NuMicro® N76E003
Brushless DC Motor Control
User Manual

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 INTRODUCTION

The close-loop controlled Brushless DC Motor reference design is implemented by the NuMicro® N76E003 microcontroller and NuMicro® NCT3605 gate driver.

The Brushless DC Motor reference design is equipped with the N76E003 microcontroller and NCT3605 gate driver. The N76E003 generates the 3 phase duty signals (U, V, and W) to the gate driver via 6 channels PWM function. The gate driver controls the rotation of the MOS to drive motor. Because the system is close-loop controlled, once the load changed, the motor speed will be calibrated automatically by target speed and real speed. The target motor speed is determined by sensing the voltage of variable resistor through ADC function, and the real motor speed is determined by feedback signals from the hall sensor inside the motor.

1.1 N76E003 Features

- 1T 8051-based N76E003, running up to 16 MHz
- Supports 2.4V to 5.5V operating voltage without LDO
- Built-in 18 Kbytes Flash (shared with APROM and LDROM; the max LDROM size can be configured as 4 Kbytes), 256 Bytes RAM, and 768 Bytes XRAM.
- Supports 4 sets of 16-bit timers, 2 sets of UART, 1 set of SPI, 1set of I^2C, and 8 sets of 12-bits ADC
- Supports Watchdog timer and wake-up timer function
- Supports 3 sets of complementary PWM / 6 sets of single channel PWM

1.2 BLDC Motor Control Features

- NuMicro® 1T 8051 microcontroller, running up to 16 MHz
- Supports 2.4V to 5.5V operating voltage without LDO
- Supports 3 sets of complementary PWM / 6 sets of single channel PWM for 3-phase motor controlled
- Supports 8 sets of 12-bits ADC for high resolution variable resistor voltage sensing
- The close-loop controlled architecture for calibrating motor speed automatically
- The NCT3605 supports protected function for preventing the upper/lower arm of NMOS from opening at the same time.

2 BLDC MOTOR CONTROL – HARDWARE INTRODUCTION

2.1 BLDC Motor Control Introduction

The BLDC Motor Control development kit includes three parts — a motor control mother board, a motor control daughter board, and a gate driver daughter board. Figure 2-1 shows the BLDC Motor Control development kit.

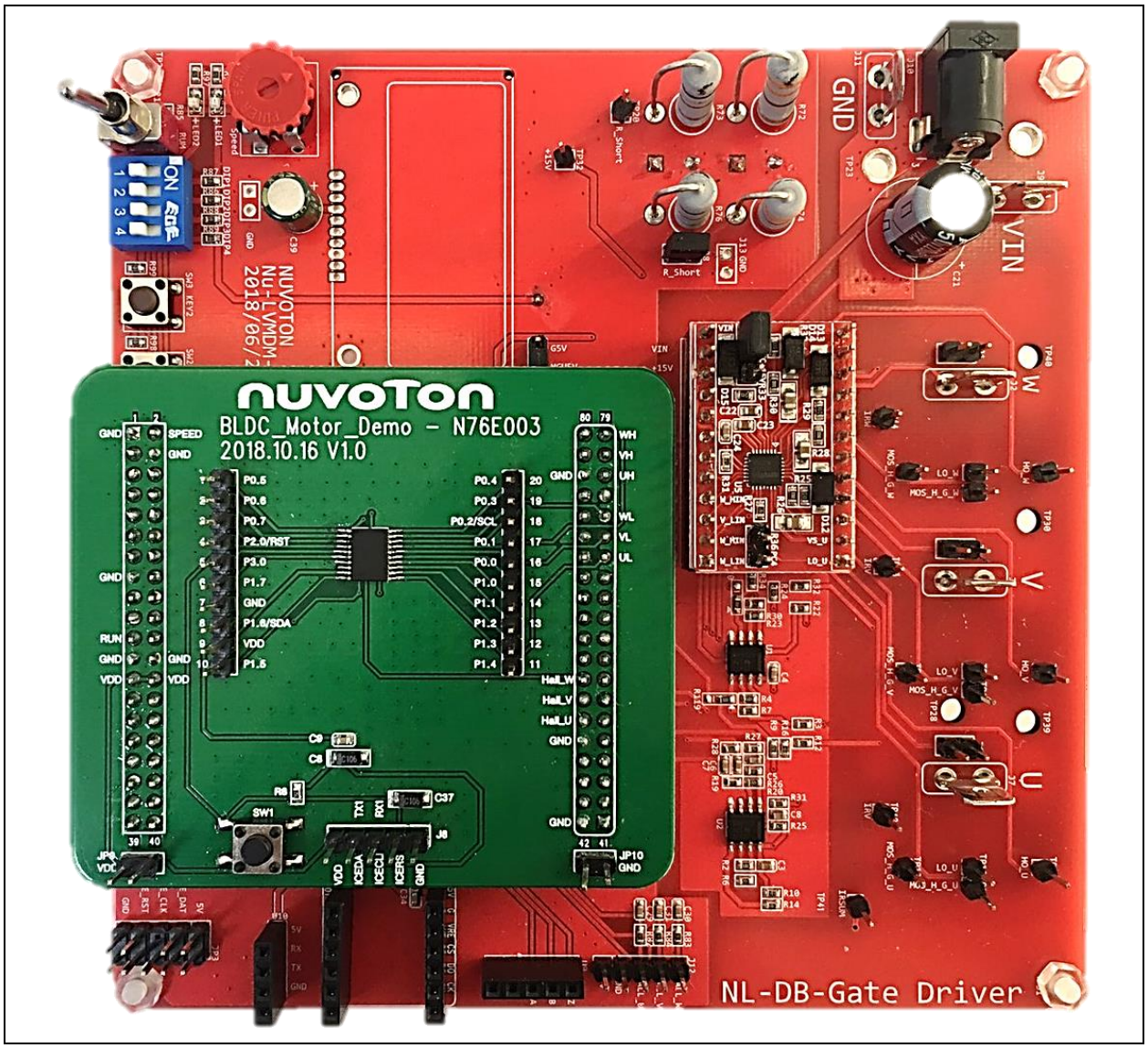

Figure 2-1 BLDC Motor Control Development Kit

2.2 Motor Control Mother Board

The motor control mother board includes several parts listed below.

- **Motor enable switch**: Enable / Disable motor
- **Variable Resistor**: The speed of motor rotation is controlled by sensing the voltage of variable resistor.
- **MCU 5V power input connector**: The power inputs to MCU via this connector.
- **24V power input connector:** The 24V power inputs for all of the devices via this connector.
- **U, V, W 3-phase control signal output end**: The motor control signal output by gate driver and it needs to connect to corresponding phase of motor
- **Hall signal feedback connector**: The hall feedback signal from motor inputs to MCU via this connector.
- **Motor control daughter board connector**: For connecting the motor control daughter board
- **Gate driver power input connector:** The power inputs to gate driver via this connector.

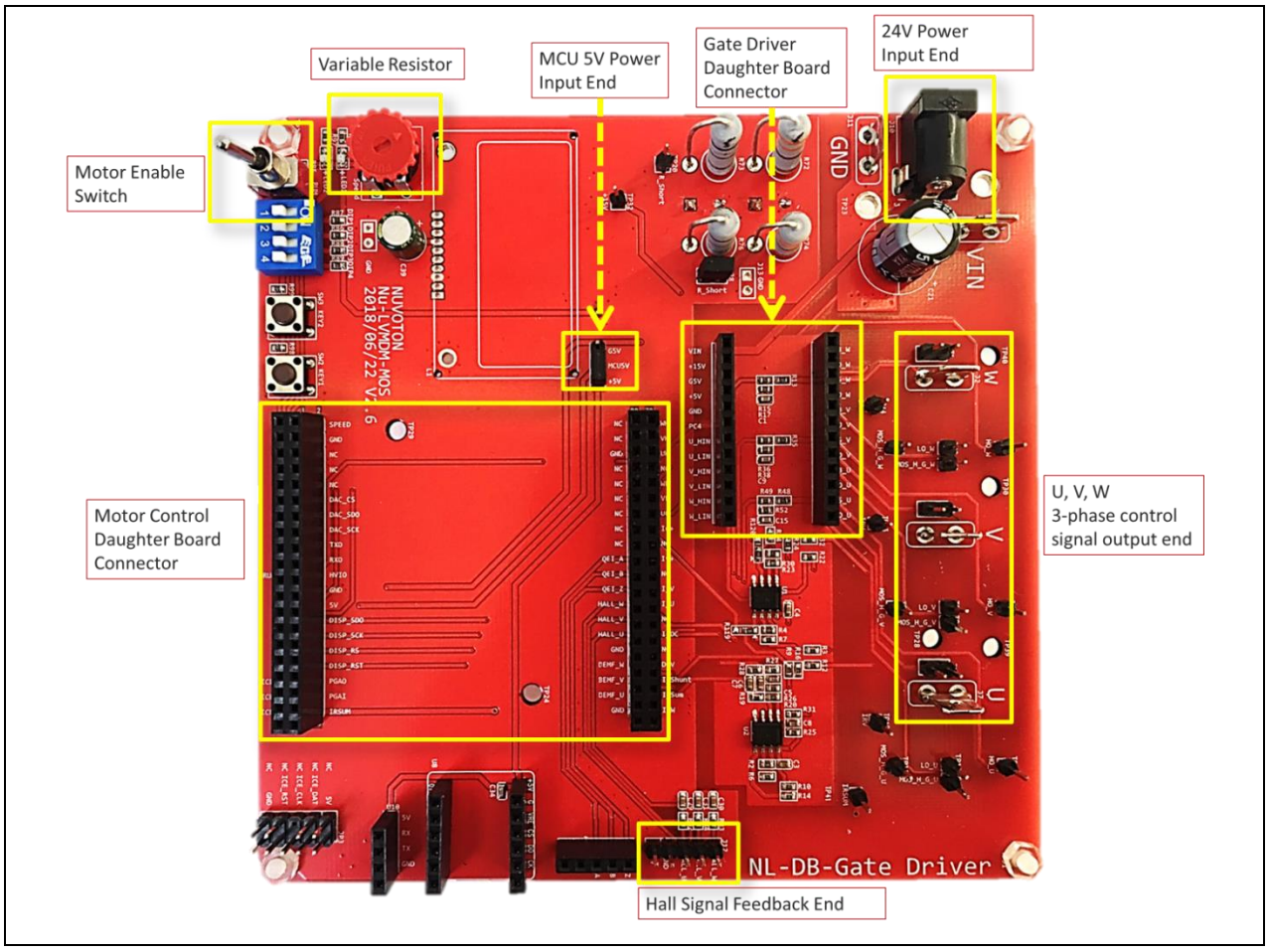

Figure 2-2 Motor Control Mother Board

2.3 Motor Control Daughter Board

The motor control daughter board includes several parts listed below.

- **Microcontroller N76E003**: The main chip for motor system control.
- **Reset button**: Pressing this button will reset the N76E003.
- **Program and debug interface**: Program and debug via this interface.

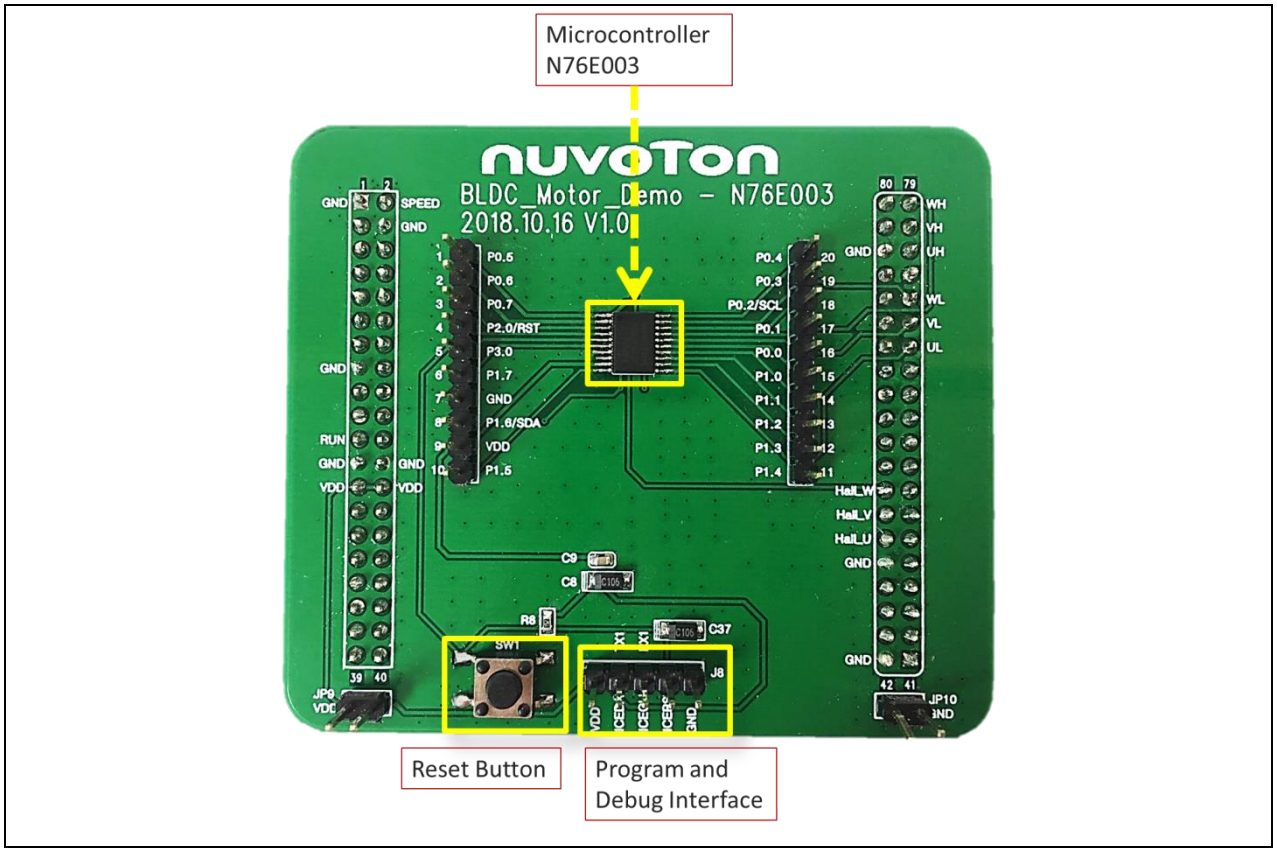

Figure 2-3 Motor Control Daughter Board

2.4 Gate Driver Daughter Board

The gate driver daughter board includes several parts listed below.

- **NCT3605 Gate Driver**: Receives the PWM signal from the N76E003 and controls the MOS for driving the motor rotation.
- **Gate driver power input connector:** The power inputs to gate driver via this connector.

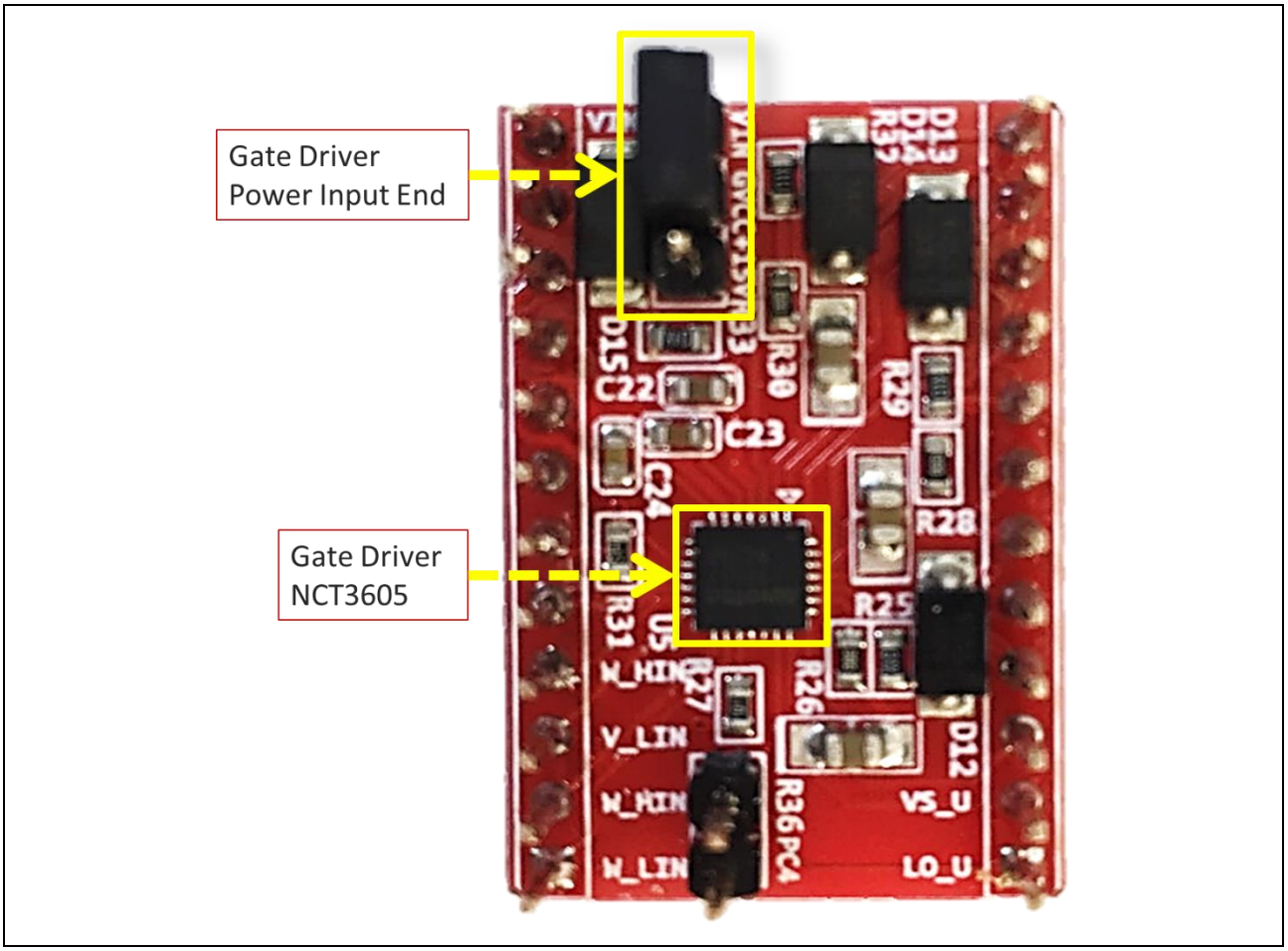

Figure 2-4 Gate Driver Daughter Board

2.5 Motor Control Mother Board Schematic

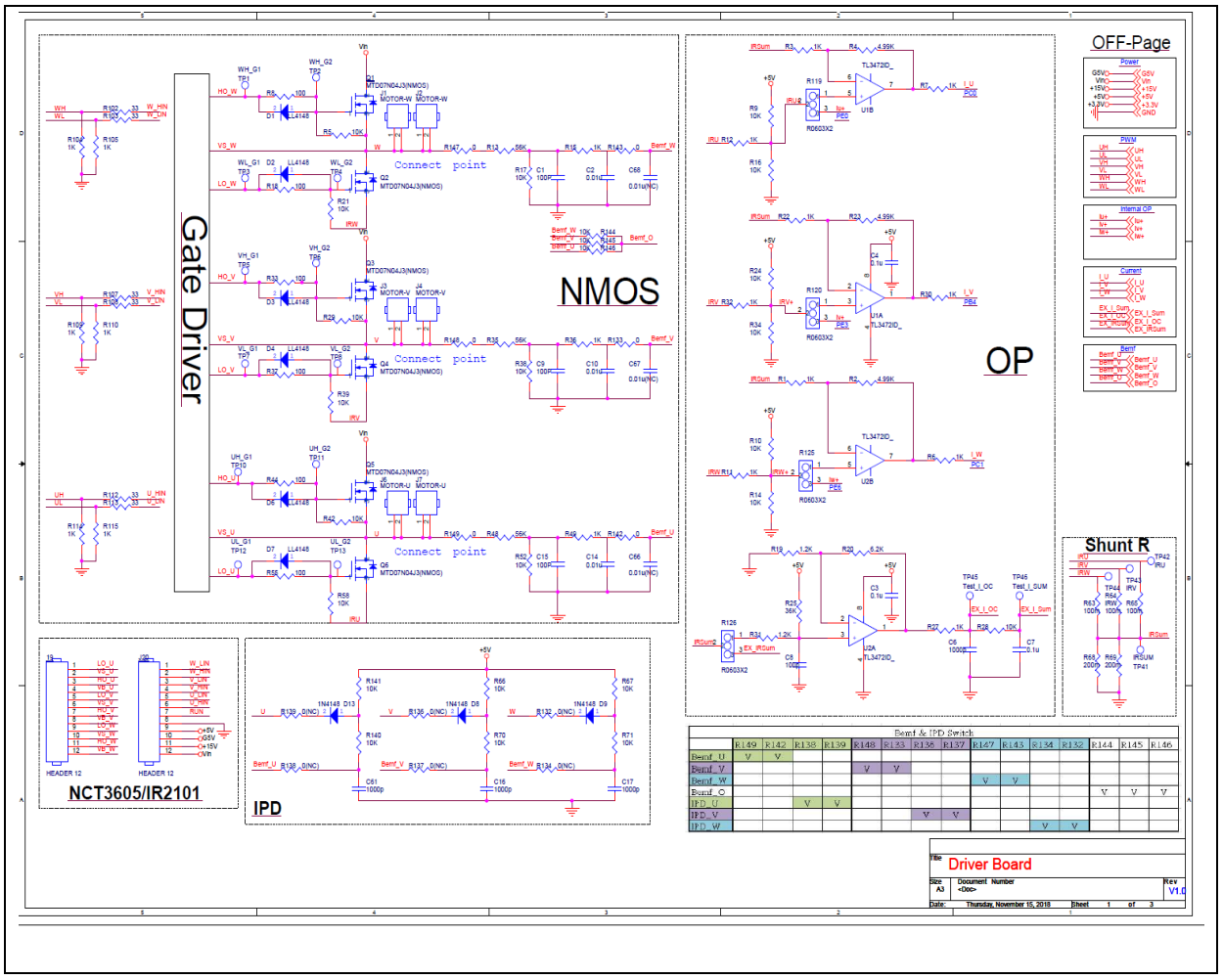

Figure 2-5 Motor Control Mother Board - Driver Board

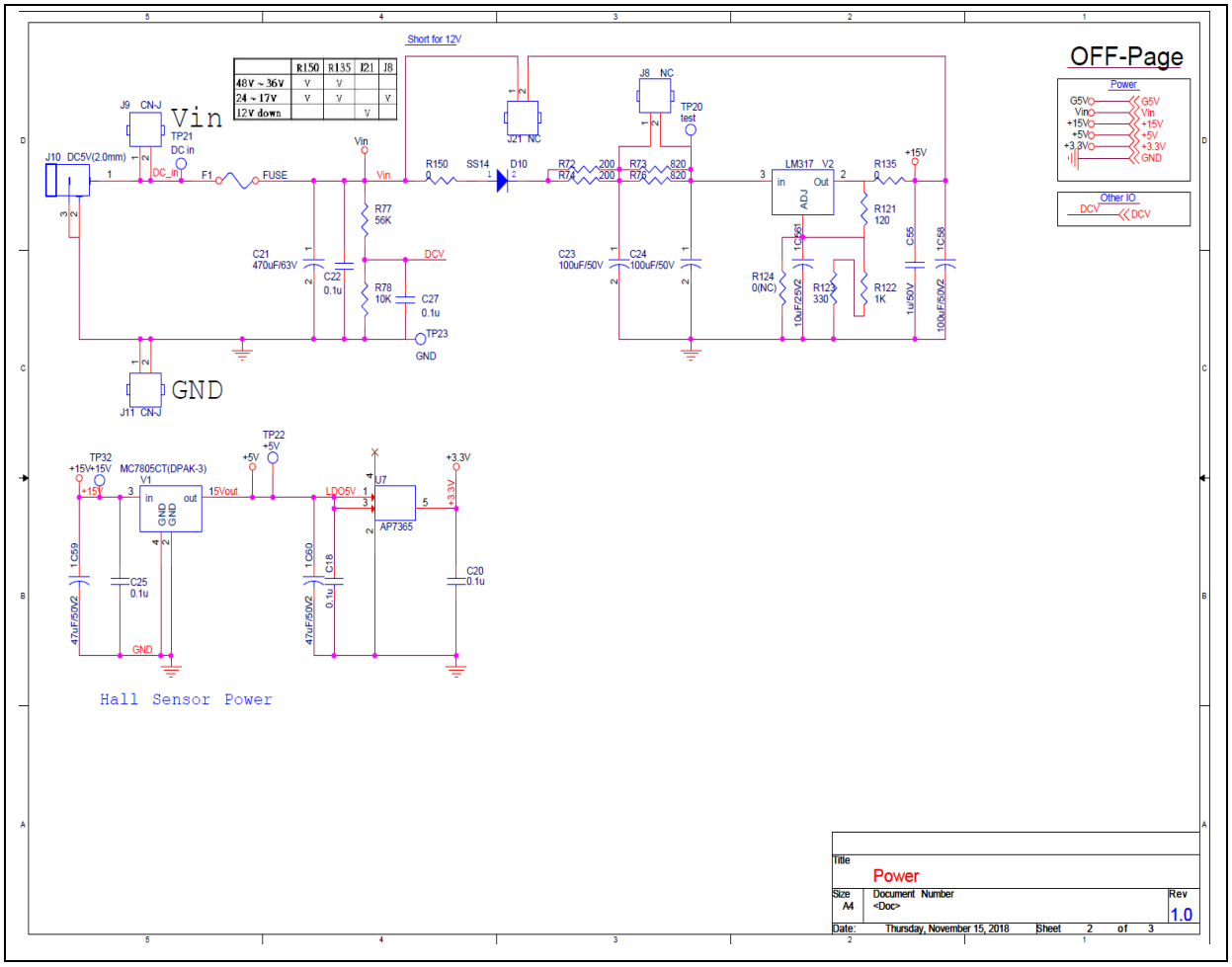

Figure 2-6 Motor Control Mother Board - Power

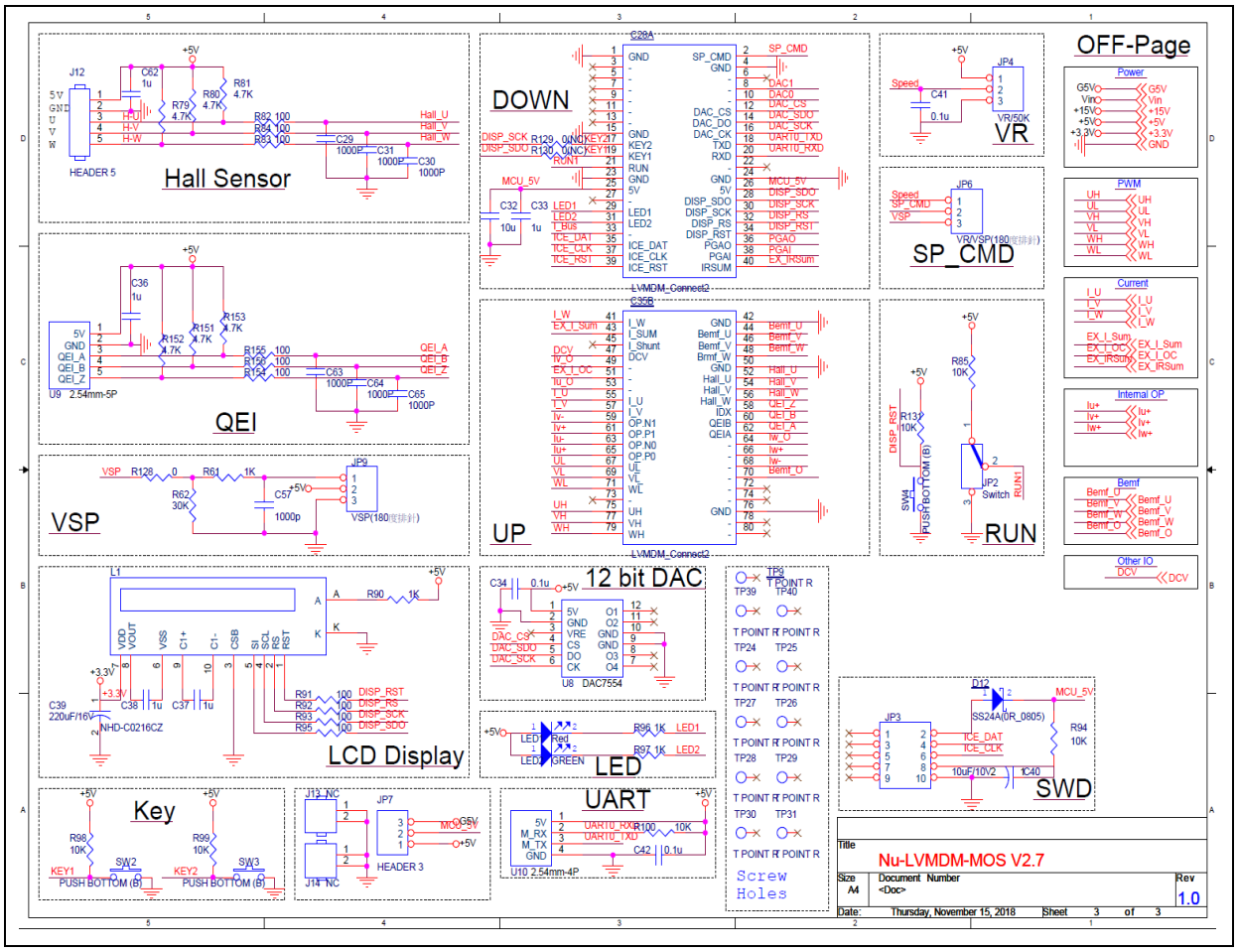

Figure 2-7 Motor Control Mother Board – Device

2.6 Motor Control Daughter Board Schematic

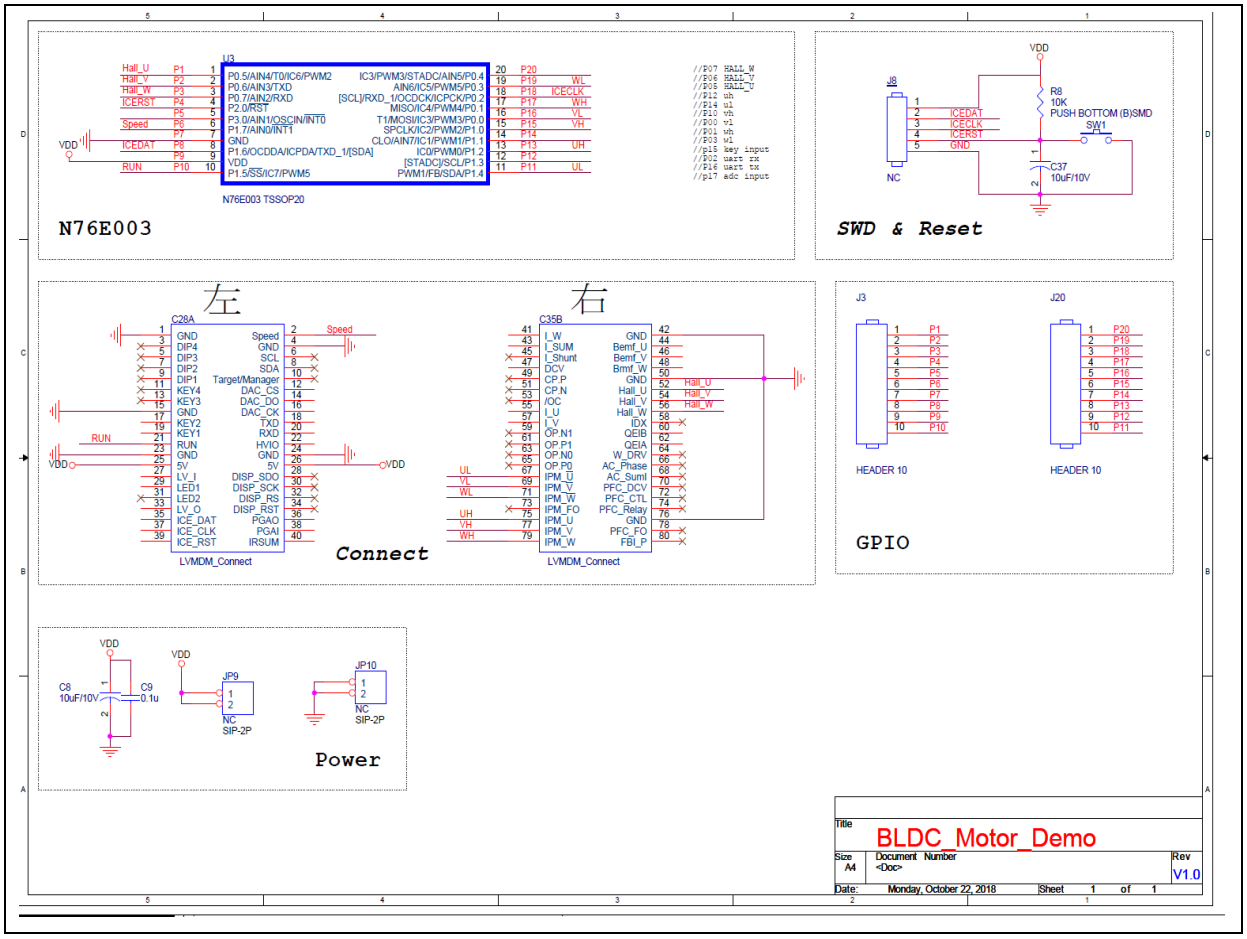

Figure 2-8 Motor Control Daughter Board Schematic

2.7 Gate Driver Daughter Board Schematic

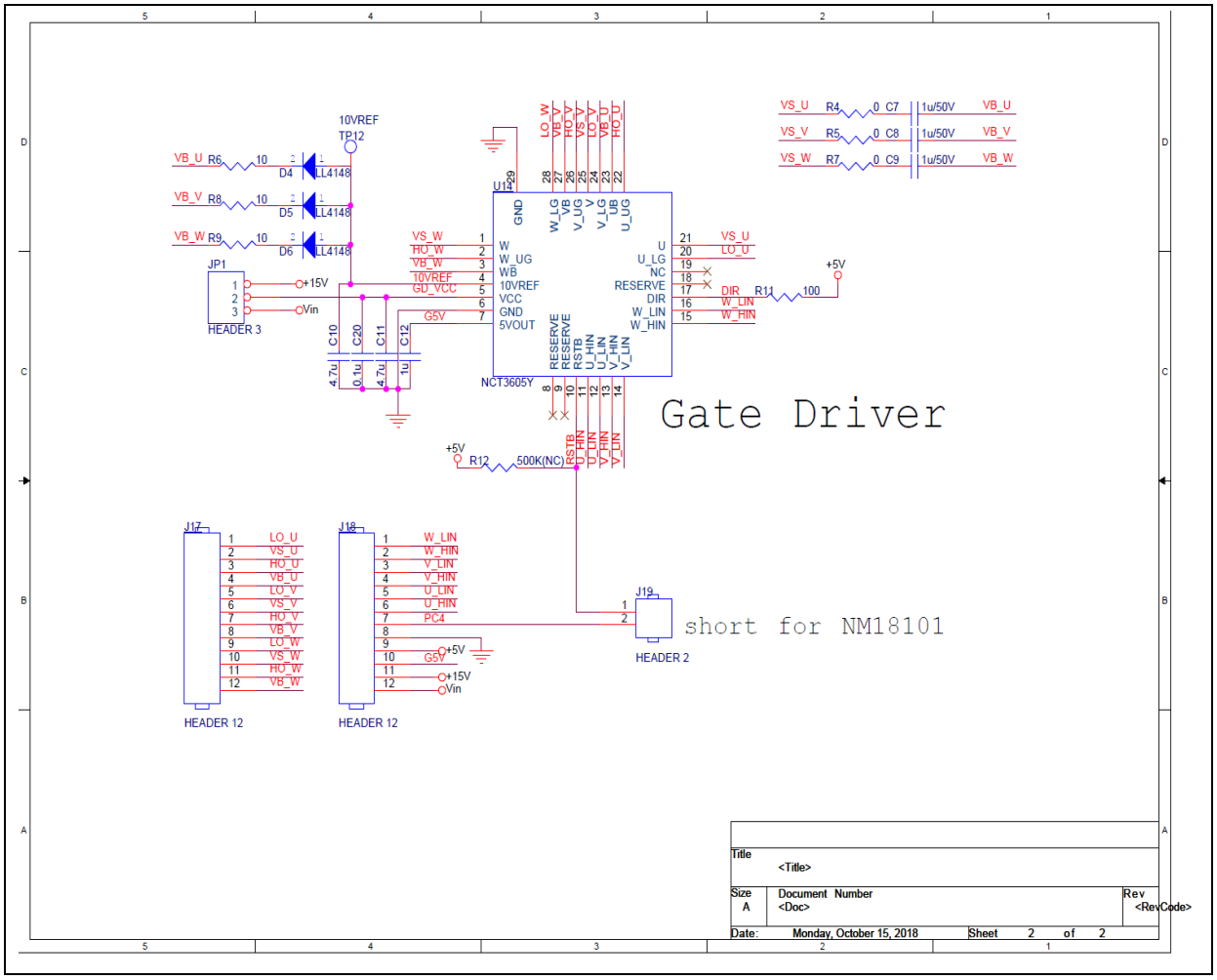

Figure 2-9 Gate Driver Daughter Board Schematic

3 BLDC MOTOR CONTROLL FIRMWARE

The Brushless DC Motor control solution generates the 3 phase duty signals (U, V, and W) to the gate driver via 6 channels PWM function. The gate driver controls the MOS to drive motor rotation. Because the system is close-loop controlled, once the load changed, the motor speed will calibrate automatically by target speed and real speed. The target motor speed is determined by sensing the voltage of variable resistor through ADC function, and the real motor speed is determined by feedback signals from the hall sensor inside the motor.

3.1 Main Process

```
/*---------------------------------------------------------------------------------------------------------*/
/* MAIN function                                                                                           */
/*---------------------------------------------------------------------------------------------------------*/
void main(void)
{
    // Initial GPIO
    InitGPIO();
#ifdef UART1_DEBUG
    // Initial UART1 for Debug
    InitialUART1_Timer3(115200);
#endif
    // Initial PWM for motor 3 phase signal
    InitPWM();
    // Initial Timer 0 for interrupt per 10 ms
    InitTimer0();
    // Initial Timer 2 for motor speed capture
    InitTimer2forCapture();

    // Reset timer and determine the hall state
    g_u8old_state=0xff;
    g_u8timer_count=0;
    hall_check((P0&0xE0));
    //Enable interrupts
    set_EA;
    //Clear and trigger ADC
    clr_ADCF;
    set_ADCS;

    //-------- PWM start run--------------
    set_LOAD;
    set_PWMRUN;

    while(1)
    {
        //Get Motor realtime speed by ADC
        if(ADCF == 1)
        {
            GetTargetSpeed();
        }
        //If switch is off (P15 == 0) => stop the Motor
        if(P15 == 0)
        {
            clr_PWMRUN;
            PMEN=0xff;
            PMD=0x00;
            while(P15 == 0);

            //Initial pwm
            g_u16Duty_vlaue=0x1F3*MotorBoostDuty;
            PWM0H = HIBYTE(g_u16Duty_vlaue);
            PWM0L = LOBYTE(g_u16Duty_vlaue);
            g_u8old_state=0xff;
```

```
            hall_check((P0&0xE0));
            set_LOAD;
            g_u8timer_count=0;
            set_PWMRUN;
        }
        //Per 30ms will enter it
        else if(g_u8timer_count>=3)
        {
            // Calculate the different between target and current speed
            g_as16diff[0]=(unsigned long int)g_u16target_speed-((unsigned long int)30000000/((unsigned
long int)(g_u16CurrentSpeed)*32));
            if(g_as16diff[0] > 0x1194) g_as16diff[0] = 0x1194;      //1194 means 4500RPM
            if(g_as16diff[0] < (-0x1194)) g_as16diff[0] = (-0x1194);

            #ifdef UART1_DEBUG
            // Print the speed of motor
            g_u8UARTPeriodCtrl++;
            if(g_u8UARTPeriodCtrl >=11)     //Per 330ms send UART data
            {
                Send_Data_To_UART1(((g_u16target_speed - g_as16diff[0])/17));
                g_u8UARTPeriodCtrl = 0;
            }
            #endif

            // Use the 3 stage buffer for eliminating the overshoot
            if((ABS(g_as16diff[2] - g_as16diff[1])<85) && (ABS(g_as16diff[2] - g_as16diff[0])<85))
            {
                // If the diff is too less or too large, it needs calibrating more fast
                if(ABS(g_as16diff[2])>150)
                {
                    g_u16Duty_vlaue += ((g_as16diff[2]>0) ? 2 : (-2));
                }
                else
                {
                    g_u8Debounce_Cnt++;
                    if(g_u8Debounce_Cnt>=5)
                    {
                        g_u16Duty_vlaue += ((g_as16diff[2]>0) ? 1 : (-1));
                        g_u8Debounce_Cnt = 0;
                    }
                }
            }
            //shift buffer data
            g_as16diff[2] = g_as16diff[1];
            g_as16diff[1] = g_as16diff[0];

            //Set new PWM duty
            PWM0H = HIBYTE(g_u16Duty_vlaue);
            PWM0L = LOBYTE(g_u16Duty_vlaue);
            set_LOAD;

            g_u8timer_count=0;
        }

        //Change the Motor phase
        if(g_u8old_state!=g_u8hall_state)
        {
            g_u8old_state=g_u8hall_state;
            CW();
        }
    }
}
```

3.2 Initial Function

There are several initial functions at the main function including initial GPIO, PWM, UART, etc. The functions are listed as follows.

```
void InitPWM(void)
{
    PWM_GP_MODE_ENABLE;
    PWM_SYNCHRONIZED_MODE;
    PWM_CLOCK_FSYS;
    PWMPH = 0x01;
    PWMPL = 0xF3;
    /*************************************************************************
        PWM frequency = Fpwm/((PWMPH,PWMPL) + 1) <Fpwm = Fsys/PWM_CLOCK_DIV>
                                                                = (16MHz)/(0x1F3 + 1)
                                                                = 1KHz (1ms)
    *************************************************************************/
    g_u16Duty_vlaue=0x1F3*MotorBoostDuty;
    //initial pwm value
    PWM0H = HIBYTE(g_u16Duty_vlaue);
    PWM0L = LOBYTE(g_u16Duty_vlaue);
    PMEN=0xff;
    PMD=0x00;
}

void InitTimer2forCapture(void)
{
    //Initial the Timer2 for capture motor speed
    TIMER2_CAP0_Capture_Mode;
    IC6_P05_CAP0_RisingEdge_Capture;
    TIMER2_DIV_512;
    set_ECAP;       //Enable Capture interrupt
    set_TR2;        //Triger Timer2
    set_EA;
}

void InitGPIO(void)
{
    Set_All_GPIO_Quasi_Mode;
    P05_Input_Mode;
    P06_Input_Mode;
    P07_Input_Mode;
    P15_Input_Mode;
    P04_PushPull_Mode;
    P12_PushPull_Mode;
    P14_PushPull_Mode;
    P10_PushPull_Mode;
    P00_PushPull_Mode;
    P01_PushPull_Mode;
    P03_PushPull_Mode;
    PWM0_P12_OUTPUT_ENABLE;//uh
    PWM1_P14_OUTPUT_ENABLE;//ul
    PWM2_P10_OUTPUT_ENABLE;//vh
    PWM3_P00_OUTPUT_ENABLE;//vl
    PWM4_P01_OUTPUT_ENABLE;//wh
    PWM5_P03_OUTPUT_ENABLE;//wl
    Enable_ADC_AIN0; //P17 for adc

    PICON = 0xFC;   //PORT 0 INT //Pin int control
    PINEN = 0XE0;   //FALL
    PIPEN = 0XE0;   //RISE
    set_EPI;        // Enable pin interrupt
    set_EX0;        //Enable external interrupt

}

void InitTimer0(void)
{
    clr_T0M;            //T0M=0, Timer0 Clock = Fsys/12
```

```
    TMOD |= 0x01;       //Timer0 is 16-bit mode
    g_u8TH0_Tmp = HIBYTE(TIMER0_VALUE);//(65536-TH0_INIT)/256;
    g_u8TL0_Tmp = LOBYTE(TIMER0_VALUE);//(65536-TL0_INIT)%256;

    TH0 = g_u8TH0_Tmp;
    TL0 = g_u8TL0_Tmp;

    set_ET0;            //enable Timer0 interrupt
    set_TR0;            // Timer0 Run
}
```

3.3 Interrupt Service Function

There are three interrupts in the main process — the GPIO interrupt for capturing the hall signal to determine the motor phase, the Timer0 interrupt for adjusting the speed of motor rotation per 10 ms, and the Timer2 interrupt for capturing the time of Hall signal raising edge to determine the real motor speed.

```
//Using the pin interrupt for hall sensor
void PinInterrupt_ISR (void) interrupt 7
{
    hall_check(P0&0xe0);
    PIF = 0x00;             //clear interrupt flag
}

void Timer0_ISR (void) interrupt 1  //interrupt address is 0x000B
{
    TH0 = g_u8TH0_Tmp;
    TL0 = g_u8TL0_Tmp;
    //Timer will add the counter per 10ms
    g_u8timer_count++;
}

void Capture_ISR (void) interrupt 12
{
    //Clear capture0 interrupt flag
    clr_CAPF0;
    //Get the current speed
    g_u16CurrentSpeed = (C0L + (C0H<<8));

    clr_TF2;
}
```

3.4 Change Phase Function

When the BLDC motor changes the phase state, the hall signal will be changed at the same time. Once GPIO interrupt triggered by motor phase state is changed, the new phase state will be determined by these functions. Then, main function will switch the magnetic field direction to make the motor rotate continuously.

```
//Calculate the motor phase state
void hall_check(unsigned char input)
{
    switch(input)
    {
        case hall_sensor1:
        {
            g_u8hall_state=1;
            break;
        }
        case hall_sensor2:
        {
            g_u8hall_state=2;
            break;
        }
        case hall_sensor3:
        {
            g_u8hall_state=3;
            break;
        }
        case hall_sensor4:
        {
            g_u8hall_state=4;
            break;
        }
        case hall_sensor5:
        {
            g_u8hall_state=5;
            break;
        }
        case hall_sensor6:
        {
            g_u8hall_state=6;
            break;
        }
    }
}

//Change the Motor phase
void CW(void)
{
    switch(g_u8hall_state)
    {
        case 1:
        {
            //hall sensor1
```

```
            PMEN=0xfe;//uh
            PMD=0x20;//wl
            break;
        }
        case 2:
        {
            //hall sensor2
            PMEN=0xfb;//vh
            PMD=0x20;//wl
            break;
        }
        case 3:
        {
            //hall sensor3
            PMEN=0xfb;//vh
            PMD=0x02;//ul
            break;
        }
        case 4:
        {
            //hall sensor4
            PMEN=0xef;//wh
            PMD=0x02;//ul
            break;
        }
        case 5:
        {
            //hall sensor5
            PMEN=0xef;//wh
            PMD=0x08;//vl
            break;
        }
        case 6:
        {
            //hall sensor6
            PMEN=0xfe;//uh
            PMD=0x08;//vl
            break;
        }
    }
}
```

4 REVISION HISTORY

Date	Revision	Description
2018.11. 22	1.00	Initial version
2018.11. 30	1.01	Modify the grammar of whole document.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.