Application Note

Sensored Brushed DC Motor Control Based on MSPM0

Jingguo Wang and Eason Zhou

ABSTRACT

This application note describes a sensored brushed DC motor control based on MSPM0C1104. In this solution, MSPM0C1104 utilizes its advanced timer to control the brushed DC motor with the motor's speed and current information, using the PI controller.

The project collateral contains the software project and hardware design, which can be downloaded from the following URL: https://www.ti.com/lit/zip/slaaem1. You can also find it under SDK with this address: `C:\ti\mspm0_sdk_x_xx_xx_xx\examples\nortos\LP_MSPM0C1104\demos\motor_control_bdc_sensor`.

Table of Contents

List of Figures

List of Tables

Trademarks

NexFET™ and Code Composer Studio™ are trademarks of Texas Instruments.
Arm® and Cortex® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
Windows® is a registered trademark of Microsoft Corporation in the United States and other countries.
All trademarks are the property of their respective owners.

1 Introduction

Brushed DC motors are known for their simplicity, ease of control, and cost-effectiveness, making them a preferred choice for applications prioritizing performance, cost efficiency, and reliability. The MSPM0C1104 is part of the MSP highly-integrated ultra-low-power 32-bit microcontroller unit (MCU) family, based on the enhanced Arm® Cortex®-M0+ core platform operating at up to 24MHz frequency, offering advanced features for motor control and making it an ideal choice for driving and controlling brushed DC motors. This document presents a discrete motor drive solution, controlled by MSPM0C1104, which reduces system costs and enhances system design flexibility and is suitable for a wide range of industrial and commercial applications.

2 System Architecture Introduction

Figure 2-1 shows the system block diagram for the motor driver solution. In addition to the MSPM0C1104 and power loop, the system primarily consists of three components: the power supply circuit, the driver circuit, and the sampling circuit.

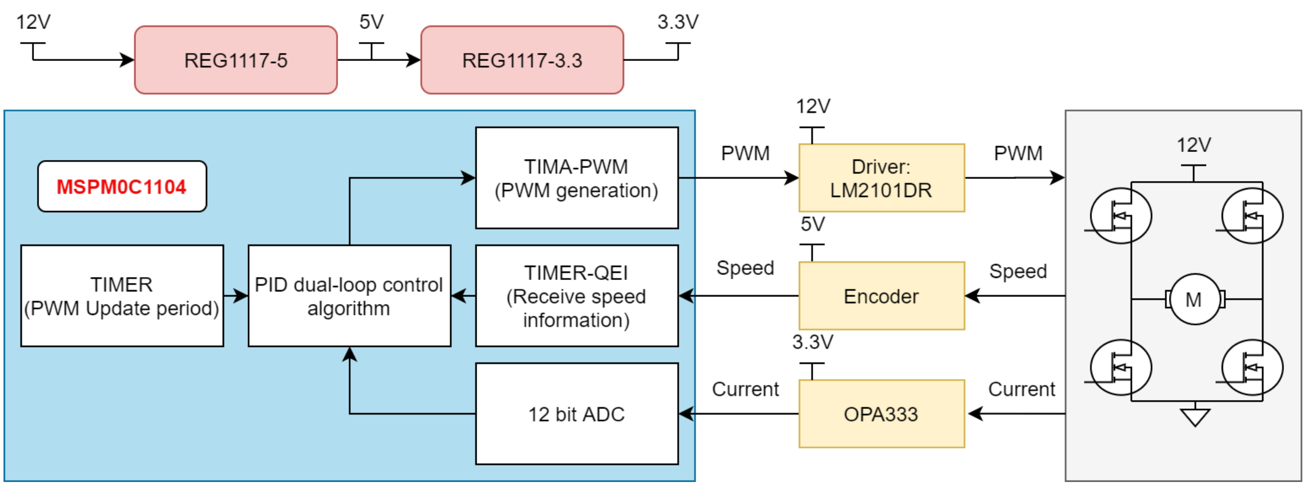

Figure 2-1. System Block Diagram

The sampling circuit collects the motor's current and velocity information through an encoder and operational amplifier, and then sends them to the MCU. The MCU computes the duty cycle through a PID closed-loop and generates PWM signals based on real-time velocity and current information. The MOSFET driver circuit converts the PWM signal generated by the MCU into a driving signal to control the MOSFET switch. The power supply circuit provides proper power rails to all the components in the system, such as MCU, encoder, driver, OPA and so on. A detailed description of the hardware circuitry and a portion of the software control code will be provided as follows.

3 Hardware Design Introduction

Figure 3-1 depicts the schematic of the entire hardware circuit. In the following section, each part of the circuit will be thoroughly explained.

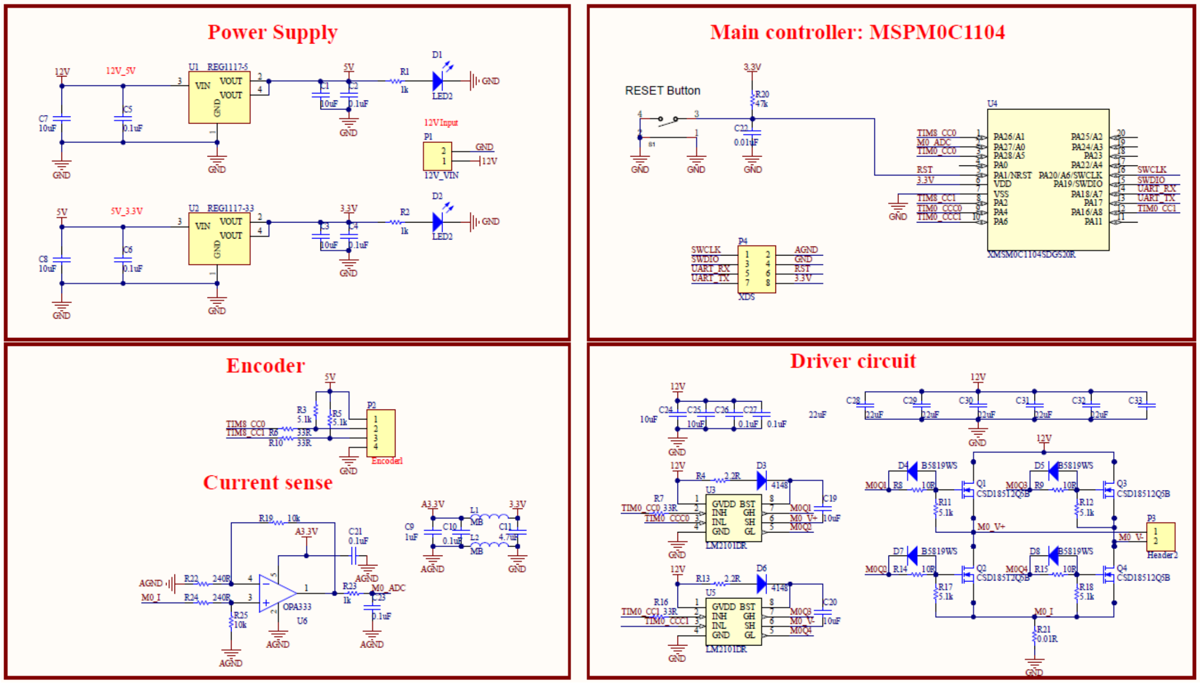

Figure 3-1. Hardware Circuit Schematic

3.1 Power Supply Circuit

The power supply circuit is depicted in Section 3.1, which provides 5V power rail to encoder circuit and 3.3V power rail to MSPM0C1104. The circuit utilizes two fixed-output LDO, leading to reduced noise and a stable output and two LED lights to indicate the normal operation of the power supply rail.

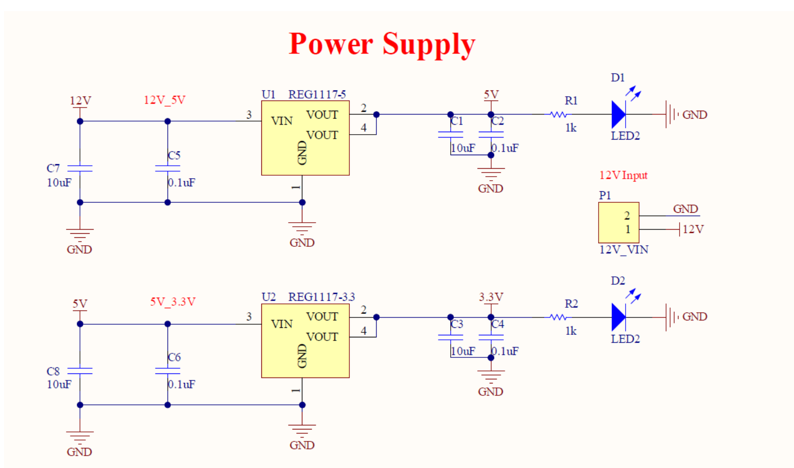

Figure 3-2. Power Supply Circuit

3.2 Drive Circuit

The gate drive and MOSFET in drive circuit are both sourced from Texas Instruments. The LM2101 is a compact, high-voltage gate driver designed to drive both the high-side and the low-side N-channel MOSFETs in a synchronous buck or a half-bridge configuration. And the CSD18512Q5B is 40V, 1.3mΩ, 5mm × 6mm NexFET™ power MOSFET, designed to minimize losses in power conversion applications. The circuit parameters are designed based on the application circuit from the *LM2101 107-V, 0.5-A, 0.8-A Half-Bridge Driver with 8-V UVLO Data Sheet*.

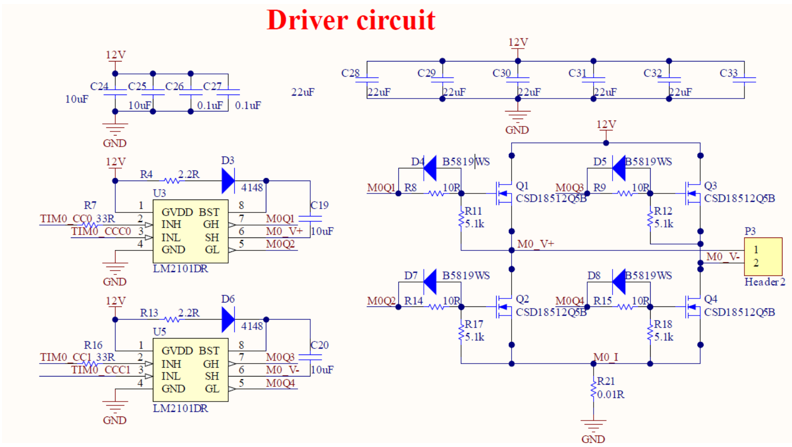

Figure 3-3. Drive Circuit

3.3 Sampling Circuit

The sampling circuit is mainly divided into speed sampling and current sampling. Speed sampling is realized by an encoder, which can send square signal containing speed information to the MCU. As for current feedback circuit, the current is converted to voltage through the sensing resistor. Then through the amplifier circuit, the voltage is amplified to within the sampling range of the analog-to-digital converter (ADC).

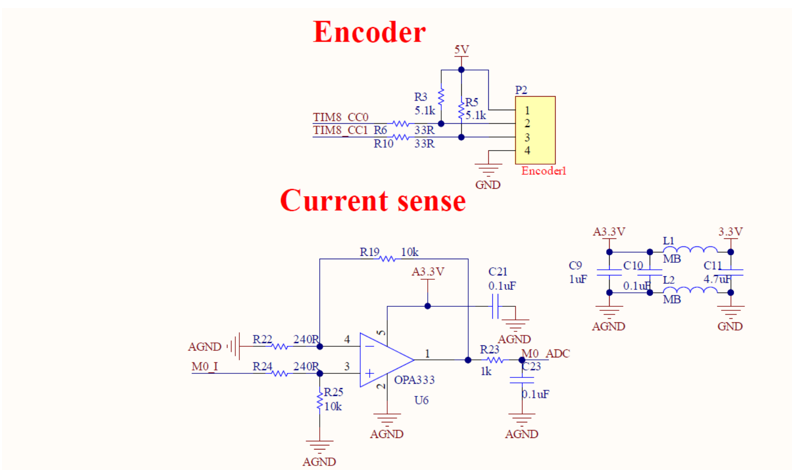

Figure 3-4. Sampling Circuit

3.4 Main Controller Circuit

The peripheral circuit of the MSPM0C1104 mainly consists of the reset circuit and the program burning interfaces, as shown in Figure 3-5.

In this solution, the assignment of pin functions for MSPM0C1104 can refer to Table 3-1. TIM0_CC0 and TIM0_CCC0 are complementary PWM signals, and TIM0_CC1 and TIM0_CCC1 are complementary PWM signals. Reserving the UART interface is for convenient debugging and only four signals are required for burning, including SWCLK, SWDIO, RST and GND.

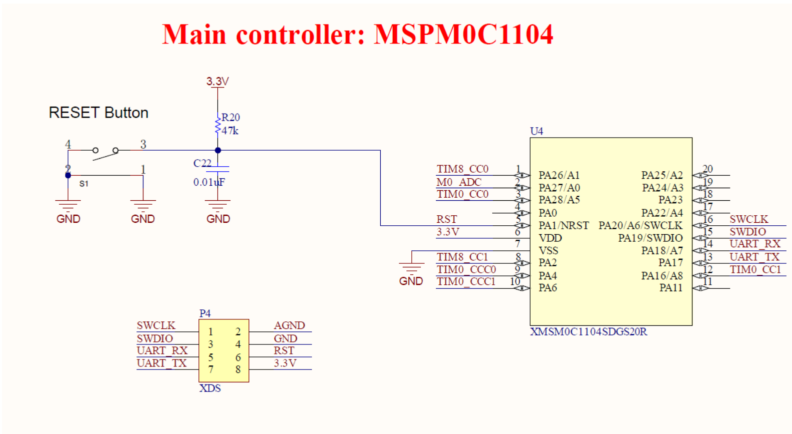

Figure 3-5. Main Controller Circuit

Table 3-1. Hardware Connection

Connection Type	Connection Net	LP-MSPM0LC1104 Pin Number: Pin Name
Power supply input	3.3V	Pin 6: VDD
	GND	Pin 7: VSS
PWM output	TIM0_CC0	Pin 3: PA28
	TIM0_CCC0	Pin 9: PA4
	TIM0_CC1	Pin 12: PA16
	TIM0_CCC1	Pin 10: PA6
ADC input	M0_ADC	Pin 2: PA27
Encoder input	TIM8_CC0	Pin 1: PA26
	TIM8_CC1	Pin 8: PA2
Program burning interfaces	SWCLK	Pin 16: PA20
	SWDIO	Pin 15: PA19
	UART_RX	Pin 14: PA18
	UART_TX	Pin 13: PA17
	RST	Pin 5: PA1

3.5 Control System Introduction

3.5.1 Drive Method

DC brushed motors have various drive methods. In this solution, a relatively simple drive method was used as shown in Figure 3-6. This method only requires a pair of complementary drive signals to operate a half-bridge. And one MOSFET in another half-bridge remains in a constant on-state, while the other is in a constant off-state, depending on the desired direction of rotation.

When the upper transistor in the left half-bridge is conducting, the power supply provides power to the motor, and the motor current flows from left to right. When the upper transistor is turned off, the motor continues to conduct through the lower transistors of the two half-bridges.

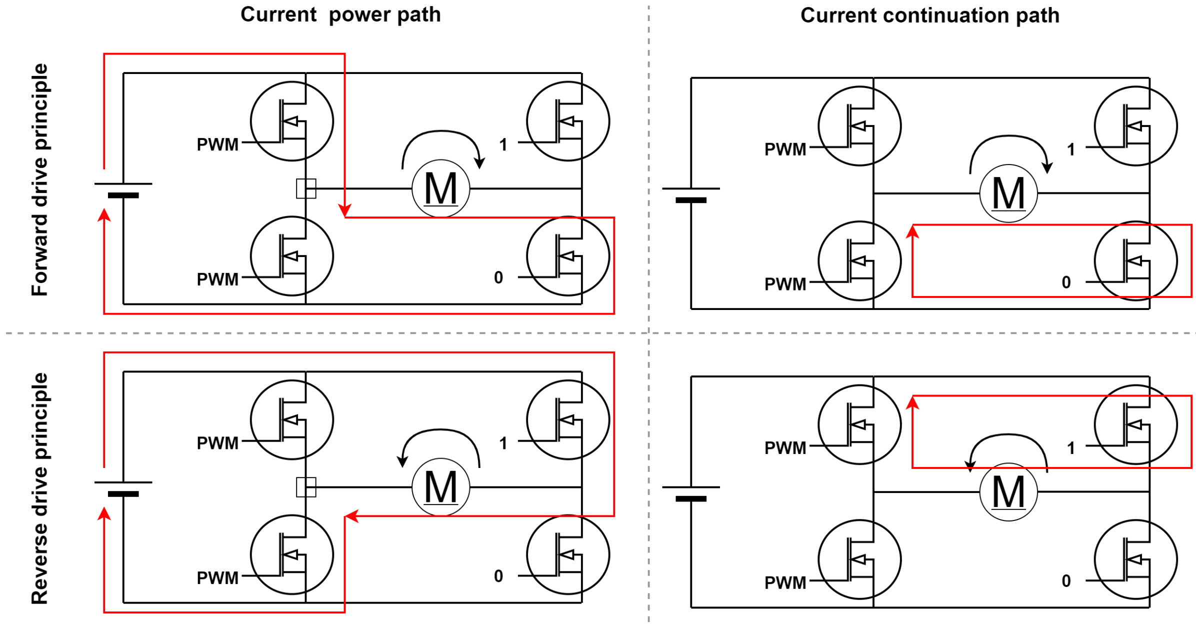

Figure 3-6. Motor Drive Principles

3.5.2 Control Method

In this solution, a dual-loop control strategy was adopted for speed and current as shown in Figure 3-7.

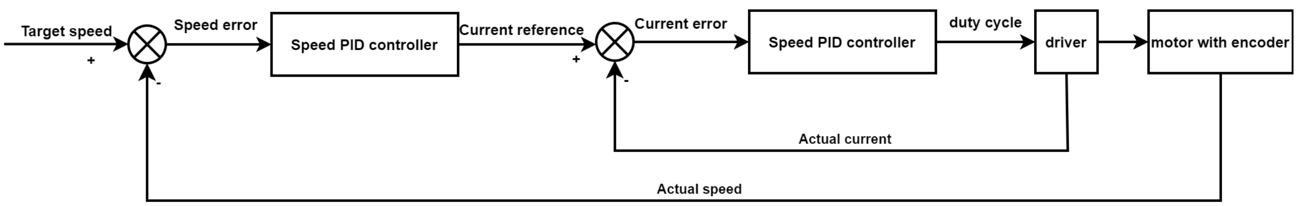

Figure 3-7. Control Block Diagram

In the speed control loop, a target speed is settled and the actual speed is measured. The difference between the target speed and the actual speed is then used to generate an error signal. This error signal is processed by a control algorithm to produce a control command, which adjusts the motor's input voltage or PWM signal to regulate the motor speed.

In the current control loop, the motor current is monitored and adjusted to ensure that sufficient torque is provided under any load condition. The current control loop can also provide overload and short-circuit protection for the motor.

By employing this speed outer loop and current inner loop control method, high-precision and high-performance control of brushed DC motors can be achieved. Consequently, this approach is widely used in applications requiring precise and reliable control, such as industrial automation, robotics, and various motion control applications.

4 Software Design Introduction

The example code can be divided into three main parts: parameter initialization, setting the rotation direction, and calculating the duty cycle output during the timer interrupt. The software execution logic is illustrated in Figure 4-1.

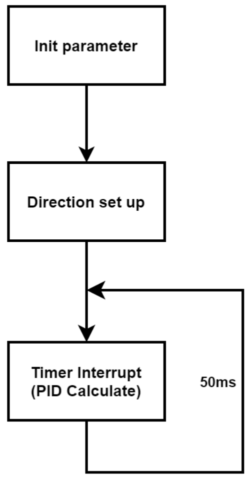

Figure 4-1. Software Flow

4.1 Parameters Initialization

This section primarily involves the initialization of three aspects: motor configuration parameters, speed loop parameters, and current loop parameters, as shown in Figure 4-2. The configuration settings include defining the target speed, run direction (up or down), encoder resolution, record time, maximum and minimum duty cycle and maximum current. The PID controller parameters, including P, I, and D gains, are specified for both speed and current control loops. Additionally, the code declares various volatile variables for capturing counts, recording distances, storing actual speed and current, defining target current, and capturing ADC readings.

```
// motor configure
#define target_speed            (150)    // speed (MAX:333)
#define run_direction           (1)      // 1:up 0:down
#define ENCODER_RESOLUTION      (1320)   // 11*4*30
#define RECORD_TIME             (0.05)   // record time(s)
#define CC_value_MAX            (475)    // D=97%
#define CC_value_MIN            (25)     // D=3%
#define current_MAX             (0.98)

// Speed PID controller parameters
#define PIDSPEED_KP     (0.01)
#define PIDSPEED_KI     (0.0005)
#define PIDSPEED_KD     (0)
volatile float speed_sum_error=0.0;
volatile float speed_error_last=0.0;
volatile int Capture_count = 0;
volatile int Last_count = 0;
volatile int distance=0;
volatile float actual_speed=0;

// current PID controller parameters
#define PIDCURRENT_KP   (250)
#define PIDCURRENT_KI   (100)
#define PIDCURRENT_KD   (0)
volatile float current_sum_error=0.0;
volatile float current_error_last=0.0;
volatile float actual_current=0;
volatile float target_current=0;
volatile int PRE_ADC=0;
volatile int CC_value=25;
```

Figure 4-2. Parameters Initialization

4.2 Direction Setup

After initializing the parameters, it is necessary to configure the motor rotation according to the set parameters, which is realized through GPIO output high or low levels in main function, as shown in Figure 4-3. After that, enable interrupts, and commence calculating and updating the duty cycle.

```
//main function
int main(void)
{
    SYSCFG_DL_init();
    switch (run_direction)
            {
                case 1: // UP
                    DL_GPIO_clearPins(GPIO_GRP_0_PORT, GPIO_GRP_0_PIN_0_PIN);
                    DL_GPIO_setPins(GPIO_GRP_0_PORT, GPIO_GRP_1_PIN_1_PIN);
                    DL_TimerA_setCaptureCompareValue(PWM_0_INST, 500, DL_TIMER_CC_0_INDEX);
                break;
                case 0: // DOWM
                    DL_GPIO_setPins(GPIO_GRP_0_PORT, GPIO_GRP_0_PIN_0_PIN);
                    DL_GPIO_clearPins(GPIO_GRP_0_PORT, GPIO_GRP_1_PIN_1_PIN);
                    DL_TimerA_setCaptureCompareValue(PWM_0_INST, 0, DL_TIMER_CC_0_INDEX);
                break;
            }
    NVIC_EnableIRQ(TIMER_0_INST_INT_IRQN);
}
```

Figure 4-3. Main Function

4.3 Timer Interrupt

The timer is configured in countdown mode, triggering an interrupt when it reaches zero. The counting period is set to 50ms, meaning the duty cycle is updated every 50ms. Within this interrupt function, the current reference value is calculated first using the speed loop controller, and current is then clamped. Subsequently, the duty cycle is determined using the current loop controller, meanwhile considering the dead-time. Finally, the duty cycle information is updated based on the actual direction of the motor.

```
void TIMER_0_INST_IRQHandler(void)
{

    switch (DL_TimerG_getPendingInterrupt(TIMER_0_INST))
    {
    case DL_TIMER_IIDX_ZERO:
            //speed loop
            target_current=speed_PID_realize(PIDSPEED_KP,PIDSPEED_KI,PIDSPEED_KD);
            //current loop
            target_current = (target_current > current_MAX) ? current_MAX : target_current; // limit max current
            target_current = (target_current < 0) ? 0 : target_current;   // limit min current
            switch (DL_TimerG_getQEIDirection(QEI_0_INST))
                      {
                            case 1: // UP
                                CC_value=500-current_PID_realize(target_current, PIDCURRENT_KP,PIDCURRENT_KI,PIDCURRENT_KD);
                            break;
                            case 0: // DOWM
                                CC_value=current_PID_realize(target_current, PIDCURRENT_KP,PIDCURRENT_KI,PIDCURRENT_KD);
                            break;
                       }
            //limit D according to dead time
            CC_value = (CC_value > CC_value_MAX) ? CC_value_MAX : CC_value;
            CC_value = (CC_value < CC_value_MIN) ? CC_value_MIN : CC_value;
            //update duty cycle
            DL_TimerA_setCaptureCompareValue(PWM_0_INST, CC_value, DL_TIMER_CC_0_INDEX);
        break;
    default:
        break;
    }
}
```

Figure 4-4. Timer Interrupt

4.4 Closed Loop Controller

Current and speed closed loop controller are as shown in Figure 4-5 and Figure 4-6. In this solution, a positional PID algorithm is implemented. Consequently, in addition to constraining the output range, it is imperative to limit the integral value to avert integral saturation. Due to the limitations in ADC resolution, it is necessary to constrain the initial error value to prevent system oscillation caused by closed-loop dead zone. The sampling current of the ADC also needs to undergo averaging filtering to minimize the impact of noise on the sampling results and enhance the stability of the system.

The MSPM0C1104 features a configurable timer (TIMA) with built-in dead-time generation function, allowing for convenient generation of the required PWM waveform based on the PID calculation results.

```
float current_PID_realize(float target_current, float Kp,float Ki, float Kd)
 {
     float PID_out;
     float current_error=0;
     int SUM_ADC=0;
     //get current from ADC
     for(int i=0; i<30; i++)
     {
     DL_ADC12_startConversion(ADC12_0_INST);
     SUM_ADC += DL_ADC12_getMemResult(ADC12_0_INST, DL_ADC12_MEM_IDX_0);
     DL_ADC12_enableConversions(ADC12_0_INST);
     }
     //average filter
     PRE_ADC=SUM_ADC/30;
     //10 bit ADC number convert to real current value
     actual_current=3.3*2.5*PRE_ADC/1024;
     current_error = target_current-actual_current;
     //limiting closed-loop deadband
     current_error = (current_error/target_current <0.1 && current_error >-0.1) ? 0 : current_error;
     current_sum_error+=current_error;     // error integrate
     //Preventing integral saturation
     if (current_sum_error > 10)
         current_sum_error = 10;
     else if (current_sum_error < -10)
         current_sum_error = -10;
     //PID calculate
     PID_out=Kp*current_error+Ki*current_sum_error+Kd*(current_error-current_error_last);     // PID calculation
     current_error_last=current_error;     // Error propagationA
     return PID_out;
 }
```

Figure 4-5. Current PID

```
float speed_PID_realize(float Kp,float Ki, float Kd)
    {
      float PID_out=0;
      float speed_error=0;
      // get speed from encoder
              switch (DL_TimerG_getQEIDirection(QEI_0_INST))
                       {
                            case 1: // UP
                                   Capture_count =(QEI_0_INST->COUNTERREGS.CTR);
                                   distance=Capture_count-Last_count;
                                   distance = (distance <0) ? (0xFFFF+distance) : distance;
                                   actual_speed=(float)distance*60 / (ENCODER_RESOLUTION*RECORD_TIME);
                            break;
                            case 0: // DOWM
                                   Capture_count =(QEI_0_INST->COUNTERREGS.CTR);
                                   distance=Last_count-Capture_count;
                                   distance = (distance <0) ? (0xFFFF+distance) : distance;
                                   actual_speed=(float)distance*60 / (ENCODER_RESOLUTION*RECORD_TIME);
                            break;
                        }
                        Last_count=Capture_count;
        speed_error = target_speed-actual_speed;
       //limiting closed-loop deadband
        speed_error = (speed_error <3 && speed_error >-3) ? 0 : speed_error;
       // PID realize
        speed_sum_error+=speed_error; // error integrate
       //Preventing integral saturation
       if (speed_sum_error > 6000)
           speed_sum_error = 6000;
       else if (speed_sum_error < -6000)
           speed_sum_error = -6000;
       PID_out=Kp*speed_error+Ki*speed_sum_error+Kd*(speed_error-speed_error_last);    // PID calculation
       speed_error_last=speed_error;    // Error propagation
       return PID_out;
    }
```

Figure 4-6. Speed PID

5 Evaluation

To evaluate the solution, the following hardware elements are required:

- A computer with Windows® 7 or later, and .NET Framework 4.5
- 12V DC power supply
- XDS110 debug probe
- Connector for driver board and debugger communication
- USB for PC and driver board communication
- Brushed DC motor with encoder. (ATK-JGB37-520E, the spec is as shown in Table 5-1)

Table 5-1. JGB37-520E Motor Parameters

Item	Value
Rated voltage	DC 12V
Current	0.2-0.5A
Idle speed	333RPM (peak value)
Gear Ratio	30:1

- Motor control board is shown as Figure 5-1.

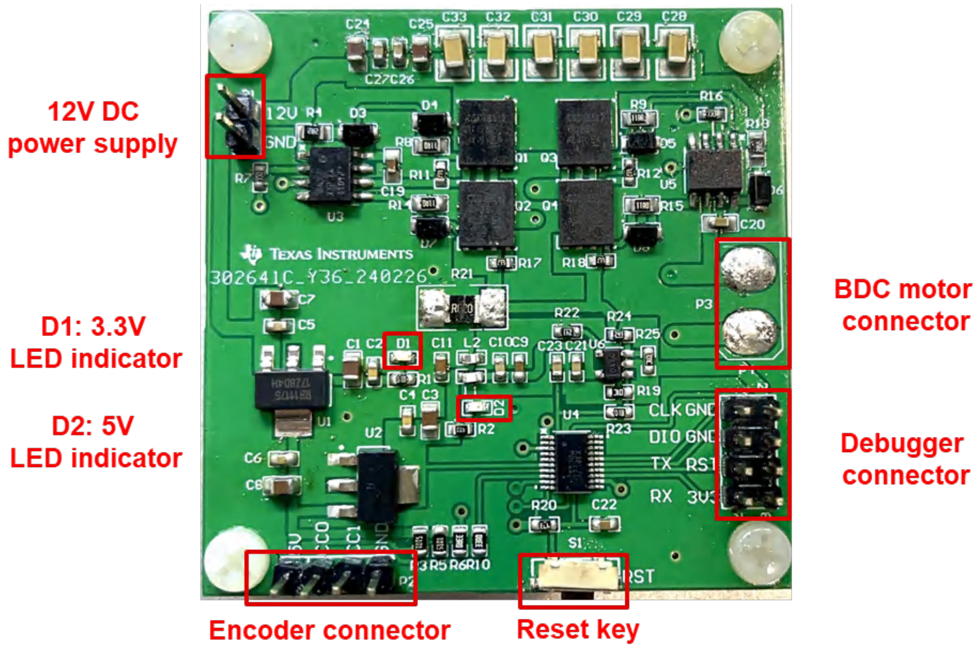

Figure 5-1. Motor Control Board

- Example code is shown in Section 4.

Once the necessary hardware and code are prepared, the hardware connection can be made according to the above description. After that, the sample code can be burned into the chip via the Code Composer Studio™ (CCS) software, and the evaluation of the solution can be carried out. The paper provides an economically flexible driving solution for brushed DC motors based on TI's MSPM0C1104, significantly reducing system costs while ensuring overall performance. Customers can flexibly develop based on this solution according to their actual needs.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265