lessons learned from the project.

Contents

1 Introduction

The primary objective of this project is to implement and analyze a BLDC motor drive system using six-step trapezoidal voltage control. BLDC motors are widely used in various industries due to their high efficiency, reliability, and low maintenance requirements. Trapezoidal voltage modulation, also known as six-step commutation, operates by applying voltage pulses to the motor's phases based on the rotor's position. In this work, the rotor position is detected via Hall sensors, and an embedded digital signal processor (DSP) serves as the control platform.

This report addresses both open-loop and closed-loop configurations. The open-loop control provides a baseline for observing motor behavior, while a closed-loop PI controller refines speed regulation. Direction reversal capability further demonstrates the flexibility of the control structure.

2 System Overview

2.1 Hardware Description

The hardware configuration for this project includes:

- **BLDC Motor:** Equipped with three Hall-effect sensors for rotor position feedback.
- **Three-Phase Inverter:** Driven by PWM signals from the DSP to power the BLDC motor.
- **DSP Board:** Generates PWM signals and reads Hall sensor inputs. GPIO pins are used for both PWM outputs and Hall sensor inputs.
- **Measuring Instruments:** A laser tachometer validates speed measurements, while an oscilloscope captures PWM and Hall sensor signals.

2.2 Software Architecture

The software component of the system consists of:

- **GPIO Configuration:** PWM outputs on selected channels (GPIO0–GPIO5) and digital inputs for Hall sensors (GPIO24–GPIO26).
- **Control Algorithms:** Open-loop control logic for baseline testing and a PI controller for closed-loop speed regulation.

- **Peripheral Initialization:** Setup of the DSP's enhanced PWM (EPWM) modules, interrupt service routines (ISRs), and timers for reliable rotor position sampling.

3 Methodology

3.1 Six-Step Trapezoidal Voltage Control

The BLDC motor is driven using six distinct voltage patterns that align with the Hall sensors' detected rotor positions. As the rotor transitions through each sector, the corresponding pair of motor phases is energized, creating a rotating magnetic field. This method is both cost-effective and relatively straightforward to implement.

3.2 PWM Configuration

PWM signals for phases A, B, and C are generated at a frequency of $20kHz$. Duty cycle adjustments are carried out by manipulating the `CMPA` and `CMPB` registers within the DSP. These values are updated in real-time to achieve the desired speed or torque output. Figure 2 (placeholder) illustrates sample waveforms for phases A and B.

3.3 Hall Sensor Integration

Three digital Hall sensors (H1, H2, and H3) are read via GPIO24, GPIO25, and GPIO26. The sensor signals encode rotor position and are mapped to the six commutation states. The commutation logic ensures that the correct phase pairs are driven at each rotor sector to maintain continuous rotation.

3.4 Speed Estimation and Validation

Speed calculation is based on the frequency of Hall sensor state transitions. For a motor with a known number of poles (P), the speed in revolutions per minute (RPM) is calculated as:

$$\text{Speed (RPM)} = \frac{60 \times f_{\text{trans}} \times 2}{P}, \tag{1}$$

where f_{trans} is the Hall sensor transition frequency, and the factor of 2 accounts for transitions per electrical cycle. A laser tachometer is used to corroborate the computed speed against actual measurements.

3.5 Closed-Loop Control (PI Controller)

Closed-loop speed control is implemented using a PI controller, which minimizes the error between the reference speed (*speed_ref*) and the actual measured speed. The proportional term quickly reduces large deviations, while the integral term counters steady-state error. The controller output adjusts the PWM duty cycle in real-time, ensuring stable speed tracking.

3.6 Direction Reversal

Direction reversal is realized through a simple modification of the commutation logic. By reordering the sequence in which phases are energized, the rotor's rotation can be reversed without requiring any additional hardware changes.

4 Results

4.1 Hall Sensor Outputs

Figure 1 (placeholder) captures the Hall sensor outputs, labeled H1, H2, and H3, as observed from the DSP's capture interface. These signals exhibit the expected 120 phase separation in accordance with the six-step commutation strategy.

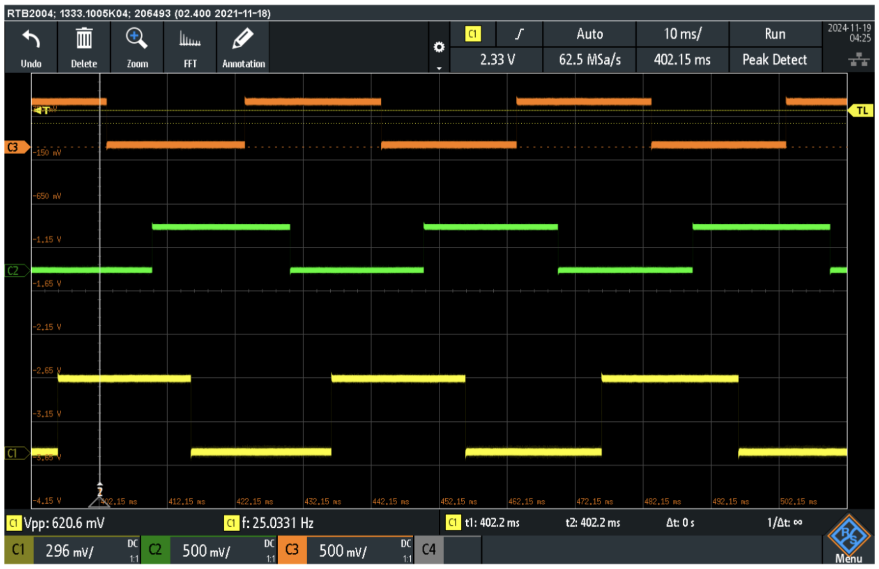


Figure 1: Representative Hall sensor signals (H1, H2, and H3).

4.2 PWM Signals

As seen in Figure 2 (placeholder), the PWM waveforms for phases A and B exhibit a frequency of $20kHz$. Variations in duty cycle reflect the real-time adjustments implemented by either the open-loop or PI control routines.

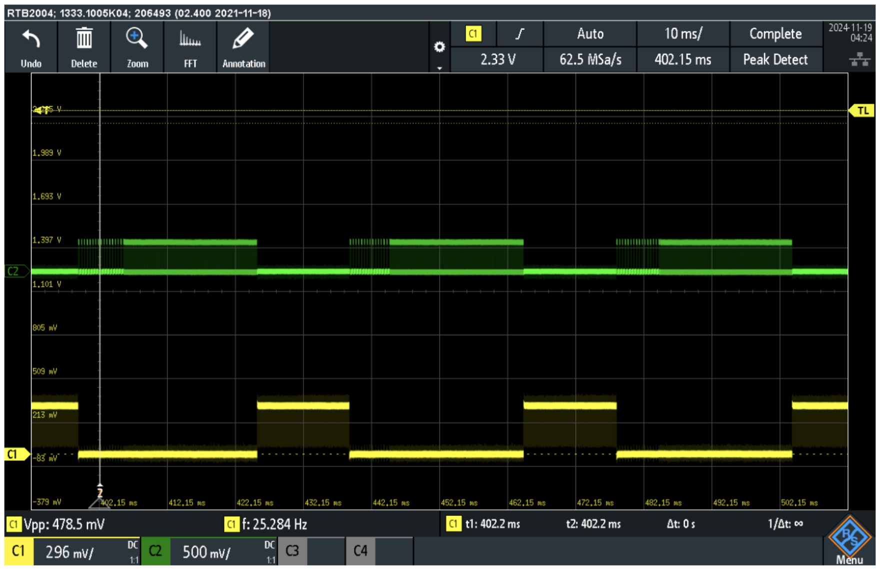
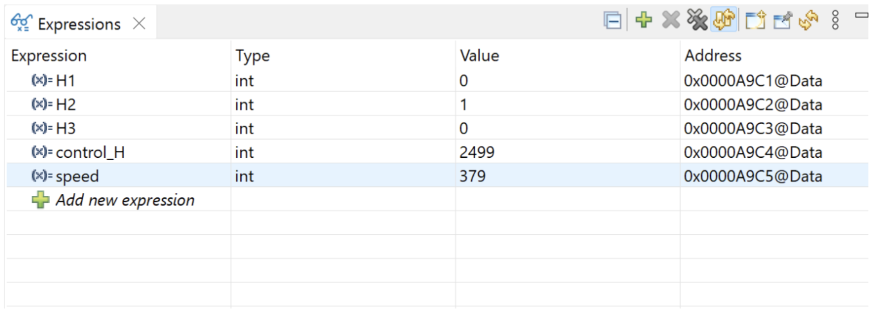


Figure 2: Sample PWM signals for phases A and B (PWM1A and PWM1B).

4.3 Speed Tracking and Regulation

In closed-loop mode, the motor speed was observed via the watch window within the DSP environment. Figure 3 (placeholder) highlights how the actual motor speed converges to the commanded reference speed (*speed_ref*) over time. The PI controller exhibited stable performance, with minimal overshoot and negligible steady-state error.



Expressions

Expression	Type	Value	Address
(x)= H1	int	0	0x0000A9C1@Data
(x)= H2	int	1	0x0000A9C2@Data
(x)= H3	int	0	0x0000A9C3@Data
(x)= control_H	int	2499	0x0000A9C4@Data
(x)= speed	int	379	0x0000A9C5@Data
Add new expression			

Figure 3: Speed tracking performance under PI control.

4.4 Direction Reversal Experiment

A direction reversal test was conducted by swapping the commutation sequence. The system reversed rotational direction seamlessly while maintaining accurate speed control, confirming the effectiveness of the software-based direction control.

5 Discussion

5.1 Comparison of Open-Loop vs. Closed-Loop

Open-loop operation established an initial performance benchmark, verifying basic commutation correctness and hardware functionality. However, speed variations under load were notable in open-loop mode. Closed-loop PI control significantly mitigated these variations, providing stable and precise speed regulation.

5.2 Speed Calculation Accuracy

Comparison between Hall-sensor-based calculations and the laser tachometer readings showed minimal deviation. This close agreement underscores the reliability of both the sensing hardware and the mathematical model used to estimate motor speed.

5.3 Practical Considerations

Although six-step trapezoidal control is relatively straightforward, it can produce higher torque ripple and more acoustic noise compared to more advanced control methods (e.g., field-oriented control). Additionally, precise calibration of the Hall sensors is vital for accurate commutation, and phase misalignment can degrade performance.

6 Conclusion and Future Work

This project successfully demonstrated a BLDC motor drive system utilizing six-step trapezoidal voltage control. Hall sensor feedback enabled accurate rotor position detection, while PWM control via the DSP facilitated precise voltage modulation. The transition from open-loop to closed-loop PI control underscored the benefits of feedback in terms of speed regulation and load disturbance rejection.

Potential avenues for future exploration include:

- **Sensorless Control:** Eliminating Hall sensors and employing back-EMF detection or other advanced estimation techniques.
- **Field-Oriented Control (FOC):** Investigating more sophisticated control algorithms for enhanced torque ripple performance and efficiency.
- **Higher Resolution Feedback:** Using incremental encoders or resolvers for improved speed and position accuracy.

A Code Listings

A.1 Open-Loop Control (main.c)

```
//###########################################################################
// Created on: 11/27/2024
// Author: Max Kotas
//###########################################################################

#include "F28x_Project.h"   // Device Header File and Examples Include File

//***************************************************************************
// Global Definitions and Variables
//***************************************************************************
#define EPWM_TIMER_TBPRD  4999
#define EPWM_TIMER_TBSTEP 2499

int16  numPoles         = 8;
int32  CPU_TIMER1_PRD   = 499999999;
int16  H1               = 0;
int16  H2               = 0;
int16  H3               = 0;
int16  control_H        = 2499;
int16  speed            = 0;

//***************************************************************************
// Function Prototypes
//***************************************************************************
void InitEPwm1Example(void);
void InitEPwm2Example(void);
void InitEPwm3Example(void);

void sector1(void);
void sector2(void);
void sector3(void);
void sector4(void);
void sector5(void);
void sector6(void);

interrupt void xint1_isr(void);
```

```
//****************************************************************************
// Main
//****************************************************************************
void main(void)
{
    // Initialize System Control (PLL, WatchDog, enable Peripheral Clocks)
    InitSysCtrl();

    // Clear all interrupts and initialize PIE vector table
    DINT;
    InitPieCtrl();

    IER = 0x0000;
    IFR = 0x0000;
    InitPieVectTable();

    //--- Configure GPIO for PWM ---
    EALLOW;
    // GPIO0 -> PWM1A
    GpioCtrlRegs.GPAGMUX1.bit.GPIO0  = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO0   = 1;

    // GPIO1 -> PWM1B
    GpioCtrlRegs.GPAGMUX1.bit.GPIO1  = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO1   = 1;

    // GPIO2 -> PWM2A
    GpioCtrlRegs.GPAGMUX1.bit.GPIO2  = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO2   = 1;

    // GPIO3 -> PWM2B
    GpioCtrlRegs.GPAGMUX1.bit.GPIO3  = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO3   = 1;

    // GPIO4 -> PWM3A
    GpioCtrlRegs.GPAGMUX1.bit.GPIO4  = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO4   = 1;

    // GPIO5 -> PWM3B
    GpioCtrlRegs.GPAGMUX1.bit.GPIO5  = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO5   = 1;

    //--- Configure GPIO for Hall sensors ---
    GpioCtrlRegs.GPAGMUX2.bit.GPIO24 = 0;
    GpioCtrlRegs.GPAMUX2.bit.GPIO24  = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO24   = 0;  // Input

    GpioCtrlRegs.GPAGMUX2.bit.GPIO25 = 0;
    GpioCtrlRegs.GPAMUX2.bit.GPIO25  = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO25   = 0;  // Input

    GpioCtrlRegs.GPAGMUX2.bit.GPIO26 = 0;
    GpioCtrlRegs.GPAMUX2.bit.GPIO26  = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO26   = 0;  // Input
```

```
// Disable PWM clock during setup
CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 0;
EDIS;

// Initialize EPWMs
InitEPwm1Example();
InitEPwm2Example();
InitEPwm3Example();

// Enable PWM clock
EALLOW;
CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;
EDIS;

//--- Configure XINT1 for Hall Sensor H1 (GPIO24) ---
EALLOW;
PieVectTable.XINT1_INT            = &xint1_isr;
GPIO_SetupXINT1Gpio(24);
XintRegs.XINT1CR.bit.POLARITY     = 1;  // Rising edge interrupt
XintRegs.XINT1CR.bit.ENABLE       = 1;
PieCtrlRegs.PIECTRL.bit.ENPIE     = 1;
PieCtrlRegs.PIEIER1.bit.INTx4     = 1;  // XINT1 is in PIE group 1
IER |= M_INT1;                         // Enable INT1 in core
EINT;                                  // Global interrupt enable

// Qualifier settings for GPIO24 to avoid false triggers
GpioCtrlRegs.GPAQSEL2.bit.GPIO24  = 2;   // 6-sample qualification
GpioCtrlRegs.GPACTRL.bit.QUALPRD3 = 255; // Qual period
EDIS;

//--- Setup CPU Timer1 for speed measurement ---
CpuTimer1Regs.TCR.bit.TSS = 1;              // Stop Timer1
CpuTimer1Regs.PRD.all     = CPU_TIMER1_PRD;  // Period
CpuTimer1Regs.TCR.bit.TRB = 1;              // Reload
// Timer starts after the first Hall interrupt triggers
// in the xint1_isr().

//----------------------------------------------------------------------
// Infinite Loop: Open-loop commutation based on Hall sensor inputs
//----------------------------------------------------------------------
for(;;)
{
    H1 = GpioDataRegs.GPADAT.bit.GPIO24;
    H2 = GpioDataRegs.GPADAT.bit.GPIO25;
    H3 = GpioDataRegs.GPADAT.bit.GPIO26;

    if(H1 == 1 && H2 == 0 && H3 == 1)
    {
        sector1();
    }
    else if(H1 == 1 && H2 == 0 && H3 == 0)
    {
        sector2();
    }
    else if(H1 == 1 && H2 == 1 && H3 == 0)
```

```
        {
            sector3();
        }
        else if(H1 == 0 && H2 == 1 && H3 == 0)
        {
            sector4();
        }
        else if(H1 == 0 && H2 == 1 && H3 == 1)
        {
            sector5();
        }
        else if(H1 == 0 && H2 == 0 && H3 == 1)
        {
            sector6();
        }
    }
}

//*****************************************************************************
// Sector (Commutation) Functions
//*****************************************************************************
void sector1(void)
{
    EPwm1Regs.CMPA.bit.CMPA = control_H;
    EPwm2Regs.CMPA.bit.CMPA = 0;
    EPwm3Regs.CMPA.bit.CMPA = 0;

    EPwm1Regs.CMPB.bit.CMPB = 0;
    EPwm2Regs.CMPB.bit.CMPB = EPWM_TIMER_TBPRD;
    EPwm3Regs.CMPB.bit.CMPB = 0;
}

void sector2(void)
{
    EPwm1Regs.CMPA.bit.CMPA = control_H;
    EPwm2Regs.CMPA.bit.CMPA = 0;
    EPwm3Regs.CMPA.bit.CMPA = 0;

    EPwm1Regs.CMPB.bit.CMPB = 0;
    EPwm2Regs.CMPB.bit.CMPB = 0;
    EPwm3Regs.CMPB.bit.CMPB = EPWM_TIMER_TBPRD;
}

void sector3(void)
{
    EPwm1Regs.CMPA.bit.CMPA = 0;
    EPwm2Regs.CMPA.bit.CMPA = control_H;
    EPwm3Regs.CMPA.bit.CMPA = 0;

    EPwm1Regs.CMPB.bit.CMPB = 0;
    EPwm2Regs.CMPB.bit.CMPB = 0;
    EPwm3Regs.CMPB.bit.CMPB = EPWM_TIMER_TBPRD;
}

void sector4(void)
```

```
{
    EPwm1Regs.CMPA.bit.CMPA = 0;
    EPwm2Regs.CMPA.bit.CMPA = control_H;
    EPwm3Regs.CMPA.bit.CMPA = 0;

    EPwm1Regs.CMPB.bit.CMPB = EPWM_TIMER_TBPRD;
    EPwm2Regs.CMPB.bit.CMPB = 0;
    EPwm3Regs.CMPB.bit.CMPB = 0;
}

void sector5(void)
{
    EPwm1Regs.CMPA.bit.CMPA = 0;
    EPwm2Regs.CMPA.bit.CMPA = 0;
    EPwm3Regs.CMPA.bit.CMPA = control_H;

    EPwm1Regs.CMPB.bit.CMPB = EPWM_TIMER_TBPRD;
    EPwm2Regs.CMPB.bit.CMPB = 0;
    EPwm3Regs.CMPB.bit.CMPB = 0;
}

void sector6(void)
{
    EPwm1Regs.CMPA.bit.CMPA = 0;
    EPwm2Regs.CMPA.bit.CMPA = 0;
    EPwm3Regs.CMPA.bit.CMPA = control_H;

    EPwm1Regs.CMPB.bit.CMPB = 0;
    EPwm2Regs.CMPB.bit.CMPB = EPWM_TIMER_TBPRD;
    EPwm3Regs.CMPB.bit.CMPB = 0;
}

//****************************************************************************
// EPWM Configuration Functions
//****************************************************************************
void InitEPwm1Example()
{
    EPwm1Regs.TBCTL.bit.CTRMODE   = 0;                  // Count up
    EPwm1Regs.TBPRD               = EPWM_TIMER_TBPRD;   // Period
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = 0;                  // TBCLK = EPWMCLK
    EPwm1Regs.TBCTL.bit.CLKDIV    = 0;

    EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm1Regs.CMPA.bit.CMPA = 0;
    EPwm1Regs.CMPB.bit.CMPB = 0;

    EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;     // PWM high at start
    EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;   // PWM low when counter = CMPA

    EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    EPwm1Regs.AQCTLB.bit.ZRO = AQ_SET;
```

```
}

void InitEPwm2Example()
{
    EPwm2Regs.TBCTL.bit.CTRMODE   = 0;
    EPwm2Regs.TBPRD               = EPWM_TIMER_TBPRD;
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = 0;
    EPwm2Regs.TBCTL.bit.CLKDIV    = 0;

    EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm2Regs.CMPA.bit.CMPA = 0;
    EPwm2Regs.CMPB.bit.CMPB = 0;

    EPwm2Regs.AQCTLA.bit.ZRO = AQ_SET;
    EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;

    EPwm2Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    EPwm2Regs.AQCTLB.bit.ZRO = AQ_SET;
}

void InitEPwm3Example()
{
    EPwm3Regs.TBCTL.bit.CTRMODE   = 0;
    EPwm3Regs.TBPRD               = EPWM_TIMER_TBPRD;
    EPwm3Regs.TBCTL.bit.HSPCLKDIV = 0;
    EPwm3Regs.TBCTL.bit.CLKDIV    = 0;

    EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm3Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm3Regs.CMPA.bit.CMPA = 0;
    EPwm3Regs.CMPB.bit.CMPB = 0;

    EPwm3Regs.AQCTLA.bit.ZRO = AQ_SET;
    EPwm3Regs.AQCTLA.bit.CAU = AQ_CLEAR;

    EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    EPwm3Regs.AQCTLB.bit.ZRO = AQ_SET;
}

//*****************************************************************************
// Interrupt Service Routine
//*****************************************************************************
interrupt void xint1_isr(void)
{
    // Stop the timer to read how much it has counted down
    CpuTimer1Regs.TCR.bit.TSS = 1;
    if (CpuTimer1Regs.TIM.all != CPU_TIMER1_PRD)
    {
        // Speed calculation (RPM) example, depends on CPU frequency and timer settings
        speed = 60 * (200000000.0) / (numPoles / 2) / (CPU_TIMER1_PRD - CpuTimer1Regs.TIM.all);
    }

    // Reload and start the timer again
    CpuTimer1Regs.TCR.bit.TRB = 1;
    CpuTimer1Regs.TCR.bit.TSS = 0;

    // Acknowledge interrupt to receive more interrupts from group 1
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

A.2 Closed-Loop Control (main2.c)

```
//###########################################################################
// Created on: 11/27/2024
// Author: Max Kotas
//###########################################################################

#include "F28x_Project.h"  // Device Headerfile and Examples Include File

//*****************************************************************************
// Global Definitions and Variables
//*****************************************************************************
#define EPWM_TIMER_TBPRD  4999
#define EPWM_TIMER_TBSTEP 2499

int16 numPoles        = 8;
int32 CPU_TIMER1_PRD  = 499999999;
int16 H1              = 0;
int16 H2              = 0;
int16 H3              = 0;
int16 control_H       = 2499;   // PWM duty-cycle control variable
int16 speed           = 0;      // Measured speed (updated in xint1_isr)

// -- PI Control Variables for Closed-Loop Speed Regulation --
int16 speed_ref       = 600;    // Commanded speed (RPM)
int16 speed_error     = 0;      // Error between commanded and measured speed
int16 ki              = 8;      // Integral constant
int16 kp              = 0;      // Proportional constant
int16 Up              = 0;      // Proportional term
int16 Ui              = 0;      // Integral term
int16 Out             = 0;      // PI controller output

//*****************************************************************************
// Function Prototypes
//*****************************************************************************
void InitEPwm1Example(void);
void InitEPwm2Example(void);
void InitEPwm3Example(void);

void sector1(void);
void sector2(void);
void sector3(void);
void sector4(void);
void sector5(void);
void sector6(void);

interrupt void xint1_isr(void);

//*****************************************************************************
// Main
//*****************************************************************************
void main(void)
{
    // System Initialization
    InitSysCtrl();
    DINT;
    InitPieCtrl();
    IER = 0x0000;
    IFR = 0x0000;
    InitPieVectTable();

    // Configure PWM GPIOs
    EALLOW;
    GpioCtrlRegs.GPAGMUX1.bit.GPIO0  = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO0   = 1; // PWM1A
    GpioCtrlRegs.GPAGMUX1.bit.GPIO1  = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO1   = 1; // PWM1B

    GpioCtrlRegs.GPAGMUX1.bit.GPIO2  = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO2   = 1; // PWM2A
    GpioCtrlRegs.GPAGMUX1.bit.GPIO3  = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO3   = 1; // PWM2B

    GpioCtrlRegs.GPAGMUX1.bit.GPIO4  = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO4   = 1; // PWM3A
    GpioCtrlRegs.GPAGMUX1.bit.GPIO5  = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO5   = 1; // PWM3B

    // Configure Hall Sensors (GPIO24, GPIO25, GPIO26) as inputs
    GpioCtrlRegs.GPAGMUX2.bit.GPIO24 = 0;
    GpioCtrlRegs.GPAMUX2.bit.GPIO24  = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO24   = 0;

    GpioCtrlRegs.GPAGMUX2.bit.GPIO25 = 0;
    GpioCtrlRegs.GPAMUX2.bit.GPIO25  = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO25   = 0;

    GpioCtrlRegs.GPAGMUX2.bit.GPIO26 = 0;
    GpioCtrlRegs.GPAMUX2.bit.GPIO26  = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO26   = 0;

    // Disable PWM clock during configuration
    CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 0;
    EDIS;

    // Initialize ePWM Modules
    InitEPwm1Example();
    InitEPwm2Example();
    InitEPwm3Example();

    // Re-enable PWM clock
    EALLOW;
    CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;
    EDIS;

    // Setup XINT1 interrupt for Hall sensor on GPIO24
    EALLOW;
    PieVectTable.XINT1_INT = &xint1_isr;
    GPIO_SetupXINT1Gpio(24);
    XintRegs.XINT1CR.bit.POLARITY   = 1;   // Rising-edge interrupt
    XintRegs.XINT1CR.bit.ENABLE     = 1;   // Enable XINT1
    PieCtrlRegs.PIECTRL.bit.ENPIE   = 1;   // Enable the PIE block
    PieCtrlRegs.PIEIER1.bit.INTx4   = 1;   // XINT1 is in PIE group 1
    IER |= M_INT1;                        // Enable GROUP1 in core
    EINT;                                 // Enable Global Interrupts
    GpioCtrlRegs.GPAQSEL2.bit.GPIO24  = 2; // Qualification (6-sample)
    GpioCtrlRegs.GPACTRL.bit.QUALPRD3 = 255;
    EDIS;

    // Setup CPU Timer1 for speed measurement
    CpuTimer1Regs.TCR.bit.TSS = 1;                // Stop timer
    CpuTimer1Regs.PRD.all     = CPU_TIMER1_PRD;   // Period
    CpuTimer1Regs.TCR.bit.TRB = 1;                // Reload

    //-------------------------------------------------------------------------
    // Main Control Loop
    //-------------------------------------------------------------------------
    for(;;)
    {
        // 1) Read Hall Sensor Inputs
        H1 = GpioDataRegs.GPADAT.bit.GPIO24;
        H2 = GpioDataRegs.GPADAT.bit.GPIO25;
        H3 = GpioDataRegs.GPADAT.bit.GPIO26;

        // 2) Update Commutation Sectors
        if(H1 == 1 && H2 == 0 && H3 == 1)
        {
            sector1();
        }
        else if(H1 == 1 && H2 == 0 && H3 == 0)
        {
            sector2();
        }
        else if(H1 == 1 && H2 == 1 && H3 == 0)
        {
            sector3();
        }
        else if(H1 == 0 && H2 == 1 && H3 == 0)
        {
            sector4();
        }
```

```
        else if(H1 == 0 && H2 == 1 && H3 == 1)
        {
            sector5();
        }
        else if(H1 == 0 && H2 == 0 && H3 == 1)
        {
            sector6();
        }

        // 3) Closed-Loop PI Control Example (Optional)
        //    Adjust 'control_H' (duty cycle) in real-time
        speed_error = speed_ref - speed;
        Up          = kp * speed_error;
        Ui         += ki * speed_error;
        Out         = Up + Ui;

        // Add clamping if desired to avoid integral windup:
        // if(Out > EPWM_TIMER_TBPRD) Out = EPWM_TIMER_TBPRD;
        // if(Out < 0) Out = 0;

        control_H   = (int16)Out;
        // 'control_H' is used in sectorX() functions to set CMPA.
    }
}

//*****************************************************************************
// ePWM Configuration
//*****************************************************************************
void InitEPwm1Example(void)
{
    EPwm1Regs.TBCTL.bit.CTRMODE   = 0;                  // Up-count
    EPwm1Regs.TBPRD               = EPWM_TIMER_TBPRD;
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = 0;                  // TBCLK = EPWMCLK
    EPwm1Regs.TBCTL.bit.CLKDIV    = 0;

    EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm1Regs.CMPA.bit.CMPA = 0;
    EPwm1Regs.CMPB.bit.CMPB = 0;

    EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;    // PWM high at start
    EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;  // PWM low at CMPA

    // Not strictly used here, but we set them for completeness
    EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    EPwm1Regs.AQCTLB.bit.ZRO = AQ_SET;
}

void InitEPwm2Example(void)
{
    EPwm2Regs.TBCTL.bit.CTRMODE   = 0;
    EPwm2Regs.TBPRD               = EPWM_TIMER_TBPRD;
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = 0;
    EPwm2Regs.TBCTL.bit.CLKDIV    = 0;

    EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm2Regs.CMPA.bit.CMPA = 0;
    EPwm2Regs.CMPB.bit.CMPB = 0;

    EPwm2Regs.AQCTLA.bit.ZRO = AQ_SET;
    EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;

    EPwm2Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    EPwm2Regs.AQCTLB.bit.ZRO = AQ_SET;
}

void InitEPwm3Example(void)
{
    EPwm3Regs.TBCTL.bit.CTRMODE   = 0;
    EPwm3Regs.TBPRD               = EPWM_TIMER_TBPRD;
    EPwm3Regs.TBCTL.bit.HSPCLKDIV = 0;
    EPwm3Regs.TBCTL.bit.CLKDIV    = 0;

    EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm3Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm3Regs.CMPA.bit.CMPA = 0;
    EPwm3Regs.CMPB.bit.CMPB = 0;

    EPwm3Regs.AQCTLA.bit.ZRO = AQ_SET;
    EPwm3Regs.AQCTLA.bit.CAU = AQ_CLEAR;

    EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    EPwm3Regs.AQCTLB.bit.ZRO = AQ_SET;
}

//*****************************************************************************
// Sector (Commutation) Functions
//*****************************************************************************
void sector1(void)
{
    EPwm1Regs.CMPA.bit.CMPA = control_H;
    EPwm2Regs.CMPA.bit.CMPA = 0;
    EPwm3Regs.CMPA.bit.CMPA = 0;

    EPwm1Regs.CMPB.bit.CMPB = 0;
    EPwm2Regs.CMPB.bit.CMPB = EPWM_TIMER_TBPRD;
    EPwm3Regs.CMPB.bit.CMPB = 0;
}

void sector2(void)
```

```
{
    EPwm1Regs.CMPA.bit.CMPA = control_H;
    EPwm2Regs.CMPA.bit.CMPA = 0;
    EPwm3Regs.CMPA.bit.CMPA = 0;

    EPwm1Regs.CMPB.bit.CMPB = 0;
    EPwm2Regs.CMPB.bit.CMPB = 0;
    EPwm3Regs.CMPB.bit.CMPB = EPWM_TIMER_TBPRD;
}

void sector3(void)
{
    EPwm1Regs.CMPA.bit.CMPA = 0;
    EPwm2Regs.CMPA.bit.CMPA = control_H;
    EPwm3Regs.CMPA.bit.CMPA = 0;

    EPwm1Regs.CMPB.bit.CMPB = 0;
    EPwm2Regs.CMPB.bit.CMPB = 0;
    EPwm3Regs.CMPB.bit.CMPB = EPWM_TIMER_TBPRD;
}

void sector4(void)
{
    EPwm1Regs.CMPA.bit.CMPA = 0;
    EPwm2Regs.CMPA.bit.CMPA = control_H;
    EPwm3Regs.CMPA.bit.CMPA = 0;

    EPwm1Regs.CMPB.bit.CMPB = EPWM_TIMER_TBPRD;
    EPwm2Regs.CMPB.bit.CMPB = 0;
    EPwm3Regs.CMPB.bit.CMPB = 0;
}

void sector5(void)
{
    EPwm1Regs.CMPA.bit.CMPA = 0;
    EPwm2Regs.CMPA.bit.CMPA = 0;
    EPwm3Regs.CMPA.bit.CMPA = control_H;

    EPwm1Regs.CMPB.bit.CMPB = EPWM_TIMER_TBPRD;
    EPwm2Regs.CMPB.bit.CMPB = 0;
    EPwm3Regs.CMPB.bit.CMPB = 0;
}

void sector6(void)
{
    EPwm1Regs.CMPA.bit.CMPA = 0;
    EPwm2Regs.CMPA.bit.CMPA = 0;
    EPwm3Regs.CMPA.bit.CMPA = control_H;

    EPwm1Regs.CMPB.bit.CMPB = 0;
    EPwm2Regs.CMPB.bit.CMPB = EPWM_TIMER_TBPRD;
    EPwm3Regs.CMPB.bit.CMPB = 0;
}

//****************************************************************************
```

```
// Interrupt Service Routine for Hall Sensor (XINT1)
//****************************************************************************
interrupt void xint1_isr(void)
{
    // Stop CPU Timer1 to measure elapsed time (period between Hall pulses)
    CpuTimer1Regs.TCR.bit.TSS = 1;

    if (CpuTimer1Regs.TIM.all != CPU_TIMER1_PRD)
    {
        // Example speed calculation (RPM). Adjust as necessary
        speed = 60 * (200000000.0)
                / (numPoles / 2)
                / (CPU_TIMER1_PRD - CpuTimer1Regs.TIM.all);
    }

    // Reload and restart Timer1
    CpuTimer1Regs.TCR.bit.TRB = 1;
    CpuTimer1Regs.TCR.bit.TSS = 0;

    // Acknowledge PIE interrupt group
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```