AN10661

Brushless DC motor control using the LPC2141

Rev. 01 — 17 October 2007 Application note

Document information

Info	Content
Keywords	LPC2148, ARM7, Brushless DC motor control
Abstract	This application note demonstrates the use of a low cost ARM7 based LPC2141 microcontroller for sensored brushless DC motor control.

founded by Philips

Revision history

Rev	Date	Description
01	20071017	Initial version.

Contact information

For additional information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

This application note demonstrates the use of a low cost NXP Semiconductors LPC2141 microcontroller for brushless DC motor control. It may be used as a starting point for motor control system designers using an NXP LPC2000 microcontroller.

The LPC2141 is based on a 16/32-bit ARM7 CPU combined with embedded high-speed flash memory. A superior performance as well as their tiny size, low power consumption and a blend of on-chip peripherals make these devices ideal for a wide range of applications. Various 32-bit timers, 10-bit ADC and PWM features through output match on all timers, make them particularly suitable for industrial control. Main reason to use the LPC2141 for this reference design (see Fig 1) is the on-chip USB interface, which is used to communicate with a PC GUI (Graphical User Interface) controlling the motor.

Besides the use of an LPC2141, the reference design in this application note shows a complete motor control system solution from NXP Semiconductors in terms of NXP Microcontroller – NXP MOSFET driver – NXP MOSFET.

Brushless DC (Direct Current) motors are most commonly used in easy to drive, variable speed and long life applications. They have become widespread and are available in all shapes and sizes from large-scale industrial models to small motors for light applications (such as 12 V BLDC motors).

Applications:

Air conditioners, electric pumps, fans, printers, robots, electric bikes, -doors, -windows, -sun roofs, -seats, mixers, food processors, blenders, vacuum cleaners, toothbrushes, razors, coffee grinders, etc.

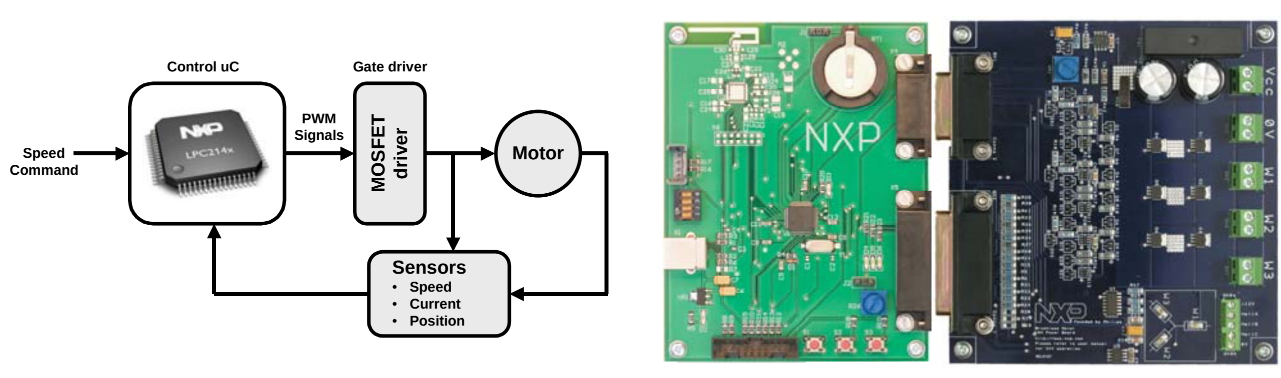


Fig 1. Controller (green) and Power (blue) demo boards for BLDC motor application

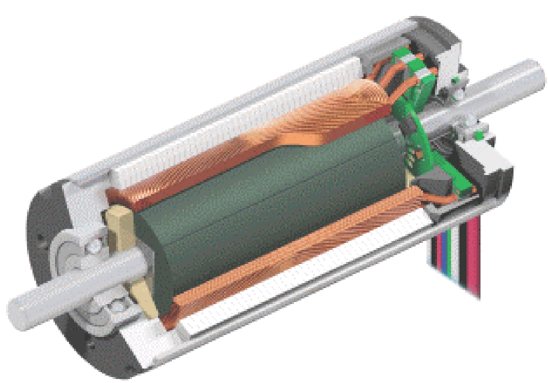
2. Brushless DC motor fundamentals

Brushless DC motors consist of a permanent magnet rotor with a three-phase stator winding. As the name implies, BLDC motors do not use brushes for commutation; instead, they are electronically commutated. Typically three Hall sensors (see Fig 2) are used to detect the rotor position and commutation is based on these sensor inputs.

Brushless DC (BLDC) motors are rapidly gaining popularity. They offer longer life and less maintenance than conventional brushed DC motors. Some other advantages over brushed DC motors and induction motors are: better speed versus torque characteristics, noiseless operation and higher speed ranges. And in addition, the ratio of torque delivered to the size of the motor is higher, making them useful in applications where space and weight are critical factors.

In a brushless DC motor, the electromagnets do not move; instead, the permanent magnets rotate and the three-phase stator windings remain static (see Fig 2). This gets around the problem of how to transfer current to a moving rotor. In order to do this, the brush-commutator assembly is replaced by an intelligent electronic “controller”. The controller performs the same power distribution as found in a brushed DC motor, but is using a solid-state circuit rather than a commutator/brush system.

The speed and torque of the motor depend on the strength of the magnetic field generated by the energized windings of the motor, which depend on the current through them. Therefore adjusting the rotor voltage (and current) will change the motor speed.



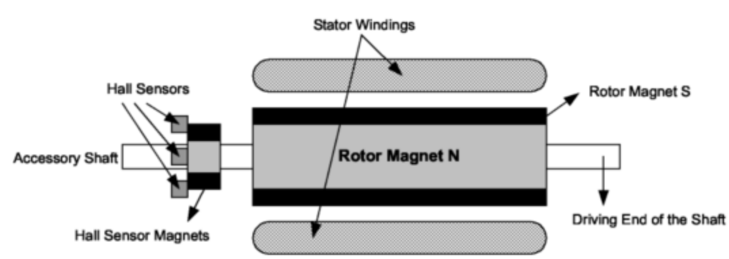


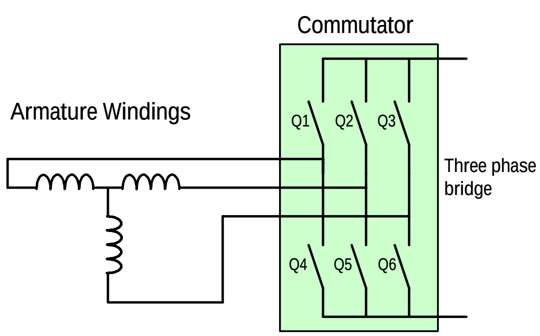
Fig 2. Brushless DC motor

3. How to control a brushless DC motor

3.1 Rotation

A BLDC motor is driven by voltage strokes coupled with the given rotor position. These voltage strokes must be properly applied to the active phases of the three-phase winding system so that the angle between the stator flux and the rotor flux is kept close to 90° to get the maximum generated torque. Therefore, the controller needs some means of determining the rotor's orientation/position (relative to the stator coils.)

In our design we use Hall effect sensors (some use a rotary encoder, others sense the back EMF in the un-driven coils) to directly measure the rotor's position. Each sensor element outputs a high level for 180° of an electrical rotation, and a low level for the other 180°. The three sensors have a 60° relative offset from each other. This divides a rotation into six phases (3-bit code). Fig 3 and Fig 4 show the relationship between the Hall sensor input code and the required active motor windings.



Hall Sensor code	Phase #	Active drive	
101	1	Q1 (PWM1)	Q6 (PWM6)
100	2	Q1 (PWM1)	Q5 (PWM5)
110	3	Q3 (PWM3)	Q5 (PWM5)
010	4	Q3 (PWM3)	Q4 (PWM4)
011	5	Q2 (PWM2)	Q4 (PWM4)
001	6	Q2 (PWM2)	Q6 (PWM6)

Fig 3. Three phase bridge and sensor input by active switch table

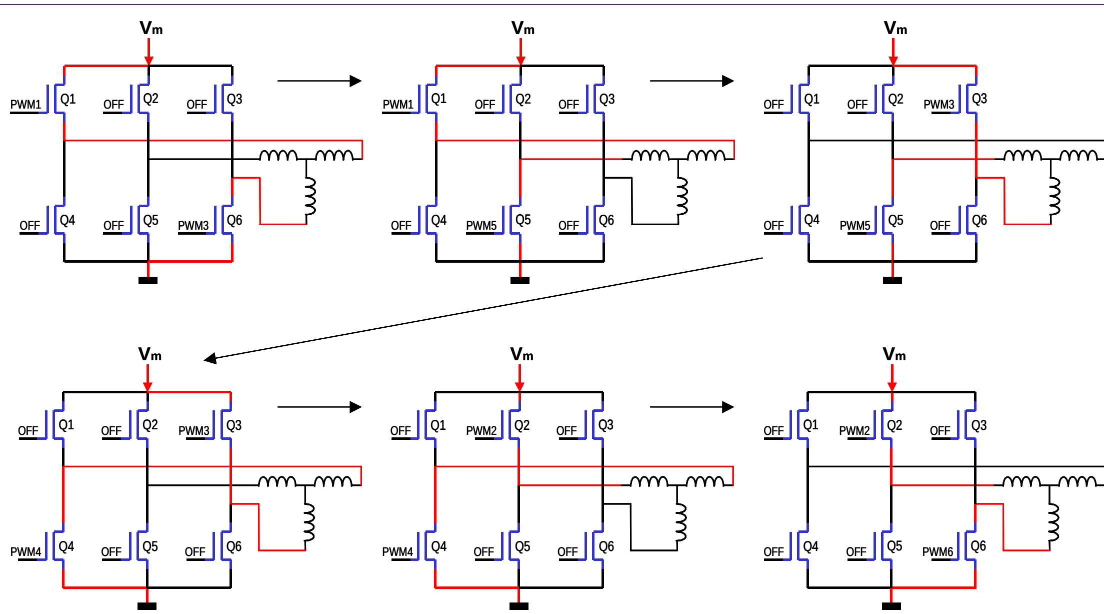


Fig 4. Motor rotation Q1 to Q6 switch sequence

3.2 Speed control

By simply varying the voltage across the motor, one can control the speed of the motor. When using PWM outputs to control the six switches of the three-phase bridge, variation of the motor voltage can be achieved easily by changing the duty cycle of the PWM signal (see Fig 5).

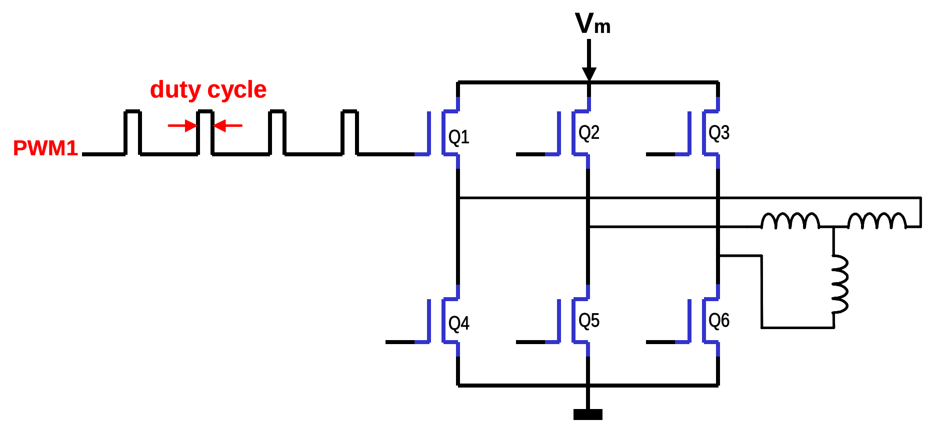


Fig 5. PWM speed control

3.3 Motor feedback

3.3.1 Current sense

Low cost motor current measuring can be implemented (like in this application note) using a current sensing resistor between the switching MOSFETs and ground (see also block diagram Fig 6). The small voltage appearing across the current sense resistor is filtered and amplified, before being fed to an ADC input of the microcontroller.

Like in this application note measuring the motor current is often used as a safety. In case the motor is in a stalled position, the current will increase dramatically. Due to this exceptional increase in current, the ADC values will reach a current limit level that will cause the system to shut down, avoiding any damages (switch into ‘coast’ mode).

3.3.2 RPM measurement

For closed loop speed control the ‘real’ motor speed must be known. By having the Hall sensor signals available at the LPC2141 microcontroller input pins, they can easily be “misused” for exact motor speed (RPM) measurement.

One possible way for example is to connect the Hall sensor outputs to external interrupt input pins of the microcontroller. This results in having an interrupt every 60° degrees of an electrical rotation. By simply counting the number of interrupts within a certain exact time (for example 1 second) it’s easy to calculate the exact motor speed.

Another possibility is to connect the sensor signals to Timer Capture inputs of the microcontroller. This way the exact time is measured between every rotation phase change.

4. Application setup

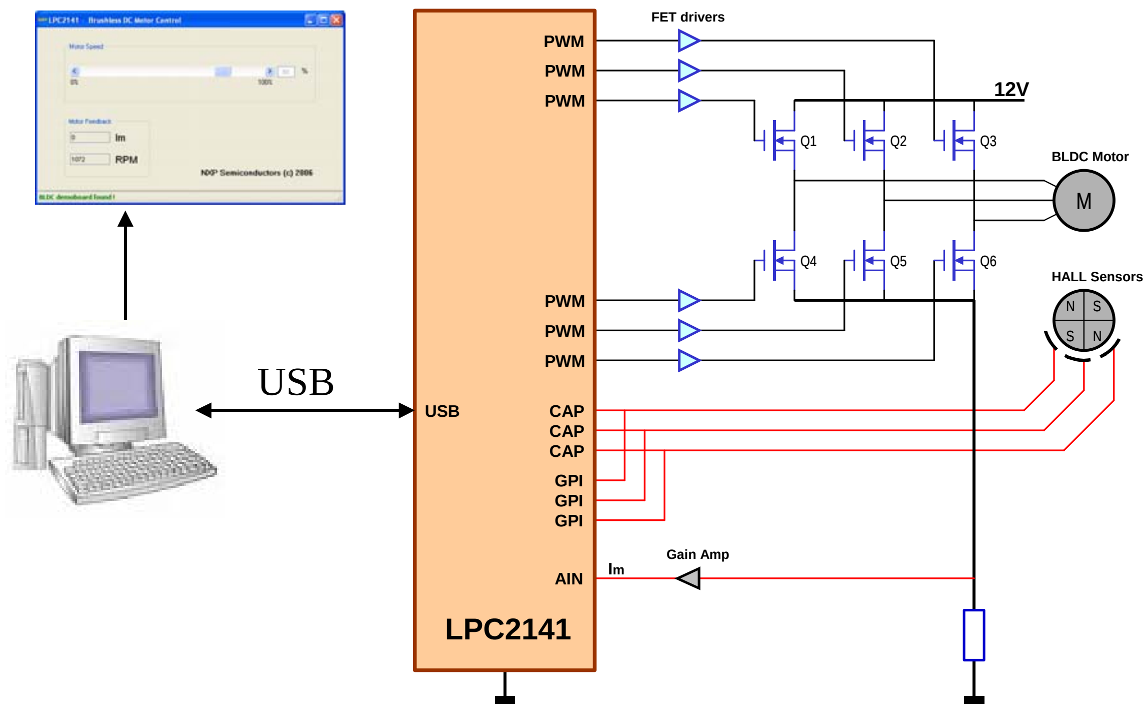


Fig 6. Block diagram

4.1 Using the LPC2141

For this application note the LPC2141 is used (see Fig 6), mainly because of its six-channel PWM timer and the on-chip USB interface. Available in an LQFP64 package it is a small and cheap member of NXP's ARM7 based LPC2000 family. It offers high speed (60 MHz) 32-bit CPU performance, 8 kB of on-chip static RAM and 32 kB of on-chip flash program memory. For larger memory - or additional specific peripheral (CAN, Ethernet, etc.) requirements, a broad selection of (compatible) NXP - LPC2000 family members are available. To give an impression of the possibilities this microcontroller offers, for this application note:

– CPU load is less than 5 %, used code size is 6 kB (including USB communication)
– Unused peripherals: UART, I2C, SPI/SSP, RTC, 2 x Timer and 5 x A/D input
– Over 30 unused GPIO pins available for user's application

4.2 Motor selection

For this application note a 120 W Maxon EC-40 motor is used. The 'no-load' speed is 5900 RPM at 24 V input. The maximum continuous current is 6 A.

4.3 MOSFET selection

The NXP Semiconductors PH20100S N-channel TrenchMOS logic level FET is used for this system. It is chosen in relation with the selected motor, which is supplied with 24 V.

For a 24 V - supplied motor, the MOSFET V_{DS} needs to be at least 40 V, while the drain current needs to be high enough to deal with the motor (starting) current. The latter is already reduced thanks to a soft-acceleration mechanism (in small steps up towards the required speed) implemented in software. The PH20100S can deal with a maximum drain current of 34.3 Amps and a peak current of 137 Amps and is available in an SMD SOT669 (LFPAK) package (see Fig 7).

4.4 MOSFET driver selection

MOSFET drivers are needed to raise the controller's output signal (driving the MOSFET) to the motor supply voltage level. In this application note we selected the PMD3001D and the PMGD400UN from NXP Semiconductors, as shown in Fig 7.

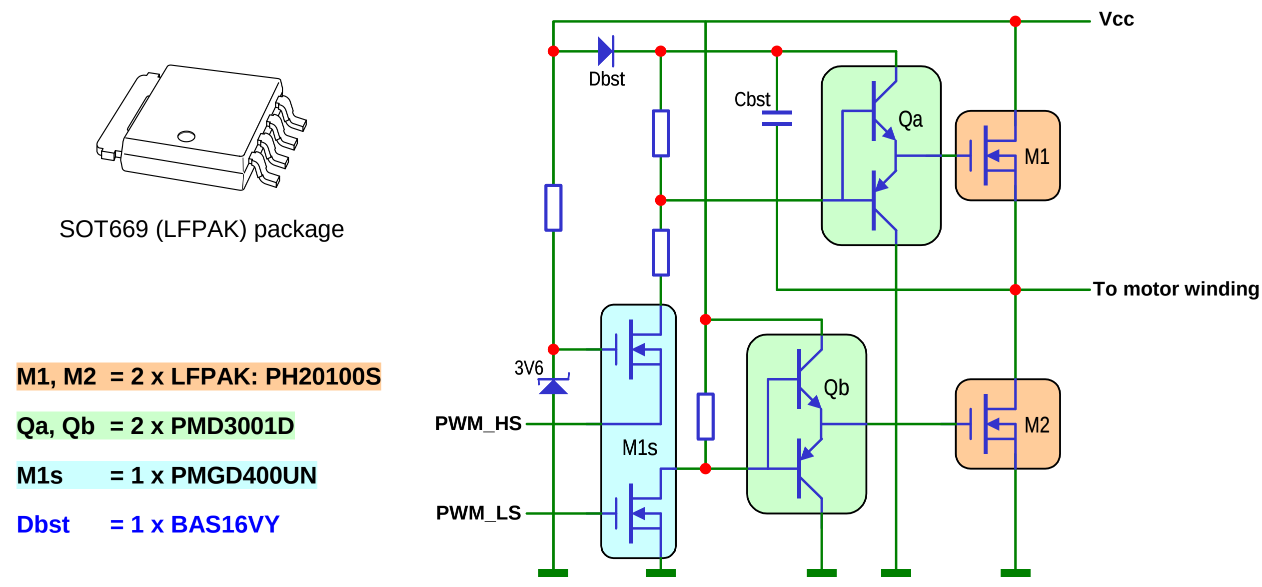


Fig 7. Simplified MOSFET – driver diagram for low and high side driver

4.5 Adjusting motor speed

The LPC2141 has an on-chip six-channel (32-bit) PWM timer, which makes it ideal for using it to control a three-phase bridge. Values for desired motor speed are received via the USB interface

5. Hardware schematics

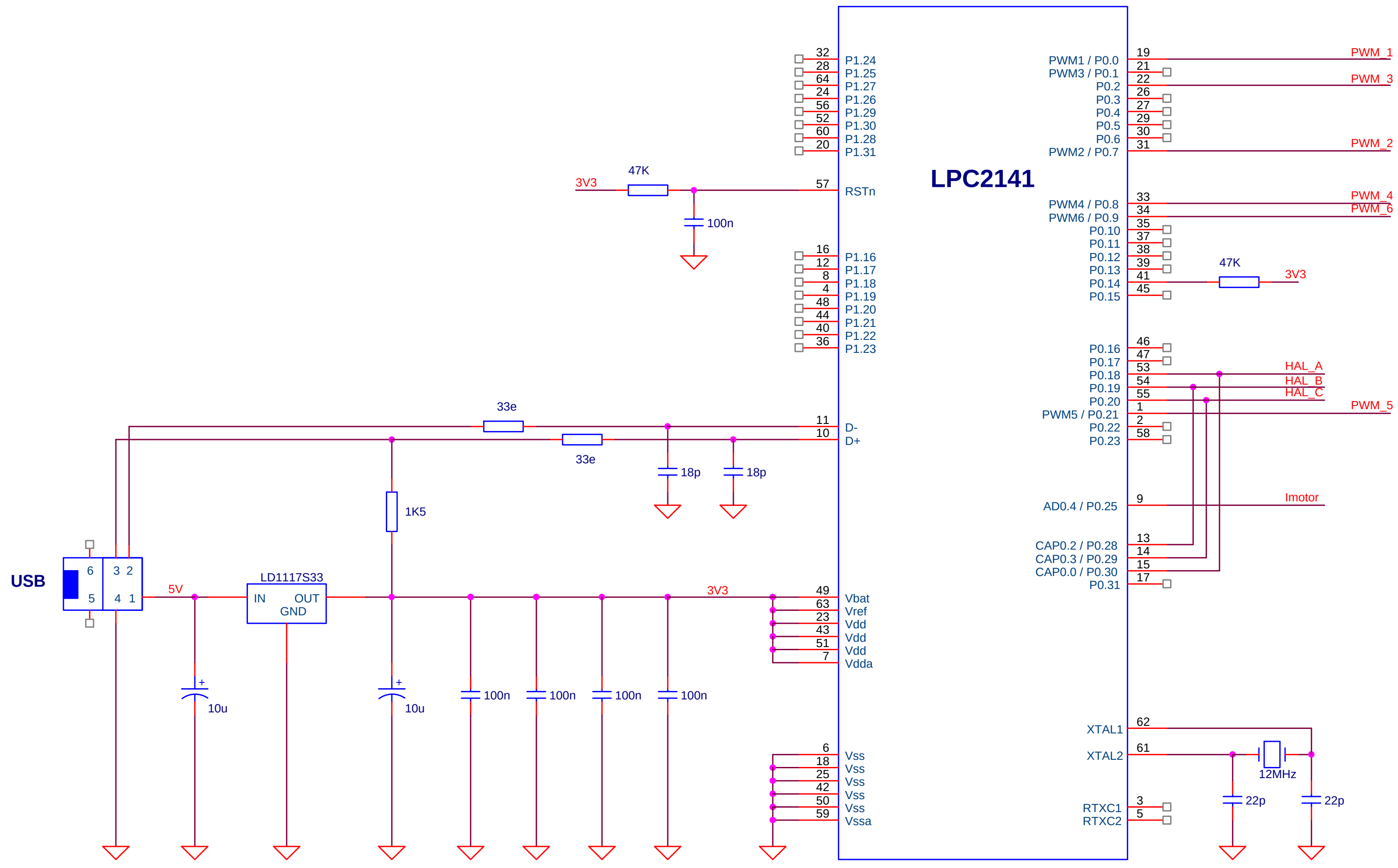


Fig 8. Hardware schematics – controller part

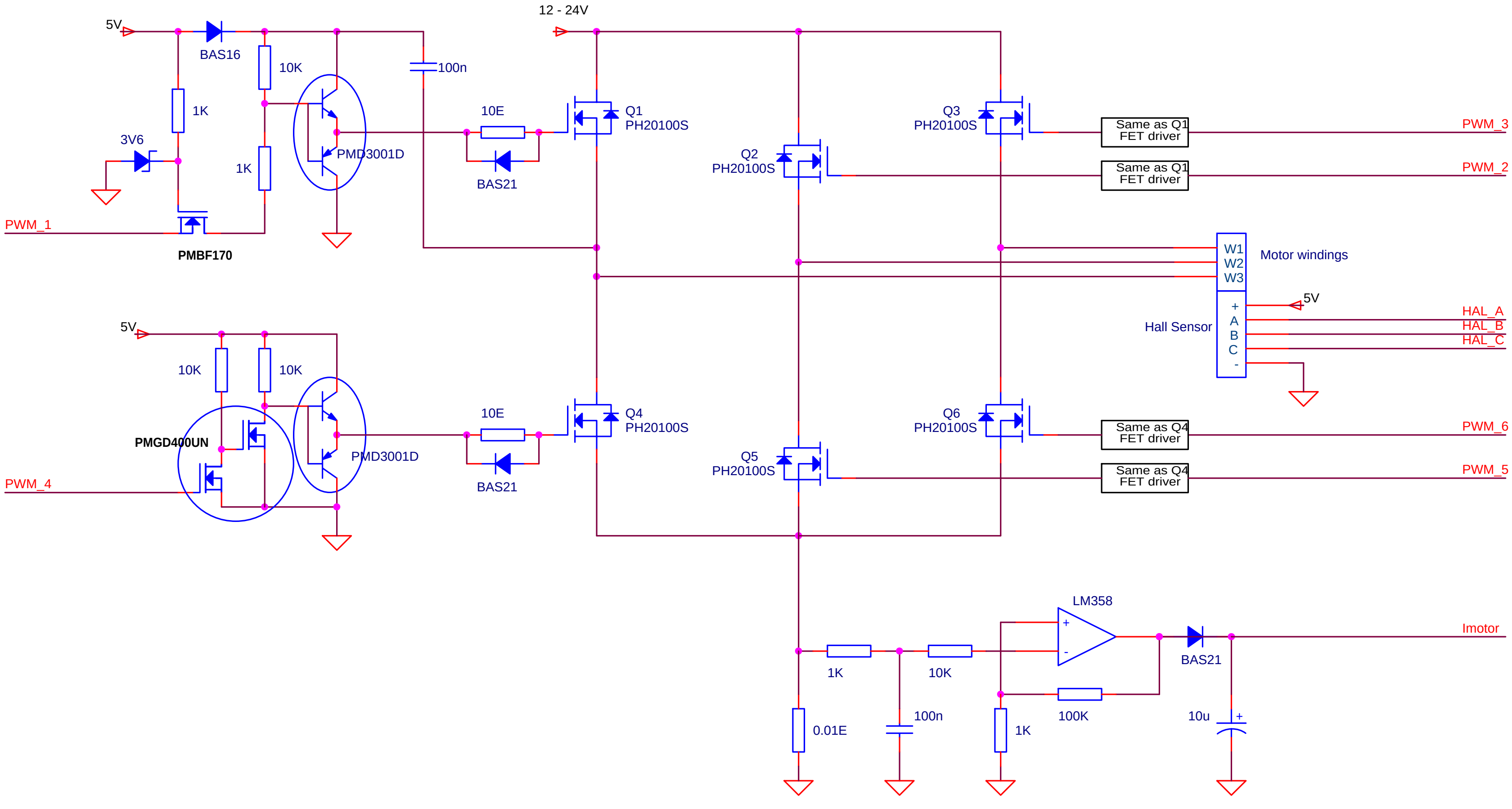


Fig 9. Hardware schematics – power / motor part

6. Software

The software for the complete demo contains three main parts: User Interface (GUI), USB driver and the Motor Control application code.

6.1 User interface

A Windows® user interface is available to control the BLDC demo (see Fig 10). The program is called “BLDC_USBGUI.EXE” and is developed in Microsoft Visual Basic 2005 Express, so it needs the Microsoft .NET framework installed at your PC. The program offers easy control of speed and readouts of motor current and RPM.

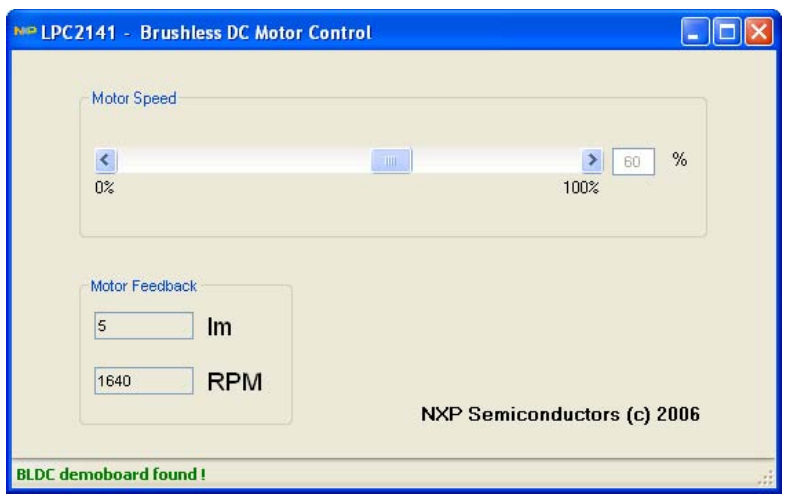


Fig 10. Windows user interface screen

6.2 USB device driver

For USB communication Keil’s LPC2148 USB HID (human interface device) software example is used. For more information please check website of Keil.

6.3 BLDC Motor Control code

The example software is written in C language and compiled using Keil’s uVision (ARM7 RealView, V3.0) free demo compiler. It performs following main tasks:

- USB interface for receiving desired speed, sending motor current and measure and send calculated RPM
- Read and ‘guard’ the motor current, using 10-bit ADC input
- Use Timer 1 to generate a system-interrupt every 10 milliseconds.
- Motor commutation by reading Hall sensors (using Timer 0 input capture pins), set the PWM Timer duty cycle for speed and drive Q1-Q6 MOSFET outputs for control of the three-phase bridge.

7. Source code listings

The motor control part consists of five modules (bldc.c – adc.c – pwm.c – hsensor.c timer1.c) and a header file (bldc.h), all listed below. The USB modules from Keil's HID example are not listed in this application note. For LPC2141 configuration the standard startup code from Keil was used and set as CCLK = PCLK = 60 MHz.

7.1 BLDC.C

```
#include <LPC214x.H>                           // LPC214x definitions
#include "bldc.h"

unsigned char actualSpeed  = 0;
unsigned char desiredSpeed = 0;
unsigned int  RPM,fRPM;

void GetInReport(unsigned char *rep)           // Host is asking for an InReport
{
    rep[0] = fRPM;                             // send measured motor speed (low byte)
    rep[1] = fRPM >> 8;                        // send measured motor speed (high byte)
    rep[2] = AD0DR4 >> 8;;                     // send potm value for debugging
}

void SetOutReport(unsigned char *rep)          // OutReport received from USB host
{
    if (rep[0] < 101)
        desiredSpeed = rep[0];                 // New desired speed value received
}

int main (void)
{
    ADC0_Init();                               // ADC0 Initialization
    T1_Init();                                 // 10 msec tick
    PWM_Init();                                // PWM Timer Initialization
    HES_Init();
    USB_Init();                                // USB Initialization
    USB_Connect(1);                            // USB Connect

    while (1)                                  // Loop forever
    {
        if (((AD0DR4 >> 8) & 0xFF) > MAX_Im)   // Check motor overcurrent
        {
            VICIntEnClr = 0xFFFFFFFF;          // disable all interrupts!
            PWMMR1 = 0;                        // Q1 off
            PWMMR2 = 0;                        // Q2 off
            PWMMR3 = 0;                        // Q3 off
            PWMMR4 = 0;                        // Q4 off
            PWMMR5 = 0;                        // Q5 off
            PWMMR6 = 0;                        // Q6 off
            PWMLER = 0x7F;                     // enable PWM0-PWM6 match latch (reload)
            while (1) ;                        // wait for a RESET
        }
```

```

        if (f_10ms)                         // every 10 mseconds
        {
            f_10ms = 0;

            if (actualSpeed > desiredSpeed)
                actualSpeed --;
            else if (actualSpeed < desiredSpeed)
                actualSpeed ++;

            RPM = 10000000 / T0CR0;         // calculate motor speed
            fRPM = ((fRPM * 15) + RPM) / 16;    // filter it
        }
    }
}
```

7.2 ADC.C

```
#include <LPC214x.h>

void ADC0_Init(void)
{
    PINSEL1 |= 0x00040000;          // P0.25 = AIN0.4
    AD0CR    = 0x00200F10;          // initialise ADC0, select AIN4
    AD0CR   |= 0x00010000;          // start burst mode now, see errata ADC.2
}
```

7.3 PWM.C

```
#include <LPC214x.h>

void PWM_Init(void)
{
    PINSEL0 |= 0x000A800A;          // select PWM1-4 and PWM6
    PINSEL1 |= 0x00000400;          // select PWM5

    PWMPR = 20;                     // prescaler to 20, timer runs at 60 MHz / 20 = 3 MHz
    PWMPC = 0;                      // prescale counter to 0
    PWMTC = 0;                      // reset timer to 0
    PWMMR0 = 100;                   // -> PMW base frequency = 3 MHz / 100 = 30 KHz
    PWMMR1 = 0;                     // Match 1 for Q1 (off)
    PWMMR2 = 0;                     // Match 2 for Q2 (off)
    PWMMR3 = 0;                     // Match 3 for Q3 (off)
    PWMMR4 = 0;                     // Match 4 for Q4 (off)
    PWMMR5 = 0;                     // Match 5 for Q5 (off)
    PWMMR6 = 0;                     // Match 6 for Q6 (off)
    PWMMCR = 0x00000002;            // reset TC on MR0
    PWMPCR = 0x7E00;                // enable PWM1 - PWM6 outputs
    PWMLER = 0x7F;                  // enable PWM0 - PWM6 match latch (reload)
    PWMTCR = 0x09;                  // enable PWM mode and start timer
}
```

7.4 HSENSOR.C

```
#include <LPC214x.H>                    // LPC214x definitions
#include "bldc.h"

__irq void T0_Isr(void)
{
    T0TC = 0;                           // Reset timer

    switch ((IO0PIN >> 18) & 7)         // read Hall sensor inputs P0.18, P0.19 and P0.20
    {
      case 1:  PWMMR1 = actualSpeed;    // phase 6: 001
               PWMMR2 = 0;
               PWMMR3 = 0;
               PWMMR4 = 0;
               PWMMR5 = 0;
               PWMMR6 = actualSpeed;
               break;
      case 2:  PWMMR1 = 0;              // phase 4: 010
               PWMMR2 = actualSpeed;
               PWMMR3 = 0;
               PWMMR4 = actualSpeed;
               PWMMR5 = 0;
               PWMMR6 = 0;
               break;
      case 3:  PWMMR1 = 0;              // phase 5: 011
               PWMMR2 = actualSpeed;
               PWMMR3 = 0;
               PWMMR4 = 0;
               PWMMR5 = 0;
               PWMMR6 = actualSpeed;
               break;
      case 4:  PWMMR1 = 0;              // phase 2: 100
               PWMMR2 = 0;
               PWMMR3 = actualSpeed;
               PWMMR4 = 0;
               PWMMR5 = actualSpeed;
               PWMMR6 = 0;
               break;
      case 5:  PWMMR1 = actualSpeed;    // phase 1: 101
               PWMMR2 = 0;
               PWMMR3 = 0;
               PWMMR4 = 0;
               PWMMR5 = actualSpeed;
               PWMMR6 = 0;
               break;
      case 6:  PWMMR1 = 0;              // phase 3: 110
               PWMMR2 = 0;
               PWMMR3 = actualSpeed;
               PWMMR4 = actualSpeed;
               PWMMR5 = 0;
```

```
                PWMMR6 = 0;
                break;
        default: break;                 // invalid
    }

    T0IR = 0xFF;                        // reset flags
    PWMLER = 0x7F;                      // enable PWM0 - PWM6 match latch (reload)
    VICVectAddr = 0;                    // Acknowledge interrupt by reseting VIC
}

void HES_Init(void)
{
    VICVectAddr1  = (unsigned int) &T0_Isr;
    VICVectCntl1  = 0x24;               // Channel1 on Source#4 ... enabled
    VICIntEnable |= 0x10;               // Channel#4 is the Timer 0

    PINSEL1 |= 0x3A000000;              // P0.30,P0.28,P0.29 as CAP0.0,CAP0.2,CAP0.3

    T0PR  = 60;                         // pre 60, timer runs at 60 MHz / 60 = 1 MHz
    T0MR0 = 1000000;                    // = 1 sec / 1 us
    T0MCR = 3;
    T0CCR = 0x0FC7;                     // Capture on both edges and enable the interrupt
    T0TC  = 0;                          // Reset timer
    T0TCR = 1;                          // start timer
}
```

7.5 TIMER1.C

```
#include <LPC214x.H>                // LPC214x definitions

char f_10ms = 0;

__irq void T1_Isr(void)             // Timer 1 ISR every 10 msec
{
    f_10ms = 1;                     // toggles every 10 mseconds
    T1IR = 0x01;                    // reset interrupt flag
    VICVectAddr = 0;                // reset VIC
}

void T1_Init(void)
{
    VICVectAddr2  = (unsigned int) &T1_Isr;
    VICVectCntl2  = 0x25;           // Channel2 on Source#5 ... enabled
    VICIntEnable |= 0x20;           // Channel#5 is the Timer 1

    T1MR0 = 600000;                 // = 10 msec / 16,67 nsec
    T1MCR = 3;                      // Interrupt on Match0, reset timer on match
                                    // Pclk = 60 MHz, timer count = 16,67 nsec
    T1TC  = 0;                      // reset Timer counter
    T1TCR = 1;                      // enable Timer
}
```

7.6 BLDC.H

```
#define MAX_Im          0xF0            // max motor current limit

extern unsigned char actualSpeed;

extern void GetInReport(unsigned char *rep);
extern void SetOutReport(unsigned char *rep);
extern void USB_Init(void);
extern void USB_Connect(unsigned int con);

extern void ADC0_Init(void);
extern void PWM_Init(void);
extern void HES_Init(void);

extern void T1_Init(void);
extern char f_10ms;
```

8. Legal information

8.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

8.2 Disclaimers

General — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is for the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

8.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

9. Contents

Date of release: 17 October 2007
Document identifier: AN10661_1