AN10513

Brushed DC motor control using the LPC2101

Rev. 01 — 12 January 2007 **Application note**

Document information

Info	Content
Keywords	LPC2101, ARM7, Brushed DC motor control
Abstract	This application note demonstrates the use of a low cost ARM7 based LPC2101 microcontroller for bidirectional brushed DC motor control. The LPC2101, or one of its LPC2000 family members, offer users high-speed 32-bit CPU performance to handle (any) other application tasks as well, making it ideal for one-chip system solutions. It also shows a complete solution from NXP Semiconductors in terms of Microcontroller – MOSFET driver – MOSFET. Application examples: moving toys, fans, printers, robots, electric bikes, -doors, -windows, -sun roofs, -seats, mixers, food processors, can openers, blenders, vacuum cleaners, toothbrushes, razors, coffee grinders, etc.

founded by Philips

Revision history

Rev	Date	Description
01	20070112	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 LPC2101 microcontroller for bidirectional brushed DC motor control.

The LPC2101 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 and 16-bit timers, 10-bit ADC and PWM features through output match on all timers, make them particularly suitable for industrial control.

Brushed DC (Direct Current) motors are most commonly used in easy to drive, variable speed and high start-up torque 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 DC motors).

2. Brushed DC motor fundamentals

A brushed DC motor typically consists of stationary fixed permanent magnets (the stator), a rotating (electro)magnet (the rotor) and a metal body to concentrate the flux (see Fig 1). By attraction of opposite poles and repulsion of like poles, a torque acts on the rotor and makes it turn. As soon as the rotor begins to turn, fixed brushes make and break contact with the rotating segments (commutation) in turn. The rotor coils are energized and de-energized in such way that as the rotor turns, the axis of the new rotor poles are always opposed to the stator poles. Because of the way the commutation is arranged, the rotor is in constant motion. By reversing the power supply to the motor, the current in the rotor coils and therefore the north and south poles are reversed and the motor changes its direction of rotation.

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. In this application note speed control is based on generating and varying a PWM signal by the LPC2101 microcontroller.

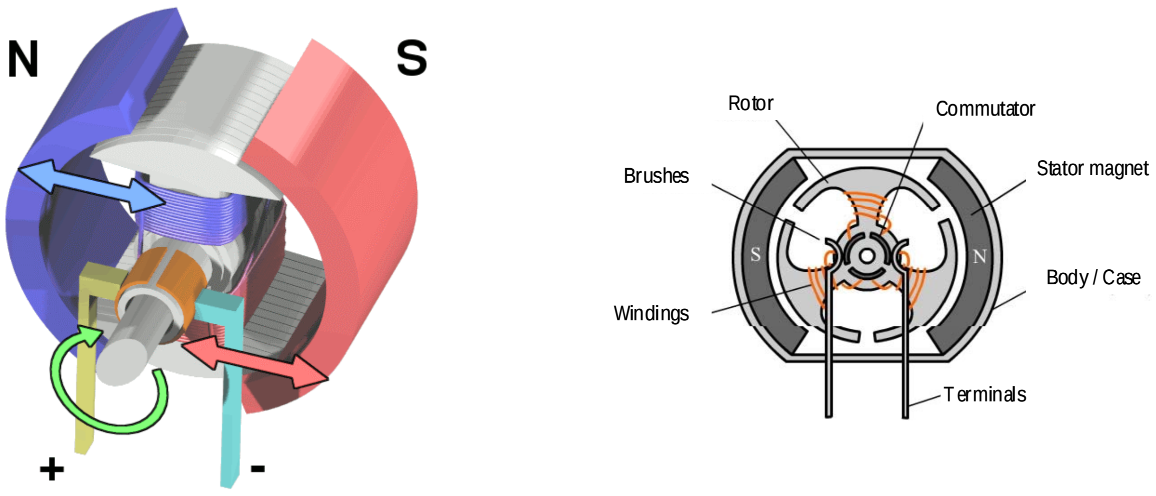


Fig 1. Brushed DC motor

3. How to control a brushed DC motor

3.1 Bidirectional rotation

Driving a brushed DC motor in both directions, by reversing the current through it, can be accomplished using a full-bridge (see Fig 2), which consists of four N-channel MOSFETs. For ‘forward’ rotation Q1 and Q4 are switched on while Q2 and Q3 are off. For ‘reverse’ rotation Q2 and Q3 are on while Q1 and Q4 are off.

If the upper two MOSFETs are turned off and the lower ones are turned on, the motor is ‘braking’. The motor will ‘coast’ (free running) if all four switches are turned off.

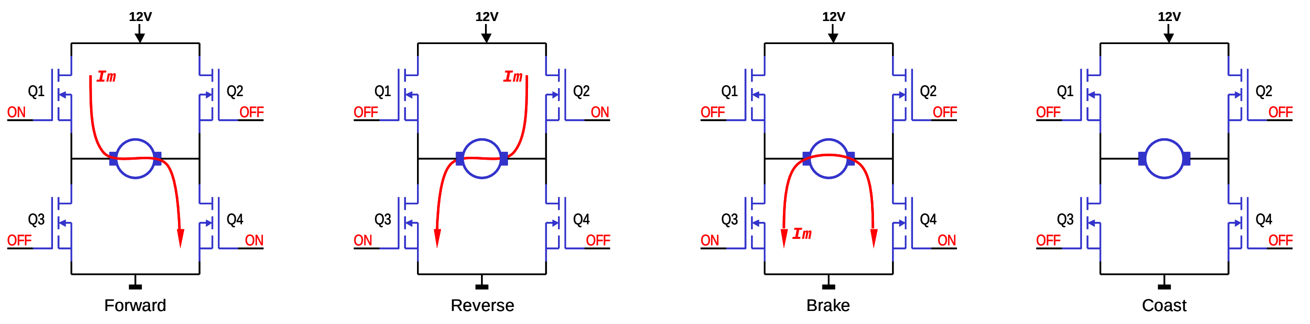


Fig 2. Bidirectional rotation using a full-bridge

3.2 Speed control

The no-load motor speed is proportional to the voltage applied across the motor. Thus by simply varying the voltage across the motor, one can control the speed of the motor. Pulse Width Modulation (PWM) is used to implement this (see Fig 3). It is based on a fixed frequency pulse waveform with a variable duty cycle. The average voltage applied to the motor is proportional to the PWM duty cycle.

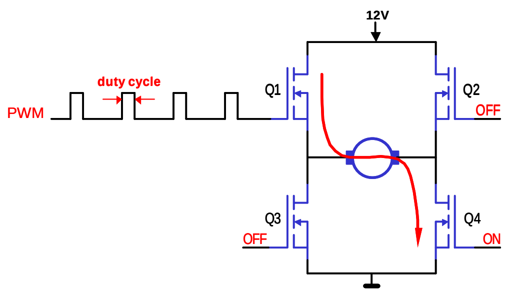


Fig 3. PWM speed control

In our application, the PWM signals (for Q1 and Q2) are generated by two Timer 2 match outputs of the LPC2101 microcontroller. A third Timer 2 match register is used to determine the signals base frequency. The motor speed (duty cycle) and direction are adjusted by reading a potentiometer using one of the LPC2101s ADC inputs (see Fig 4).

3.3 Motor feedback

3.3.1 Current sense

Low cost motor current measuring is implemented using a current sensing resistor between the MOSFETs and ground (see Fig 4). The small voltage appearing across the current sense resistor is filtered and amplified, before being fed to an ADC input of the microcontroller. Current is always measured at its highest point, just before the end of the PWM ‘on’ time. This is accomplished by using an extra Timer match interrupt that starts the AD conversion. The converted value represents the motor current.

In this application note measuring the motor current is 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).

3.4 RPM measurement

Low cost sensorless motor rotation speed feedback is implemented by Back EMF voltage measuring (see Fig 4). Back electromotive force (also called BEMF) is an electromotive force that occurs in electric motors and generators where there is relative motion between the rotor magnet of the motor and the external magnetic field. In other words, the motor acts like a generator as long as it rotates. The RPM is directly proportional to the back EMF voltage.

Back EMF is measured with the modulated MOSFET switched off (‘brake’ mode). In this application note the BEMF measurement is used to determine whether or not the motor has completely stopped, before for example the rotation direction is reversed. A voltage divider is used to fit the back EMF voltage (max. 12 V) into the 0 V to 3V3 range of the LPC2101 ADC input.

4. Application setup

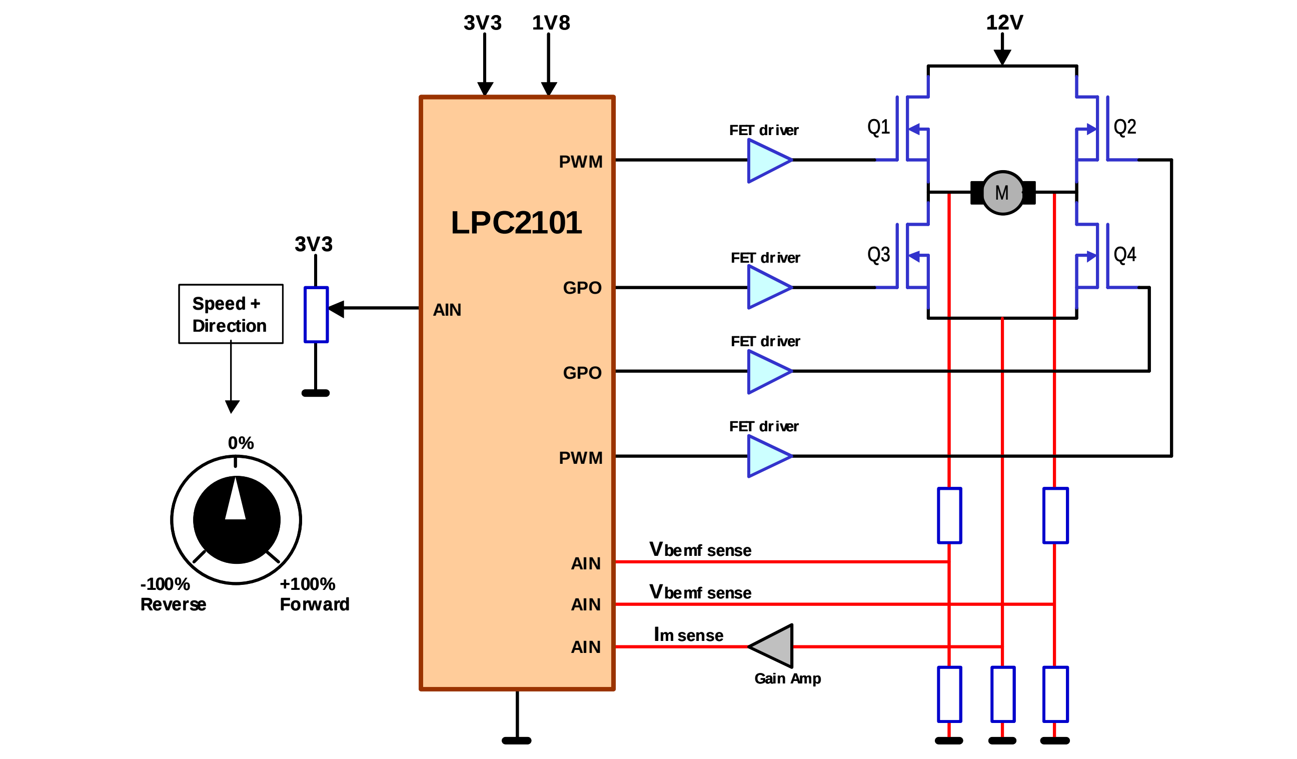


Fig 4. System configuration

4.1 Using the LPC2101

For this application note we decided to use the LPC2101. Available in an LQFP48 package it is the smallest and cheapest member (for now) of the ARM7 based LPC2000 family. It offers high speed (70 MHz) 32-bit CPU performance, 2 kB of on-chip static RAM and 8 kB of on-chip flash program memory. For larger memory - or specific peripheral (USB, 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:

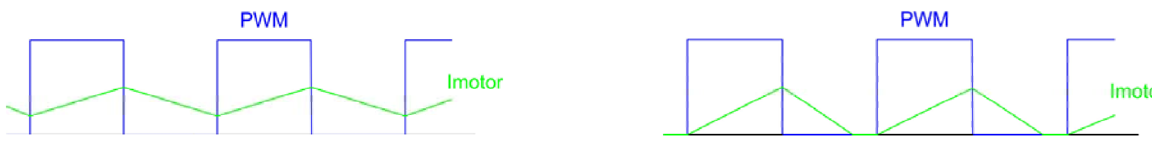
– CPU load is less than 5 %, used code size is 3 kB (including RS232 communication)
– Unused peripherals: UART, I2C, SPI/SSP, RTC, 2 x Timer and 4 x A/D input
– 20 unused GPIO pins available for user's application

4.2 Motor selection

For this application note a 150 W Maxon RE-40 motor is used. No load speed is 6920 RPM at 12 V input. The maximum continuous current is 6 A.

The base frequency of the PWM signal plays an important role in the sound of the motor, thus it affects the human ear that can detect frequencies from 20 Hz to 20 kHz. It will also influence the behavior of the motor. Sometimes there is insufficient armature inductance to prevent the current from falling to zero for part of each cycle. This is known as the ‘discontinuous’ current mode (see Fig 5b), and it is usually encountered when the motor is lightly loaded. It is very undesirable because when the current is discontinuous, the speed falls off rapidly when the load increases. With discontinuous current, the relevant part of the torque-speed curve is very droop and it will cause some pulsing in the motor armature, making the motor much noisier.

For this application, using the mentioned Maxon motor, and in order to accomplish ‘continuous’ current mode a PWM frequency of 8 KHz has been selected.



a. Continuous mode (wanted)

b. Discontinuous mode

Fig 5. Influence of PMW base frequency

4.3 MOSFET selection

The NXP Semiconductors PH1875L N-channel TrenchMOS logic level FET is used for this system. It is chosen in relation with the selected motor, which is supplied with 12 V, and requires a maximum starting current of 103 A.

For a 12 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 / down towards the required speed) implemented in software. The PH1875L can deal with a maximum drain current of 45.8 Amps and a peak current of 183 Amps and is available in an SMD SOT669 (LFPAK) package (see Fig 6).

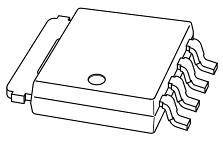


Fig 6. SOT669 (LFPAK) package

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 PMD2001D and the PMGD280UN from NXP Semiconductors to do the job, as shown in Fig 7.

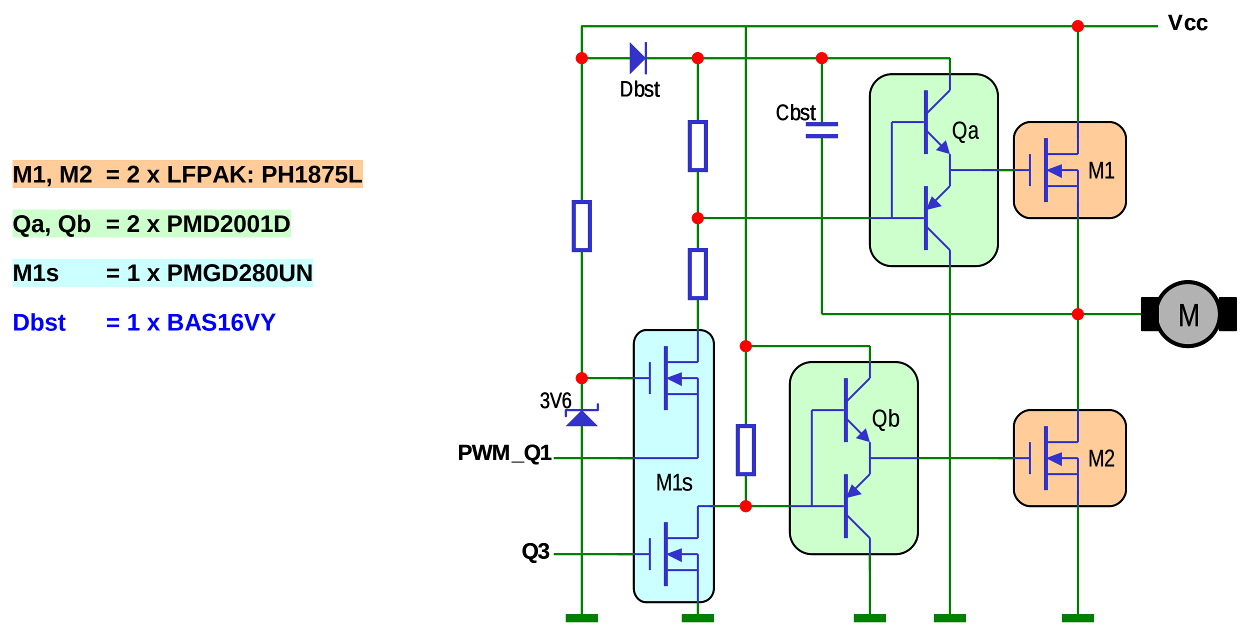


Fig 7. Simplified MOSFET – MOSFET driver diagram for half of the full bridge

4.5 Controlling speed and direction

In order to control the direction and speed of the motor a 10 kΩ potentiometer, connected to an ADC input of the LPC2101, is used (see Fig 4). The A/D converter has a 10-bit resolution, but in our case only 8 bits are used (no jitter). This means there are 256 possible potentiometer steps (see Fig 8). The center position (+ hysteresis) is the resting point for the motor speed ('break').

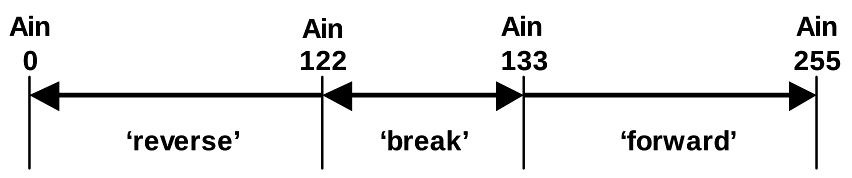


Fig 8. Potentiometer analog input 'speed' and 'direction' scale

5. Hardware schematics

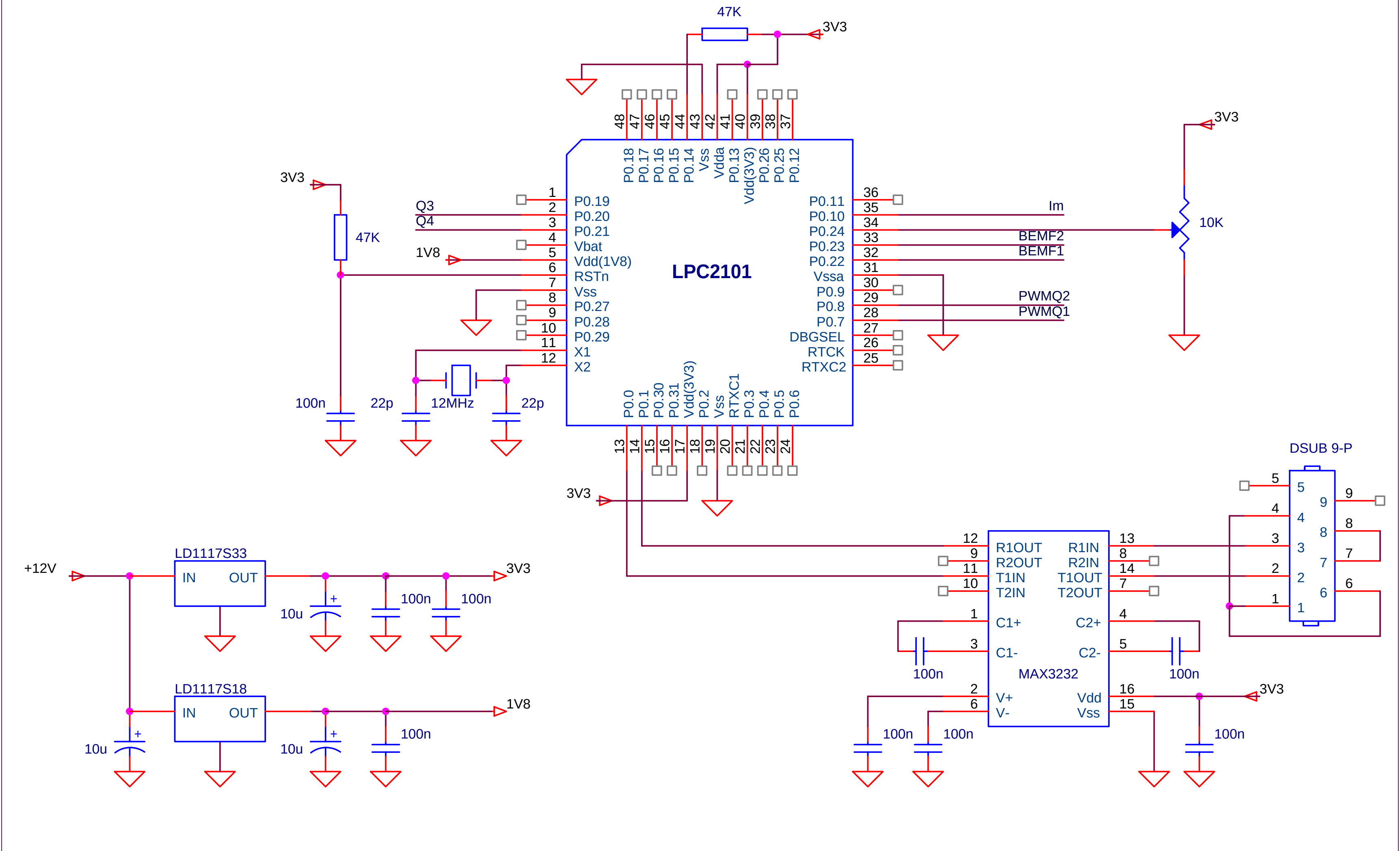


Fig 9. Hardware schematics – controller part

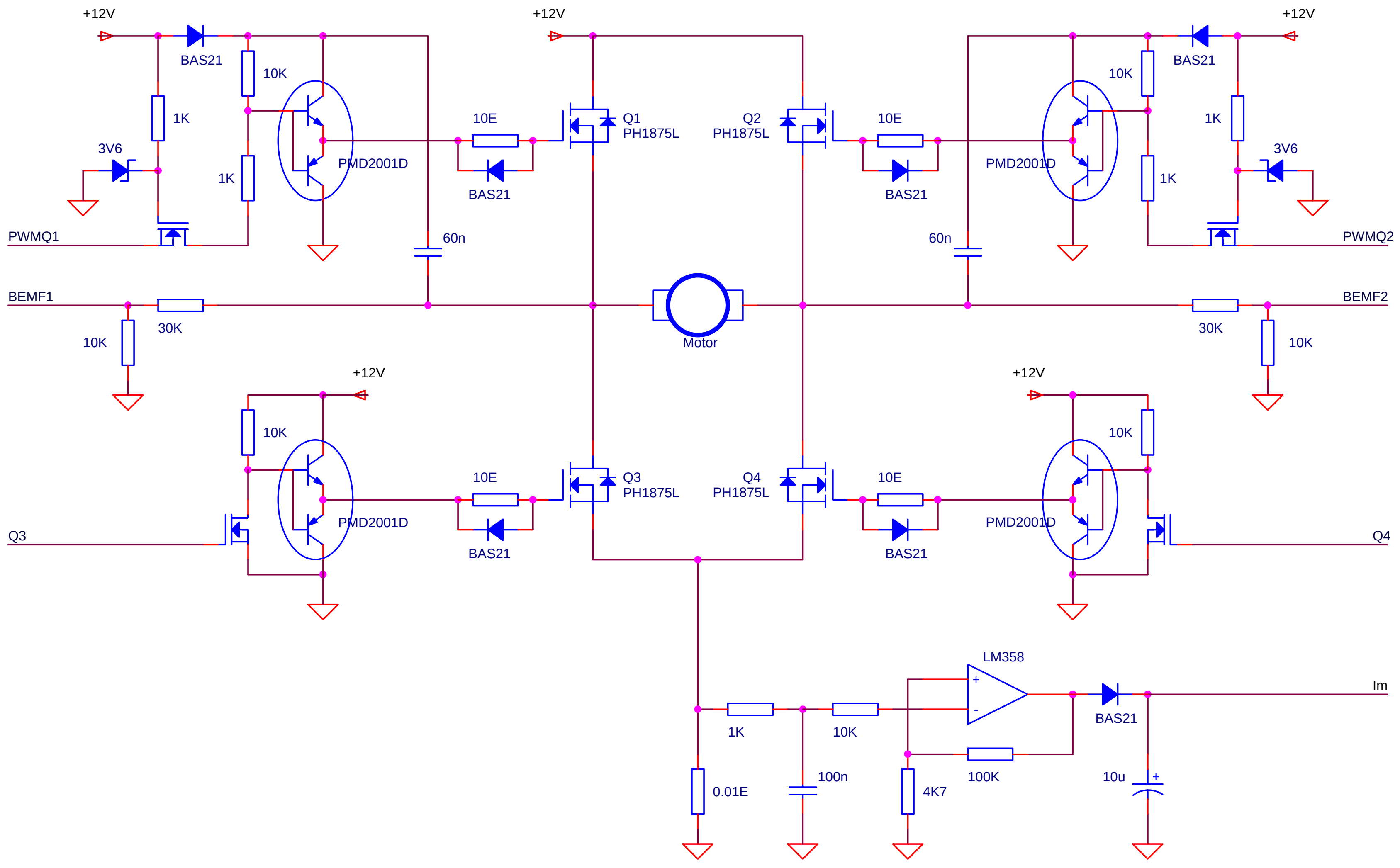


Fig 10. Hardware schematics – power / motor part

6. Software

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:

- Read potentiometer for desired speed and direction
- Read and 'guard' the motor current
- Set PWM duty cycle and Q1-Q4 MOSFET outputs
- RS232 communication (optional)

Fig 11 shows a flow chart of the main loop. The white blocks are for sending system information (every 200 ms) to a PC using RS232 communication and are 'optional'. The main motor control software part is a state machine as shown in Fig 12.

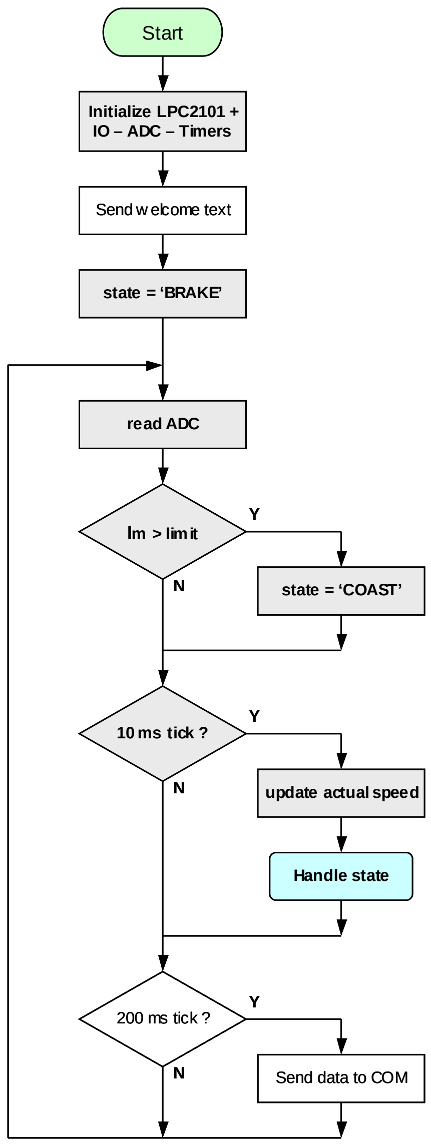


Fig 11. Main routine

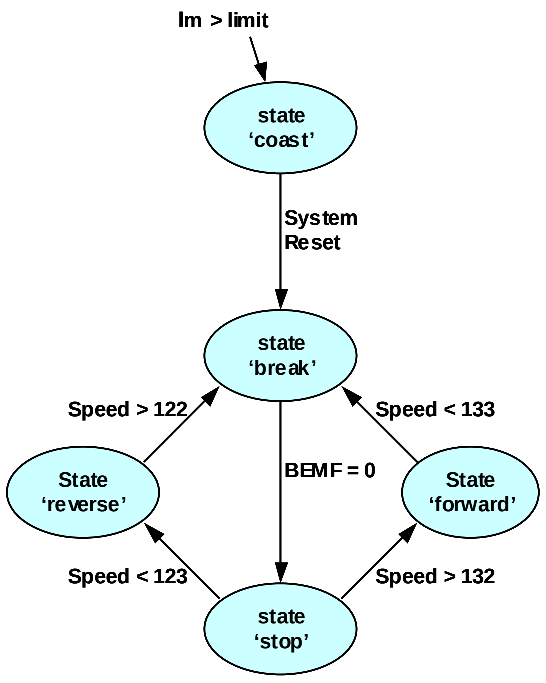


Fig 12. State handler

State changes are handled inside the main loop only. Timer 2 of the LPC2101 is used to generate the PWM signals. At the beginning of each PWM cycle an interrupt routine is entered (*T2_Isr*) to change the duty cycle according the desired speed and to set MOSFET outputs Q1 to Q4. Timer 0 is used as a 10 ms system timer.

7. Source code listings

The complete project consists of five modules (main.c – adc.c – timer0.c – motor.c uart.c) and a header file (bcd.h), all listed below. For LPC2101 configuration the standard startup code from Keil was used and set as CCLK = 60 MHz and PCLK = 15 MHz.

7.1 MAIN.C

```
#include <LPC2103.h>
#include "bdc.h"

BYTE state, actualSpeed;

int main(void)
{
    UART0_Init();
    ADC_Init();
    T0_Init();                                  // 10 msec tick
    T2_Init();                                  // used for PWM

    state = ST_BRAKE;

    PrintString("\f\nLPC2101 Brushed DC Motor Control September 2006\n\n"
                "BEMF1  BEMF2  Speed   Im\n\n");

    while(1)
    {
        ADC_Sample();                           // get latest A/D values

        if (motorCurrent > MAX_Im)              // Check motor current
            state = ST_COAST;

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

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

            switch (state)
            {
             case ST_COAST:                     // wait for a system reset
                break;
             case ST_STOP:
                if (desiredSpeed < 123)         // reverse ?
                    state = ST_REVERSE;
                else if (desiredSpeed > 132)    // Forward ?
                    state = ST_FORWARD;
                break;
             case ST_BRAKE:
                if (bemf1 < 5 && bemf2 < 5)     // wait till motor has stopped
                {
                    actualSpeed = 127;
                    state = ST_STOP;
                }
                break;
            case ST_FORWARD:
                if (desiredSpeed < 133)             // not forward ?
                    state = ST_BRAKE;               // then brake !
                break;
            case ST_REVERSE:
                if (desiredSpeed > 122)             // not reverse ?
                    state = ST_BRAKE;               // then brake !
                break;
            default:
                break;
        }
    }

    if (f_200ms)                                    // every 200 mseconds
    {
        f_200ms = 0;                                // reset flag

        PrintString(" ");     PrintByte(bemf1);
        PrintString("     "); PrintByte(bemf2);
        PrintString("     "); PrintByte(desiredSpeed);
        PrintString("     "); PrintByte(motorCurrent);
        PrintString("\r");
    }
  }
}
```

7.2 ADC.C

```
#include <LPC2103.h>
#include "bdc.h"

BYTE bemf1;
BYTE bemf2;
BYTE desiredSpeed;
BYTE motorCurrent;

void ADC_Init(void)
{
    PINSEL0 |= 0x00300000;        // P0.10=AIN3
    PINSEL1 |= 0x0003F000;        // P0.22=AIN0, P023=AIN1, P0.24=AIN2
}

void ADC_Sample(void)             // called by main
{
    bemf1 = ADDR0 >> 8;
    bemf2 = ADDR1 >> 8;
    desiredSpeed = ADDR2 >> 8;    // speed+direction potentiometer value
    motorCurrent = ADDR3 >> 8;
}
```

7.3 TIMER0.C

```c
#include <LPC2103.h>

char f_10ms  = 0;
char f_200ms = 0;

__irq void T0_Isr(void)                          // Timer 0 ISR every 10 msec
{
  static unsigned char cnt = 0;

    f_10ms = 1;                                  // toggles every 10 mseconds
    if (++cnt > 20)
    {
        cnt = 0;
        f_200ms = 1;                             // toggles every 200 mseconds
    }
    T0IR = 0x01;                                 // reset interrupt flag
    VICVectAddr = 0;                             // reset VIC
}

void T0_Init(void)
{
    VICVectAddr0  = (unsigned int) &T0_Isr;
    VICVectCntl0  = 0x24;                        // Channel0 on Source#4 ... enabled
    VICIntEnable |= 0x10;                        // Channel#4 is the Timer0

    T0MR0 = 150000;                              // = 10 msec / 66 nsec
    T0MCR = 3;                                   // Int on Match0, reset timer on match
                                                 // Pclk = 15 MHz, timer count = 66 nsec
    T0TC  = 0;                                   // reset Timer counter
    T0TCR = 1;                                   // enable Timer
}
```

7.4 MOTOR.C

```c
#include <LPC2103.H>                 // LPC2103 definitions
#include "bdc.h"

__irq void T2_Isr(void)              // for Timer 2 interrupt (FIQ vector see startup.s)
{
  static BYTE ch = 0x01;

    if (T2IR & 4)
    {
        ADCR  = 0x00200308;          // select AIN3 for motor current Im
        ADCR |= 0x01000000;          // Start A/D Conversion
        T2IR  = 0x04;                // clear MR2 interrupt flag
    }
    else
    {
        switch (state)
        {
          case ST_COAST:  T2MR0 = 255;                    // Q1 off
                          T2MR1 = 255;                    // Q2 off
                          IOSET = 0x300000;               // Q3 + Q4 off
                          break;
          case ST_STOP:   break;
```

```
            case ST_BRAKE:   T2MR0 = 255;                     // Q1 off
                             T2MR1 = 255;                     // Q2 off
                             IOCLR = 0x300000;                // Q3 + Q4 on
                             break;
            case ST_FORWARD: T2MR0 = ~actualSpeed & 0x00FF;   // set Q1 duty cycle
                             IOSET = 0x100000;                // Q3 off
                             break;
            case ST_REVERSE: T2MR1 = actualSpeed;             // set Q2 duty cycle
                             IOSET = 0x200000;                // Q4 off
                             break;
            default:         break;
          }
          ch <<= 1;
          if (ch == 0x08)  ch = 0x01;

          ADCR  = 0x00200300 | ch;              // select next channel of AIN0, 1 and 2
          ADCR |= 0x01000000;                   // Start A/D Conversion
          T2IR = 0x08;                          // clear MR3 int flag
      }
  }

  void T2_Init(void)                      // Init Timer 2 as PWM timer
  {
      IODIR   |= 0x00300000;              // P0.20 -> Q3 and P0.21 -> Q4
      IOSET    = 0x00300000;              // Q3 and Q4 off (coast mode)
      PINSEL0 |= 0x00028000;              // P0.7 (Q1) and P0.8 (Q2) as Timer 2 match outputs

      T2PR = 15;                          // prescaler to 15, timer runs at 15MHz / 15 = 1 MHz
      T2PC = 0;                           // prescale counter to 0
      T2TC = 0;                           // reset timer to 0

      T2MR0 = 255;                        // Match 0 for Q1 (off)
      T2MR1 = 255;                        // Match 1 for Q2 (off)
      T2MR2 = 122;                        // Match 2 for AIN3 (Im) start conversion
      T2MR3 = 127;                        // -> PMW base frequency = 1MHz / 127 = ~ 8 KHz
      T2MCR = 0x0640;                     // reset TC on MR3 and gen. Interrupt on MR2 and MR3
      T2PWMCON = 0x0000000B;              // enable PWM outputs

      VICIntEnable |= 0x04000000;         // Enable T2 in the VIC (channel#26)
      VICIntSelect  = 0x04000000;         // Assign (only) T2 to FIQ category

      T2TCR = 0x01;                       // start timer
  }
```

7.5 UART.C

```
  #include <LPC2103.H>                          // LPC21xx definitions

  const char ascii[] = "0123456789ABCDEF";

  void UART0_Init(void)
  {
      PINSEL0 |= 0x05;
      U0FCR    = 0x07;                          // 550 mode and FIFO's reset
      U0LCR    = 0x83;                          // UART 8N1, allow access to divider-latches
      U0DLL    = 0x31;                          // baud rate fixed to 19200 @ PCLK = 15 Mhz
      U0DLM    = 0x00;
      U0LCR    = 0x03;                          // UART 8N1, forbid access to divider-latches
  }

  static void ua_outchar(char c)
  {
      U0THR = c;
      while(!(U0LSR & 0x40)) ;
  }

  void PrintByte(unsigned char b)
  {
      ua_outchar(ascii[b >> 4]);
      ua_outchar(ascii[b & 0x0f]);
  }

  void PrintString(const char *s)
  {
      while (*s)
      {
          if (*s == '\n')
              ua_outchar('\r');           // output a '\r' first
          ua_outchar(*s);
          s++;
      }
  }
```

7.6 BDC.H

```
typedef unsigned char   BYTE;
typedef unsigned short  WORD;
typedef unsigned long   LONG;

#define ST_COAST       0           // all MOSFETs off
#define ST_STOP        1           // motor stopped
#define ST_BRAKE       2           // motor is braking
#define ST_FORWARD     3           // motor runs in forward direction
#define ST_REVERSE     4           // motor runs in reverse direction

#define MAX_Im         0xF0        // max motor current limit

extern BYTE state;
extern BYTE actualSpeed;

extern void T0_Init(void);
extern char f_10ms;
extern char f_200ms;

extern void UART0_Init(void);
extern void PrintByte(unsigned char b);
extern void PrintString(const char *s);

extern void ADC_Init(void);
extern void ADC_Sample(void);
extern BYTE bemf1;
extern BYTE bemf2;
extern BYTE desiredSpeed;
extern BYTE motorCurrent;

extern void T2_Init(void);
```

8. Legal information

8.1 Definitions

8.2 Disclaimers

8.3 Trademarks

9. Contents

Date of release: 12 January 2007
Document identifier: AN10513_1