AN11094

Sensored BLDC motor control with LPC111x/LPC11Cxx

Rev. 1 — 1 August 2011 | Application note

Document information

Info	Content
Keywords	LPC1112, LPC1114, LPC11C12, LPC11C14, LPC11C24, ARM Cortex-M0, BLDC, Motor control, CMSIS, White Goods, PWM, Timers, CAN, UART, LPCXpresso, LPCXpresso Motor Control, Embedded Artists
Abstract	This application note describes the implementation of Sensored Brushless DC motor control in the LPC1100 using the general purpose timers in PWM mode made for the LPCXpresso Motor Control kit.

Revision history

Rev	Date	Description
1	20110801	Initial version.

Contact information

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

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

1. Introduction

Brushless DC (BLDC) motors are replacing traditional brushed DC (BDC) motors in markets like White Goods (WG), Heating Ventilation Air Conditioning (HVAC) and industrial applications due to higher efficiency and reliability, reduced noise and weight, longer lifetime, elimination of sparks created by the commutator and overall reduction of Electro Magnetic emissions.

NXP is a broad-based supplier for industrial applications, including General Application (Rectifiers, Zener diodes, etc), Logic and Power (Triacs, Power IC) as well as Interface and Microcontroller products.

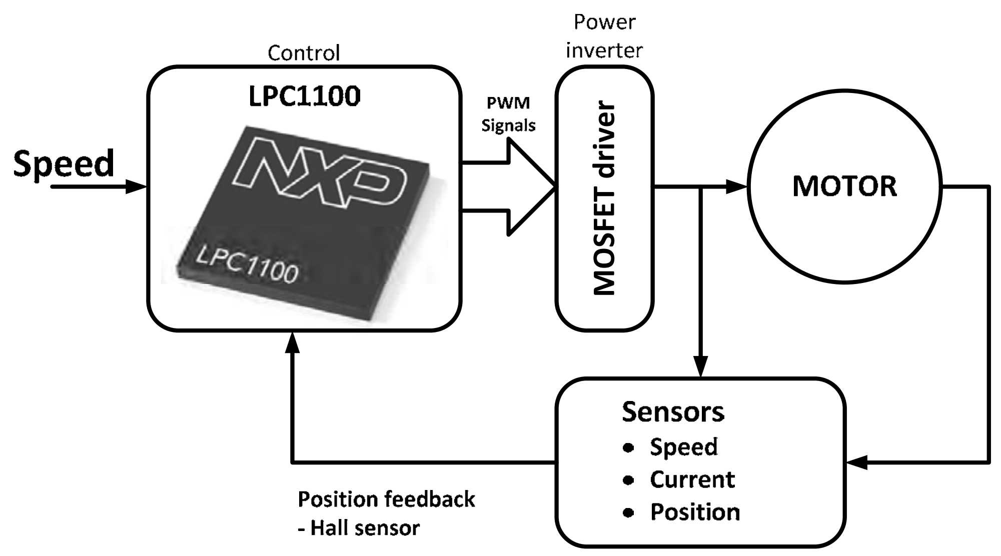


Fig 1. LPC1100 Motor control system block diagram

In common with most motor controlling, BLDC control consists of a control unit and a power unit, with NXP offering competitive solutions for both units.

1.1 LPCXpresso Motor Control Kit

This application note focuses on the LPCXpresso Motor Control Kit that Embedded Artists has developed in close cooperation with the NXP Microcontrollers applications group.

The LPCXpresso Motor Control Kit is a multi-motor type and multipurpose evaluation board. It supports BLDC, Brushless AC (BLAC), stepper and one or two BDC motors. For more detailed information on the hardware, please see Chapter 3, as well as http://www.embeddedartists.com/products/lpcxpresso/xpr_motor.php

The implementation of six-step commutation or brushless DC motor control on the LPC1100 is described in this application note together with all required hardware and software. This hardware and software can be used to build a tailored motor controlling application with the low-cost, low-power LPC1100 family.

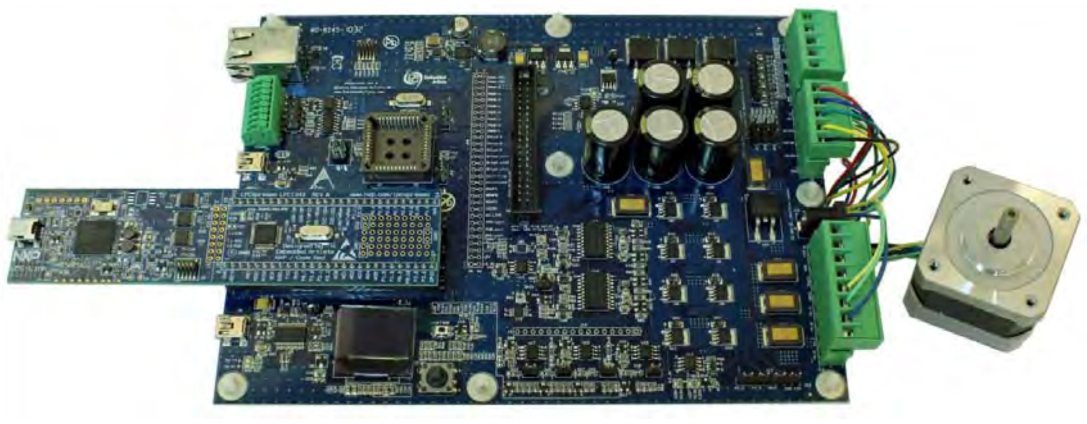


Fig 2. LPCXpresso Motor Control kit

1.2 Getting started with the LPCXpresso Motor Control Kit

The LPCXpresso Motor Control kit is an out-of-the-box solution enabling you to have a BLDC motor running within five minutes. In order to understand all the possibilities and capabilities of this kit, please register your kit at http://www.embeddedartists.com/support/. Extensive documentation and example code can be found at the Embedded Artist website.

1.3 How to read this application note

As additional information it is recommended to keep the LPC1100 User Manual (UM10398) at hand.

Application notes AN10661 “Brushless DC motor control using LPC2141” and AN10898 “BLDC motor control with LPC1700” are used as reference for this application note.

This application note is written dedicated for the Cortex-M0 LPC1100 series though the baseline works on the Cortex-M3 LPC1300 series as well. The peripherals required except for the CAN interface are also present on the LPC1300 series.

Supported devices are:

Table 1. Supported devices

This application note supports:
LPC111x
LPC11C1x
LPC11C2x

Table 2. Handy references

Document name	**Description**
UM10398	LPC11xx and LPC11Cxx User Manual
Datasheet LPC11xx	LPC11xx and LPC11Cxx Data Sheet
Embedded Artists LPCXpresso Motor control Kit	Product page on Embedded Artists LPCXpresso Motor Control Kit.
LPCXpresso_Motor_Control Users_Guide_Rev_PA4.pdf[1]	Users Guide for the LPCXpresso Motor Control kit
AN10661	Brushless DC motor control using LPC2141
42BLF01 datasheet.pdf [1]	Datasheet of the LPCXpresso Motor Control Kit datasheet
LPCXpressoLPC1114revA.pdf[1]	LPC1114 LPCXpresso board schematics
LPCXpresso LPC11C24 rev B_schematic.pdf[1]	LPC11C24 LPCXpresso board schematics
Motor_Control_Evaluation_Board_rev_A_SCHEMATICS.pdf[1]	LPCXpresso motor control board schematics
AN10898	BLDC motor control with LPC1700

[1] These documents are available in the example code bundle. Open up the example project in the LPCXpresso IDE and look in the _Documentation folder

NOTE: Please read the LPCXpresso Motor Control Kit Users Guide before continuing with this application note. This will help you with setting up the system.

2. The LPC1100

2.1 The LPC111x/LPC11Cxx

In 2009 NXP Semiconductors released the LPC1100, the first microcontroller series based on the ARM Cortex-M0 core.

From a computational point of view, the LPC1100 series is able to deliver 0.9 DMIPS/MHz according to Dhrystone benchmarks.

According to Coremark (http://www.coremark.org/), a benchmark based on real world performance of embedded devices, NXP's LPC1100 family is positioned at 1.4 Coremark/MHz which is extremely high compared to competitive 8-bit and 16-bit microcontrollers. At the same time, users can benefit from the improved code density. On average, developers can save around 40 % of the flash memory utilization compared to competitive 8/16-bit MCUs.

Cortex-M0 based devices can be used in low power applications such as medical devices, e-metering, motor control and battery powered sensors. Cortex-M family processors integrate support for multiple power modes: Sleep, deep sleep, and power-down modes.

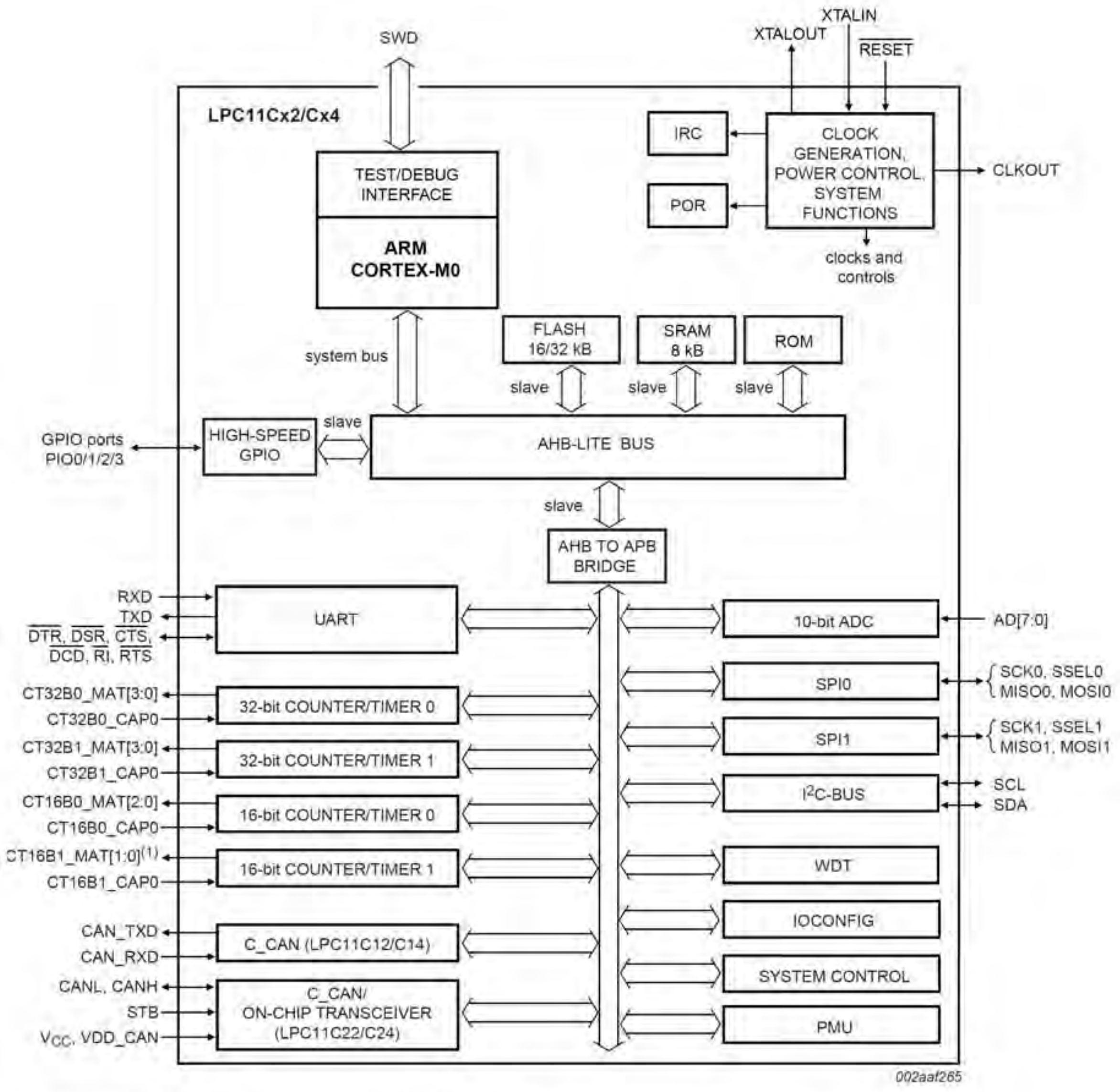
The LPC1100 series supports up to 50 MHz clock speed at zero latency and integrates a simple AHB-Lite interface.

The LPC1100 series integrates all necessary peripherals for embedded control in industrial, consumer and white goods applications. For controlling BLDC motors, the LPC1100 series incorporates four timers: Two 16-bit and two 32-bit, with a total of 13 match outputs, where each match output can be configured for Pulse Width Modulation (PWM). Six of these PWM signals are used in the demonstration board driving the high and low side MOSFETs.

The General-Purpose Input/Output (GPIO) pins on the LPC1100 are highly configurable and can be used as external interrupt triggering on the rising, falling or both edges. Rotor orientation feedback is captured through these GPIO interrupts.

The LPC1100 has an 8-channel 10-bit Analog-to-Digital Converter (ADC) from which one channel can be used for over-current protection by measuring the motor current through a shunt resistor.

By measuring the voltage on the floating phase during BLDC commutation, the rotor orientation can be determined without the use of any sensors. This requires accurate timing in capturing the floating phase voltage. In the LPC1100, an ADC conversion can be triggered by a match event of two of the four timers. This decreases CPU load and allows accurate capturing of the floating phase at the right moment.



(1) CT16B1_MAT0 not available on parts LPC11C22/C24.

Fig 3. LPC11xx/LPC11Cxx block diagram

2.2 Timers and PWM

2.2.1 Description

The LPC1100 series are equipped with two 16-bit and two 32-bit counter/timers with a PWM that uses up to three match outputs as single edge-controlled PWM outputs.

Each match output can be configured for PWM or match output. The function can be selected by the External Match Register (EMR).

One additional match register is used for determining the PWM cycle period. When a match occurs in any of the other match registers, the PWM output is set to HIGH. The timer is reset by the match register that is configured to set the PWM cycle period. When the timer is reset to zero, all currently HIGH match outputs configured as PWM outputs will be cleared.

The PWM Control (PWMC) register is used to configure the match outputs as PWM outputs.

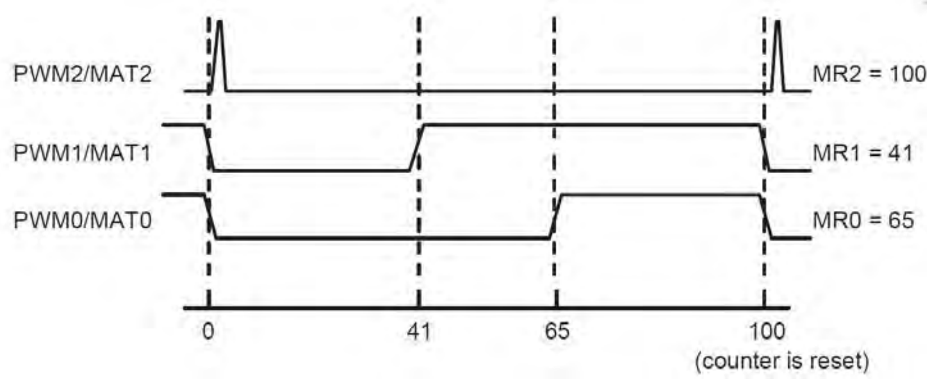


Fig 4. Sample PWM waveforms

2.2.2 How to set up the PWM

This chapter will give a brief introduction of the sequence required for setting the timer/counters in PWM mode. Please reference Chapter 4 and the source code (timer16.c and timer32.c) included with this application note for more details.

Although PWM functionality is implemented in the included source code, this step by step approach gives a high level insight how to correctly configure the timer/counter as a PWM source.

Step by step

1. Enable the clock to the timer/counter domain using the `SYSAHBCLKCTRL` register

   16 bit timer/counter 0 = CT16B0 = bit 7 of SYSAHBCLKCTRL

   16 bit timer/counter 1 = CT16B1 = bit 8 of SYSAHBCLKCTRL

   32 bit timer/counter 0 = CT32B0 = bit 9 of SYSAHBCLKCTRL

   32 bit timer/counter 1 = CT32B1 = bit 10 of SYSAHBCLKCTRL

Check the timer/counter input clock frequency; this will be needed for defining the PWM period.

2. Configure the external match pins using the EMR register.
3. Configure the I/O Pin-mux to timer/counter functionality.
4. Enable the PWM mode in the PWM Control register. When corresponding bits are set in the PWMC register, at a match the selected channels will be set HIGH.
5. Set the PWM period by writing the match value in the match register selected for resetting the timer counter. See Fig 4.
6. Enable reset and/or interrupt triggering at match of the period defining register in the match control register (MCR)

2.3 General Purpose I/O (GPIO)

The LPC1100 series GPIO can be configured as an (external) interrupt source acting on high and low level, rising, falling and both edges.

This functionality is used in this application for feeding back the Hall sensor outputs into the microcontroller. Setting up the GPIOs as interrupt sources requires setting a few registers.

Step by step

1. Enable the clock to the GPIO domain using the SYSAHBCLKCTRL register
2. Set the GPIO as inputs in the DIR register (input is default after reset)
3. Then set up the sense register, selecting whether the IO is edge or level sensitive. This can be done through the IS register.
4. Next the “both edges” sense register (IBE) should be configured. With this register one can select whether an edge sensitive input should act on both or only one edge.
5. Selection of which edge to act on is done through the interrupt event register (IEV).
6. Interrupts of other unwanted pins can be masked using the interrupt mask register (IE).
7. Now enable the GPIO interrupts.

2.4 Analog-to-Digital Converter (ADC)

The LPC1100 series has a 10-bit successive approximation ADC. This ADC can be used to measure the motor current.

The ADC input is multiplexed on 8 pins. It does a single or burst conversion on single or multiple inputs. Triggering the conversion can be done by setting a bit in the A/D Control Register, but other trigger sources like timer matches or input (capture) pins can be selected.

2.5 Controller Area Network (CAN)

The LPC11C1x series are equipped with a C_CAN peripheral. This application gives the possibility for using the CAN interface to a PC or an industrial network. Chapter 4.2.1.2 describes how the CAN is implemented and used in this application.

Controller Area Network (CAN) is the definition of a high performance communication protocol for serial data communication. The C_CAN controller is designed to provide a full implementation of the CAN protocol according to the CAN Specification Version 2.0B.

The CAN controller consists of a CAN core, message RAM, a message handler, control registers, and the APB interface. For communication on a CAN network, individual Message Objects are configured. The Message Objects and Identifier Masks for acceptance filtering of received messages are stored in the Message RAM.

All functions concerning the handling of messages are implemented in the Message Handler. Those functions are the acceptance filtering, the transfer of messages between the CAN Core and the Message RAM, and the handling of transmission requests as well as the generation of the module interrupt.

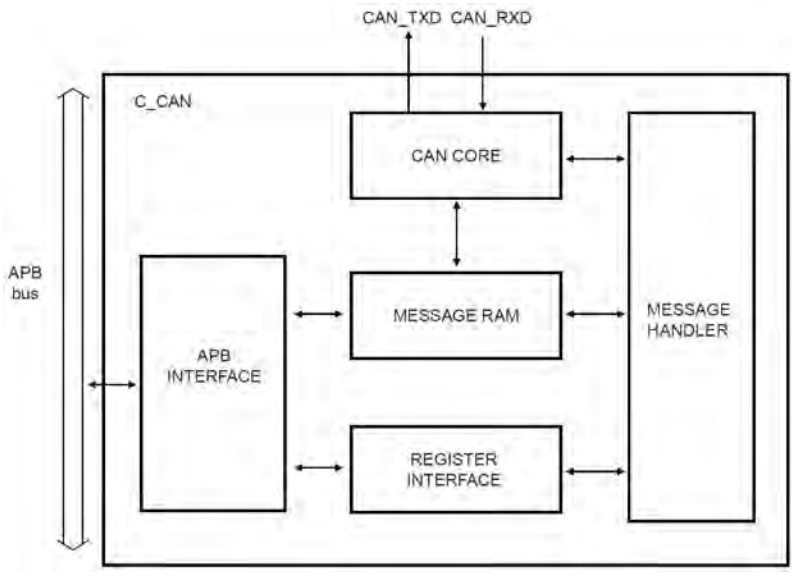


Fig 5. C_CAN block diagram

2.6 On-chip CAN drivers

NXP offers on-chip CAN drivers and CANopen initialization and communication, stored in ROM on the LPC11C1x series. An API gives user applications easy access to these drivers.

These drivers cover initialization, configuration, basic CAN send and receive, as well as a CANopen SDO interface. Receive events processing is made available through callback functions.

The on-chip drivers API include the following functionality:

- CAN set-up and initialization
- CAN send and receive messages
- CAN status
- CANopen Object Dictionary

- CANopen SDO expedited communication
- CANopen SDO fall-back handler

2.6.1 Calling the C_CAN API

The C_CAN API calling is set up to be approached through pointers. A fixed location in ROM contains a pointer to the ROM driver table, i.e. 0x1FFF 1FF8. This location is the same for all LPC11C1x parts. The ROM driver table contains a pointer to the CAN API table. Pointers to the various CAN API functions are stored in this table. CAN API functions can be called by using a C structure.

Fig 6 illustrates the pointer mechanism used to access the on-chip CAN API. On-chip RAM from address 0x1000 0050 to 0x1000 00B8 is used by the CAN API. This address range should not be used by the application. For applications using the on-chip CAN API, the linker control file should be modified appropriately to prevent usage of this area for the application's variable storage.

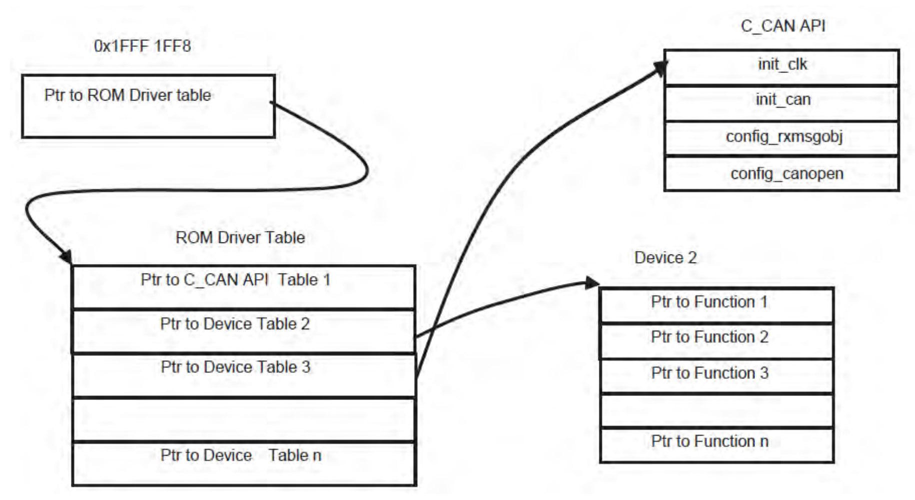


Fig 6. CAN API pointer structure

For more detailed information on the C_CAN and the on-chip CAN drivers, please see the LPC11xx/LPC11Cxx User Manual.

3. Application setup

This motor control application can be divided in three main parts: The control unit, the power unit, (both of which are implemented on the LPCXpresso Motor Control board), and the external BLDC motor (Fig 7).

This chapter will describe how the application is set up.

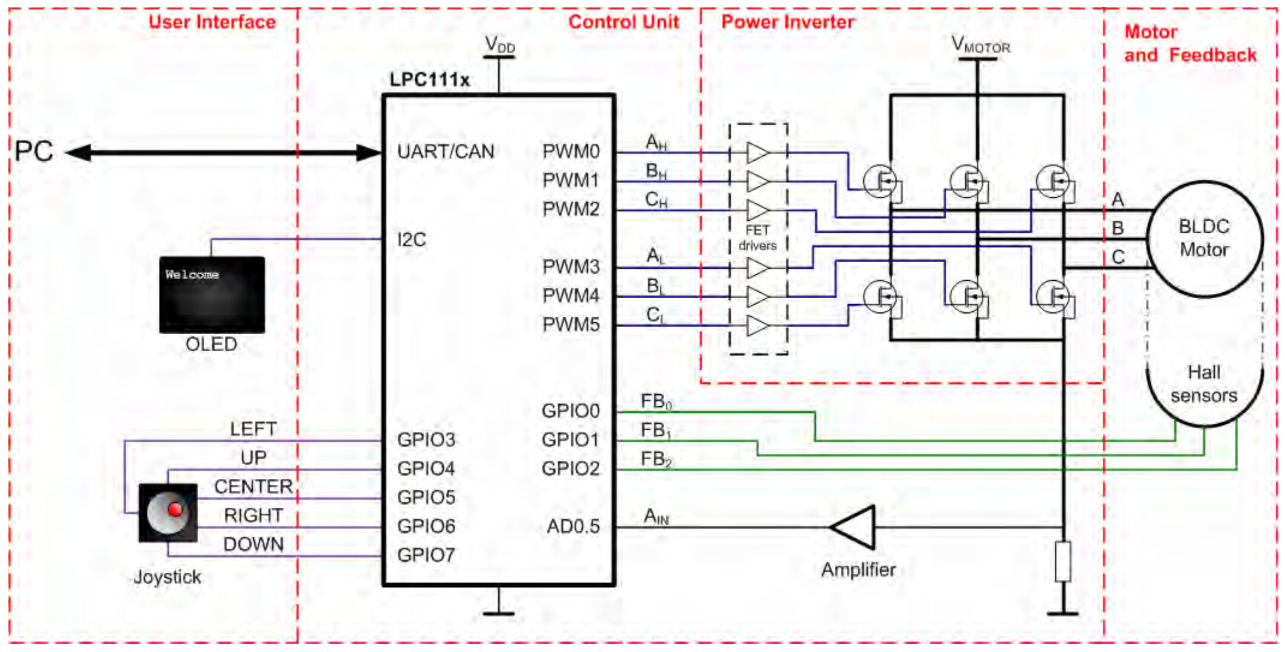


Fig 7. LPC1100 Motor control application block diagram

3.1 Control unit

The main component of the control unit is the LPC1100, gathering various inputs and then controlling outputs. The inputs are the feedback signals of the motor as well as button inputs or CAN/UART signals. Also an LED indicator is incorporated for various status indication purposes.

3.2 Power unit

The power unit is the high power part of the application. It consists of the power MOSFETs for directing and controlling the current through the motor. Since these MOSFETs can't be driven directly from the microcontroller, FET drivers are incorporated in the power unit. The temperature sensor (Fig 7) is placed as close as possible to the power MOSFETs for monitoring and fail-safe purposes.

3.3 User Interface and communications

The user interface on the LPCXpresso Motor Control board consists of a joystick and an I^2C OLED screen, and external communication by UART or CAN if available.

3.3.1 Joystick

The joystick is used in this application to control the motor in various ways.

Table 3. Joystick functionality

Joystick function	Application response
UP	Increase RPM with 50
DOWN	Decrease RPM with 50
LEFT	START/STOP the motor
RIGHT[2]	Change motor rotation direction
CENTER[2]	START/STOP the motor

[2] Not available on the LPC11Cxx

3.3.2 UART and CAN

The communication busses UART and CAN print the actual system parameters. The current setup doesn't support system control over these two interfaces.

Bitrate setup:

- UART: 115200 Baud
- CAN: 50 kBaud

NOTE: Using the UART output through the FTDI

When you are using the UART connection though the FTDI USB to UART chip, please note that a board RESET will reset the FTDI chip as well and your COM connection on the PC could be lost!

3.4 BLDC motor

The included BLDC motor in the LPCXpresso Motor Control Kit is the 42BLF01. The 42BLF01 is a 24 V_{DC} BLDC 4 pole-pair motor with Hall sensor unit attached. The data sheet of this motor can be found in the documentation folder of this application note project.

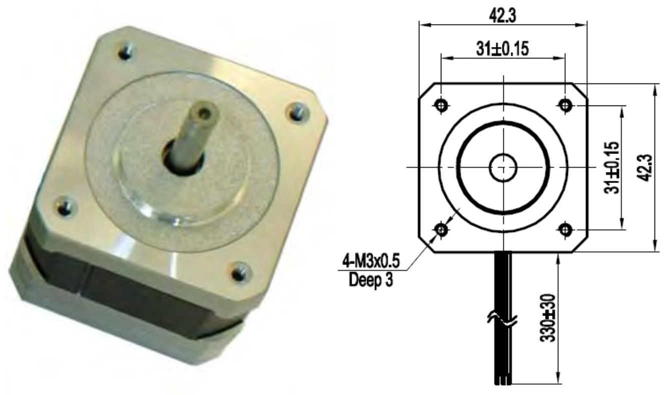


Fig 8. 42BLF01 BLDC motor

Table 4. 42BLF01 specifications

Parameter	Value
Resistance	2.2 Ω
Nominal Voltage	$24V_{DC}$
Pole pairs	4
No load speed	6000 RPM
No load current	0.5 A
Rated torque	0.063 N-m
Rated speed	4000 ±300 RPM
Back EMF Constant	4.4V/kRPM
Torque constant	0.042 N-m/A

Table 5. Motor connections

Motor		Hall Sensors	
PHASE A	Yellow	Hall A	Yellow
PHASE B	Green	Hall B	Green
PHASE C	Blue	Hall C	Blue
		+5V	Red
		GND	Black

3.5 System connections

The table below shows all connections used for this application note as depicted by Fig 7. For a complete reference of all connections of the LPCXpresso LPC11xx/LPC11Cxx to the LPCXpresso Motor Control board, please see the LPCXpresso Motor Control Kit User's Guide.

Table 6. Application connections as depicted in Fig 7

Signal	Pin name	Function Name	Signal Purpose
A_H	PIO0_1	CT32B0_MAT2	Motor phase A **high** side FET drive
A_L	PIO0_11	CT32B0_MAT3	Motor phase A **low** side FET drive
B_H	PIO0_8	CT16B0_MAT0	Motor phase B **high** side FET drive
B_L	PIO0_9	CT16B0_MAT1	Motor phase B **low** side FET drive
C_H	PIO1_1	CT32B1_MAT0	Motor phase C **high** side FET drive
C_L	PIO1_2	CT32B1_MAT1	Motor phase C **low** side FET drive
FB0	PIO2_0	PIO2_0	Hall A feedback input
FB_1	PIO2_1	PIO2_1	Hall B feedback input
FB_2	PIO2_2	PIO2_2	Hall C feedback input
A_{IN}	PIO1_0	AD1	Motor current measurement input
	PIO1_7	TXD	UART TX signal
	PIO1_6	RXD	UART RX signal
I2C	PIO0_5	SDA	I2C Data signal for the temperature sensor and OLED
I2C	PIO0_4	SCL	I2C Clock signal for the temperature sensor and OLED
RST	PIO0_0	RST	Reset button
START/STOP[1]	PIO2_6	PIO2_6	Joystick center, Start/Stop the motor
REVERSE[1]	PIO2_9	PIO2_9	Joystick right, change rotation direction
UP	PIO2_7	PIO2_7	Joystick up, increase RPM
DOWN	PIO2_8	PIO2_8	Joystick down, decrease RPM

[1] These functions are not available on the LPC11Cxx LPCXpresso boards. Please use the UP button to start the motor.

4. Software

The software written for this application note is based on the Cortex Microcontroller Software Interface Standard (CMSIS) core libraries and the standard peripheral drivers delivered with the LPCXpresso IDE. The latest version of this driver library can be found on NXP's microcontrollers website (http://www.nxp.com/microcontrollers) and is also available in the latest version of the LPCXpresso IDE.

Usage of the relevant driver library files and the application dedicated sources will be explained in this chapter.

UART and CAN can be used as a feedback mechanism and parameter control interface allowing for your preferred terminal program to be used.

4.1 Folders and files

All files needed for this application note are arranged in the following order:

Table 7. Top folder structure

Folder	Description
_Documentation	Documentation folder containing all documentation required for this application note and demo software
BLDC_Sensored	Main source folder for the BLDC_Sensored application
|--- driver	LPC1100 peripheral drivers
|--- src	Application source folder, containing all files application specific files, like the `main.c`
|--- config	Folder containing the target, system driver and application configuration files.
|--- linker	Linker files for various targets
|--- startup	Development environment specific startup file for the LPC1100
|--- Debug/Release	Folder containing compiler outputs like the .axf-file
|--- LPCXpresso_MC	LPCXpresso Motor Control kit specific driver sources
CMSISv1p30_LPC11xx	Standardized CMSIS library covering core and device specific drivers
lib_small_printf_m0	Library folder for the small footprint `printf` function.
|--- src	Library source folder
|--- Inc	Library includes folder
|--- Lib	Library compiler output folder, containing the .a-file

4.2 Program structure

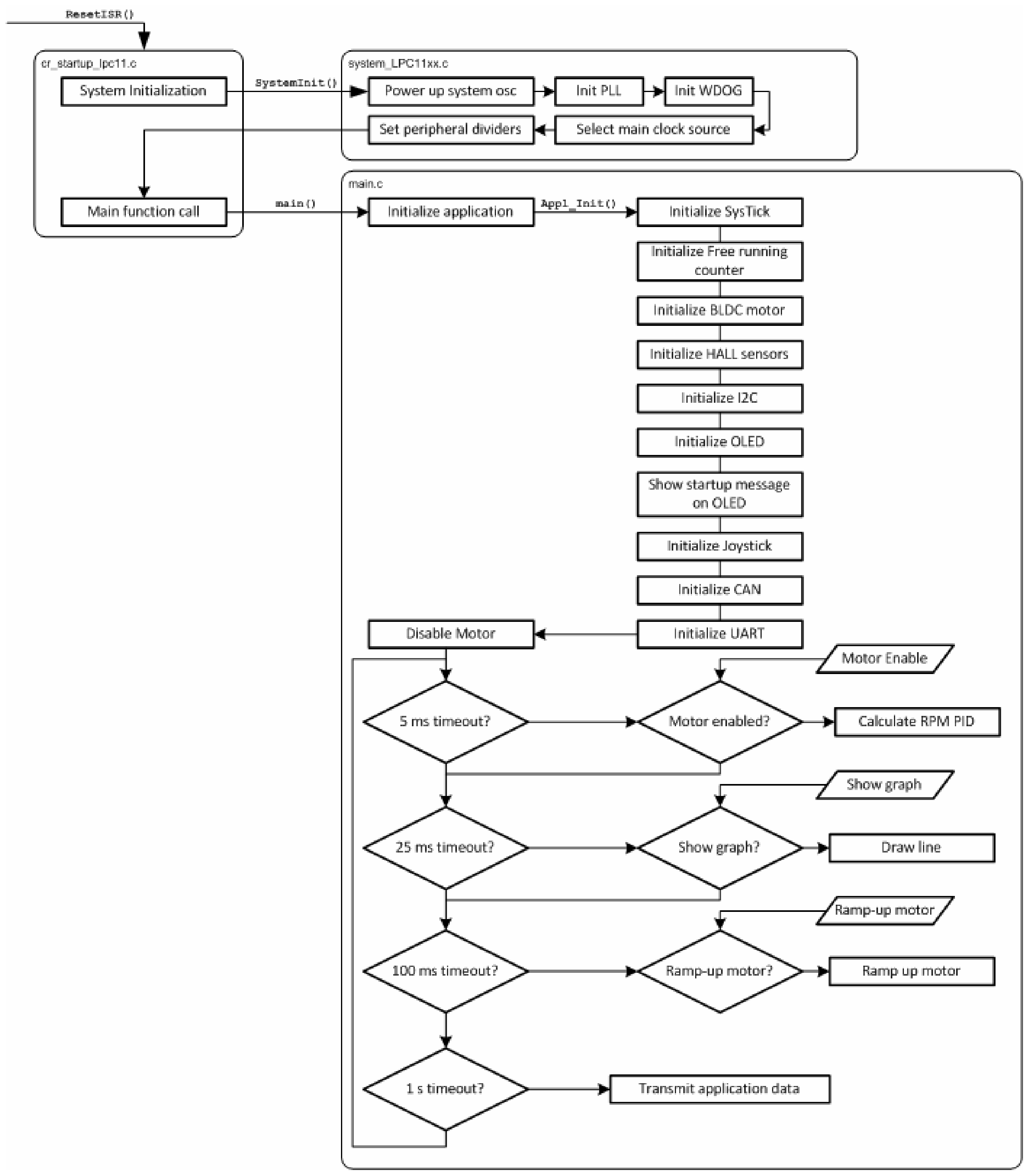


Fig 9. Main program flow diagram

4.2.1 Firmware source

The application-dedicated source files written for this application notes can be found as indicated in Table 7 in the `src` folder.

The following files are available in the src and will be described in the following sections of this chapter.

Table 8. Source files in src folder

File name	Description
`BLDC.c & BLDC.h`	Main BLDC motor control init and controlling source
`can_romhandlers.c & can_romhandlers.h`	Source files for using the LPC11Cxx CAN drivers that are place in ROM.
`comms.c & comms.h`	Communication hook to preferred comms peripheral
`ea_image.h`	Header file with the EA logo placed in a array
`gui.c & gui.h`	Source files for setting up and usage of the GUI
`handlers.c`	This source file contains all the application-related interrupt handlers.
`main.c`	Main application source file
`nxp_image.h`	Header file with the NXP logo in an array
`PID.c & PID.h`	Source files for the PID calculations

4.2.1.1 BLDC.c and BLDC.h

The BLDC.c source file incorporates the initialization and control of the BLDC motor. Table 9 describes all functions in this file.

Table 9. Functions description in BLDC.c

Function	Description
`void vBLDC_init (void)`	BLDC control initialization - Motor structure initialization with defaults - PWM initialization
`void vBLDC_Commutate (volatile uint8_t step)`	Commutate the motor to the step passed as input variable. The duty cycle of the PWM which is used in the commutation is calculated in `vPID_RPM`. ()
`void vBLDC_Stop (void)`	Stop the BLDC motor
`void vBLDC_ReadHall(void)`	Read the current Hall status and commutate the motor accordingly
`void vBLDC_CalcRPM (MOTOR_TypeDef *ptr )`	Calculate the actual RPM, using the free-running counter initialized in main.c
`void vBLDC_RampUp (MOTOR_TypeDef *ptr, uint32_t max_RPM)`	Hard-commutate the motor to give its first spin

4.2.1.2 can_romhandlers.c and can_romhandlers.h

The LPC11Cxx devices have CAN drivers incorporated in ROM as described in Chapter 2.5.

The can_romhandlers.c and .h files contain the source for getting the CAN drivers initialized and starting CAN communication.

Table 10. Functions described in can_romhandlers.c

Function	Description
`void CAN_rx( uint8_t msg_obj_num)`	CAN receive callback. This function is executed by the Callback handler after a CAN message has been received
`void CAN_tx( uint8_t msg_obj_num)`	CAN transmit callback. This function is executed by the Callback handler after a CAN message has been transmitted
`void CAN_error( uint32_t error_info)`	CAN error callback. This function is executed by the Callback handler after an error has occurred on the CAN bus.
`void CAN_IRQHandler (void)`	CAN interrupt handler. The CAN interrupt handler must be provided by the user application. It's function is to call the isr() API located in the ROM
`void vCAN_initRomHandlers`	CAN ROM handler initialize functions
`__inline void vCAN_sendObj (CAN_MSG_OBJ *can_obj)`	Transmit a CAN object

4.2.1.3 comms.c and comms.h

The comms source file gives the user the ability to hook their preferred communication peripheral to the motor control application.

Table 11. Functions description in comms.c

Function	Description
`void vDec2Ascii (uint32_t val, uint8_t *buffer, uint8_t *size)`	Convert a decimal value to an ASCII based string output.
`void vCOMMS_send (MOTOR_TypeDef *ptr)`	Main function to where the application communication could be hooked. This function allows CAN or UART to be selected , using the `USE_CAN` or `USE_UART` definition in application.h

4.2.1.4 gui.c and gui.h

The GUI files give the user the ability to set up a GUI experience on the LPCXpresso Motor Control Kit OLED screen.

These files will not be discussed further in this documentation, please see the source for more details.

4.2.1.5 handlers.c

The handlers source file contains the application specific (e.g., the Hall interrupt) handlers.

All interrupt handlers which are used differently than the predefined default handlers in the peripheral drivers should be placed in this file.

These functions are placed in the handlers.c source file:

Table 12. Functions description in handlers.c

Function	Description
`void PIOINT2_IRQHandler(void)`	The PORT2 interrupt handler in this application handles the Hall sensor and Joystick interrupts.
`void SysTick_Handler (void)`	The SysTick Handler will increase a tick counter which is used as time base for the scheduler in Main

4.2.1.6 Main.c

The `Main.c` is the application entry file. Here all application-specific initializations are made and a small pre-emptive scheduler is started for task scheduling with a 1 ms resolution. It also includes the retarget function enabling usage of the `printf` function.

Table 13. Functions description in Main.c

Function	Description
`void Call_5ms(void)`	The 5 ms scheduler Tasks: - Start PID calculation if the motor is enabled and it isn't ramping up.
`void Call_25ms(void)`	The 25 ms scheduler Tasks: - Put a pixel on the OLED in order create a graph of the RPM.
`void Call_100ms(void)`	The 100 ms scheduler Tasks: - Ramp up the motor RPM if the motor is enabled and the ramping-up variable is set TRUE. - Clear the OLED screen if requested through the `clear_screen` variable.
`void Call_1s(void)`	The 1 s scheduler Tasks: - Send the motor structure to the COMMS send handler to be transmitted. - If the temperature should be shown on the OLED, write it to the display.
`void Call_5s(void)`	Not used in this application example
`void Appl_Init(void)`	Initialize the application - Systick configuration - Free-running counter - BLDC Motor - Hall Sensor - I2C - OLED - Joystick - CAN - UART
`int main(void)`	Main program entry function - Initialize the application - Run scheduler in endless loop

4.2.1.7 PID.c and PID.h

The PID consists only of one function, **`vPID_RPM`** which actually is a PI controller. The input of this function is a pointer to a `MOTOR_TypeDef` type structure. This structure contains all variables needed for the PI controller to calculate the manipulated value or output.

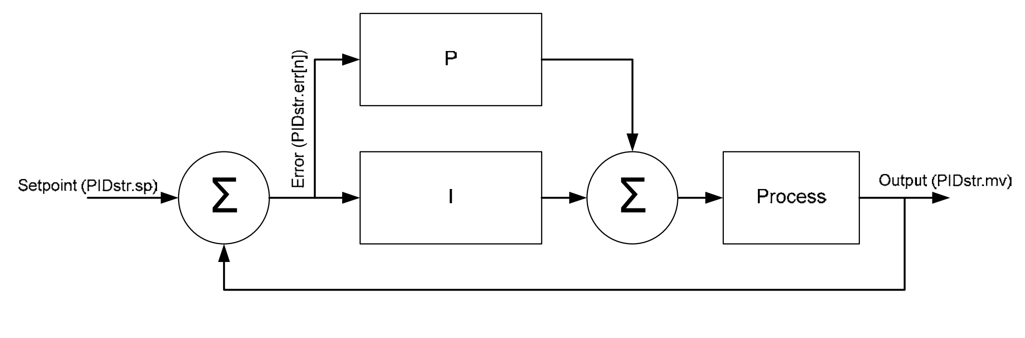


Fig 10. PI controller

4.3 Configuration

This application has various configuration defines and variables that can be set to configure the application a bit differently.

4.3.1 Microcontroller choice

First, one can select which microcontroller to use. This should be done in the properties dialog of the BLDC_Sensored project. The properties dialog can be reached through right-clicking on the project as depicted in the picture below.

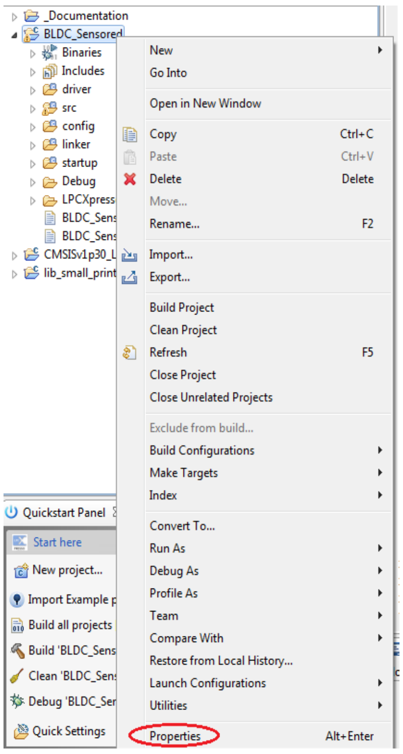


Fig 11. Open BLDC_Sensored properties

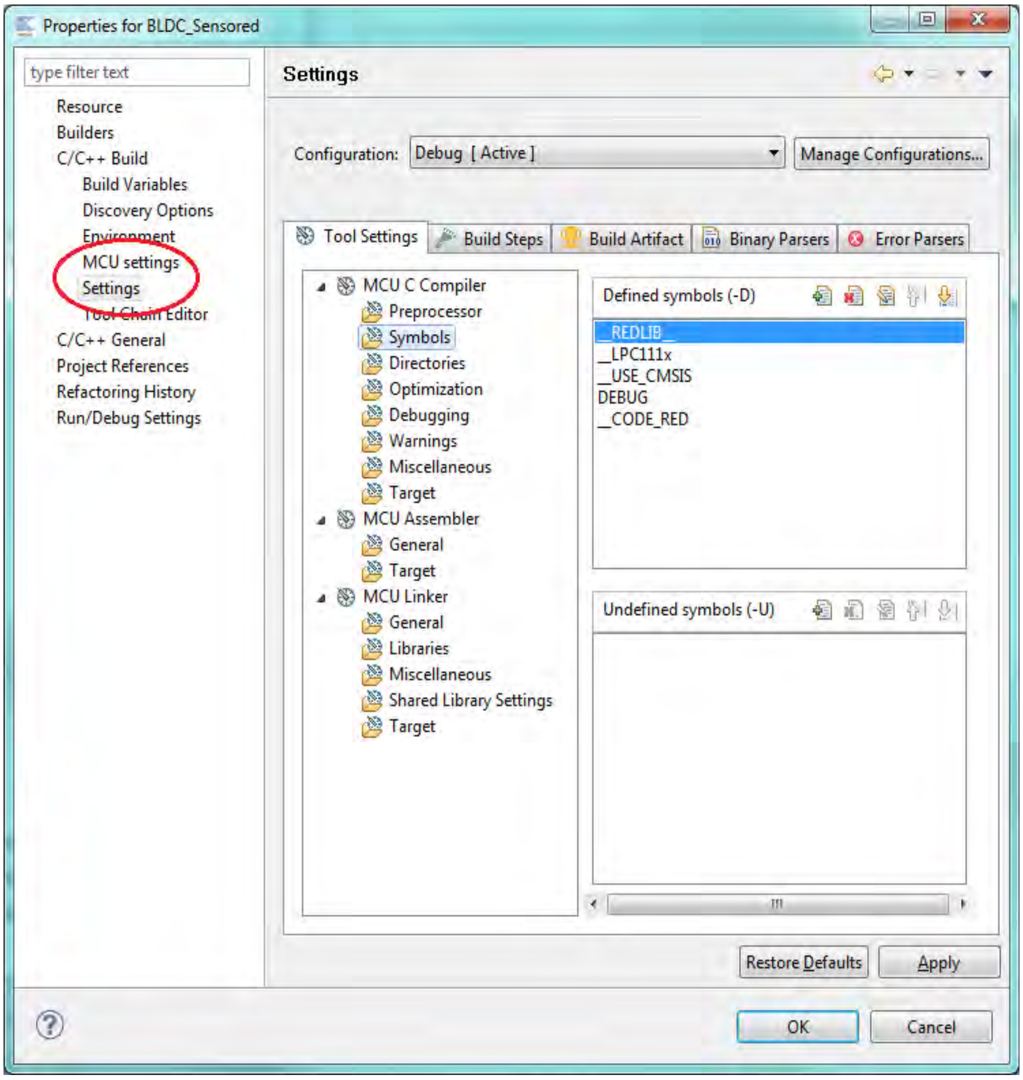
In this dialog, select *C/C++ Build* → *MCU Settings* to select the microcontroller that will be used in your application.

When opening the example project, everything is configured correctly so then you don't have to worry about this.

In the *Settings → MCU C Compiler → Symbols* dialog a global microcontroller definition is made which configure various options, e.g., enabling the CAN interface for the LPC11C24

The definition can be edited in the *Defined Symbol* part of the *Tool Settings* tab. The possible defines that are used in the application are:

- __LPC111x
- __LPC11C2x



Configuration: Debug [Active]

Defined symbols (-D)

__REDLIB__
__LPC111x
__USE_CMSIS
DEBUG
__CODE_RED

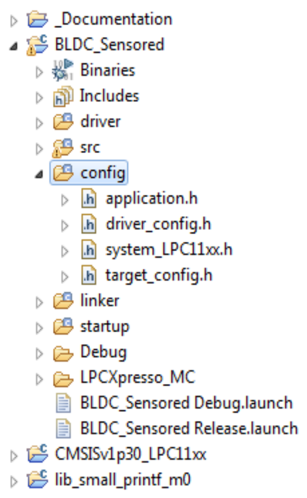
Undefined symbols (-U)

Fig 12. Configure the global define symbols

4.3.2 Configuration files

Apart from the global definitions and configurations that are described in the previous chapter, there are some configuration files that are used to setup the application.

These files can be found in the config folder of the BLDC_Sensored project, as depicted below.



```
_Documentation
BLDC_Sensored
    Binaries
    Includes
    driver
    src
    config
        application.h
        driver_config.h
        system_LPC11xx.h
        target_config.h
    linker
    startup
    Debug
    LPCXpresso_MC
    BLDC_Sensored Debug.launch
    BLDC_Sensored Release.launch
CMSISv1p30_LPC11xx
lib_small_printf_m0
```

Fig 13. ***config* folder of the BLDC_Sensored project**

4.3.2.1 Application.h

In this file some configurations can be made to setup the application, like enabling or disabling the UART interface.

Please have a look at the available configurations and definitions to get a better understanding of the project architecture.

4.3.2.2 Driver_config.h

This configuration file connects to the drivers available in the driver folder. In this file the user can setup the peripheral drivers and enable it for use in the application.

4.4 Controlling structure `Motor_TypeDef`

```
typedef volatile struct _MOTORstr {
      int32_t    Kp;           // PID Kp input value
      int32_t    Ki;           // PID Ki input value
      int32_t    Kd;           // PID Kd input value
      uint32_t   sp;           // The RPM Setpoint
      uint32_t   pv;           // the process value, used by PID
      int32_t    err[3];       // PID error array
      int32_t    IntError;     // Integration error
      int32_t    LastError;    // Previous PID calc error
      uint32_t   mv;           // Manipulated value, PID output
      uint32_t   HALstate[2];  // Current HALL state
      uint32_t   TMRval[2];    //
      uint32_t   Deadtime;     // Deadtime *NOT USED*
      uint32_t   CMT_CNT;      // Commutation counter value
      uint32_t   CMT_step;     // Current commutation step
      uint32_t   RPM;          // Actual RPM value
      uint8_t    Enable;       // Overall motor ENABLE
      uint8_t    Direction;    // Motor rotor direction, CW or CCW
      uint8_t    RampingUp;    // Motor ramping up indication
      uint32_t   Startup;      //
      uint8_t    Brake;        // Motor break indication
      uint8_t    CMT_flag;     //
      uint32_t   max_mv;       // Max manipulated value, to scale to
      uint8_t    PolePairs;    // Motor pole pairs
      uint32_t   Trap_CNT;     //
      uint16_t   Current[3];   // Motor current
      uint16_t   Voltage[3];   // Motor/System voltage
} MOTOR_TypeDef;
```

5. The brushless DC motor fundamentals

5.1 The brushless DC motor

Brushless DC (BLDC) 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 (Fig 14) are used to detect the rotor position and commutation is based on these sensor inputs.

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 14). This solves the problem of transferring 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 uses a solid-state circuit rather than a commutator/brush system.

The speed and torque of the motor depends on the strength of the magnetic field generated by the energized windings of the motor, which in turn depends on the current flow through each winding. Therefore adjusting the voltage (and current) will change the motor speed.

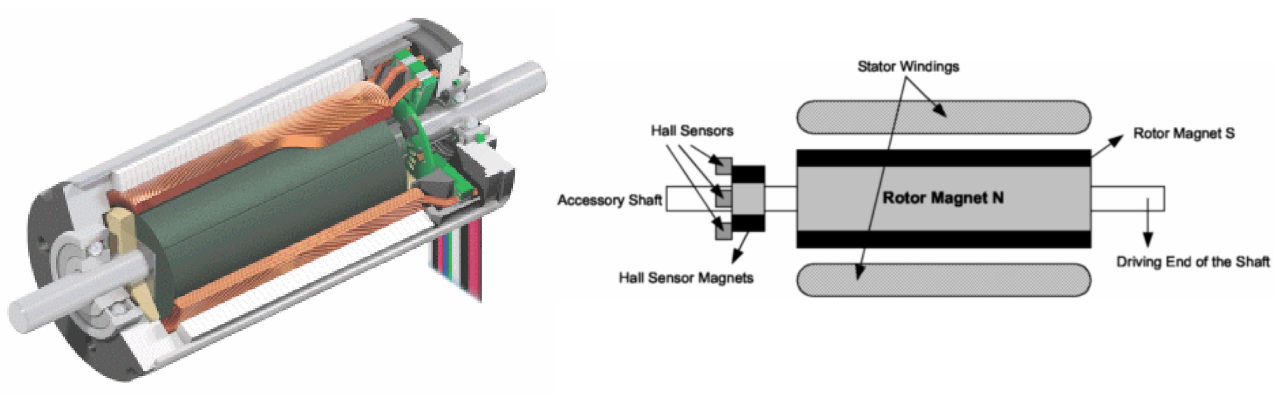


Fig 14. The brushless DC motor

5.2 Electrical commutation

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 maximize torque. Therefore, the controller needs some means of determining the rotor's orientation/position (relative to the stator coils.)

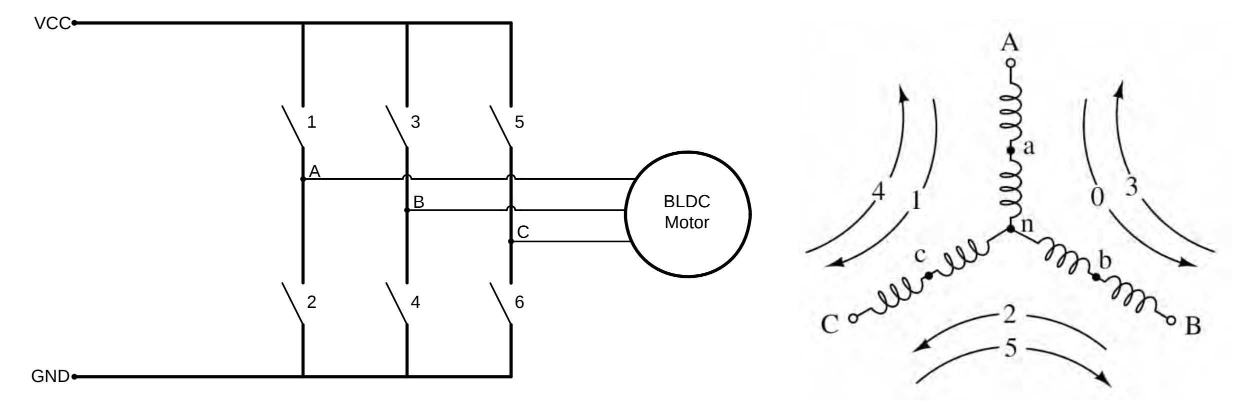


Fig 15. Three phase bridge and coil current direction

Fig 15 depicts a systematic implementation of how to drive the motor coils for correct motor rotation. The current direction through the coils determines the orientation of the stator flux. By sequentially driving or pulling the current though the coils, the rotor will be either pulled or pushed. A BLDC motor is wound in such a way that the current direction in the stator coils will cause an *electrical* revolution by applying it in six steps. As also shown in Fig 15 each phase driver is pushing or pulling current through its phase in two consecutive steps. These steps are shown in Table 14. This is called trapezoidal commutation. Fig 17 shows the relation between the six-step commutation (six Hall sensor edges H1, H2 and H3), block commutation (i_a, i_b, i_c) and trapezoidal commutation (e_a, e_b, e_c).

Table 14. Switching sequence

Sequence number	Switching interval	Phase current			Switch closed	
		A	B	C		
0	0° - 60°	+	-	OFF	1	4
1	60° - 120°	+	OFF	-	1	6
2	120° - 180°	OFF	+	-	3	6
3	180° - 240°	-	+	OFF	3	2
4	240° - 300°	-	OFF	+	5	2
5	300° - 360°	OFF	-	+	5	4

5.3 Revolution speed control

Varying the voltage across the motor coils can simply control the rotor speed. This can be achieved by pulse width modulation (PWM) of the phase voltage. By increasing or decreasing the duty-cycle, more or less current per commutation step will flow through the stator coils. This affects the stator flux and flux density, which changes the force between the rotor and stator.

This means that the rotation speed is determined by the load of the rotor, the current during each phase, and the voltage applied.

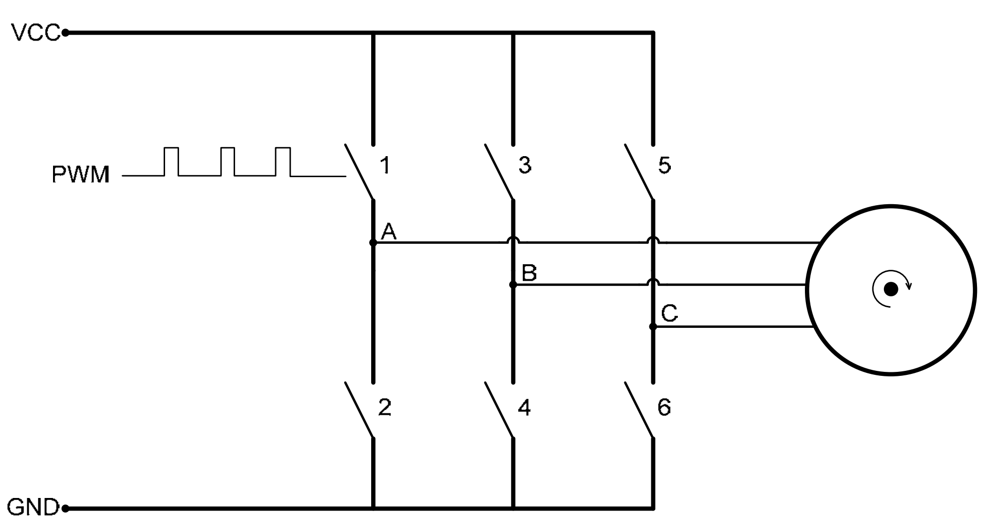


Fig 16. Speed control through PWM

5.4 Torque control

Just like speed control, torque is controlled by the amount of the current through the stator coils. For maximum torque, the angle between the stator and rotor flux should be kept at 90°. With trapezoidal commutation, the control resolution is 60° and the angle between the stator and rotor flux is from -30° to +30°, which introduces a torque ripple.

5.5 Position feedback

The rotor position feedback can be accomplished though a couple of techniques. Most commonly is the hall sensor feedback, but other techniques include using an encoder or even eliminating sensors entirely. This application note will only focus on the hall sensor feedback and will not explore sensor-less operation.

5.5.1 Hall sensor feedback

The hall sensors are placed such that they generate an edge at each switching interval as explained in Chapter 5.2. This makes it very easy to determine the current rotor orientation and to activate each phase in the right sequence.

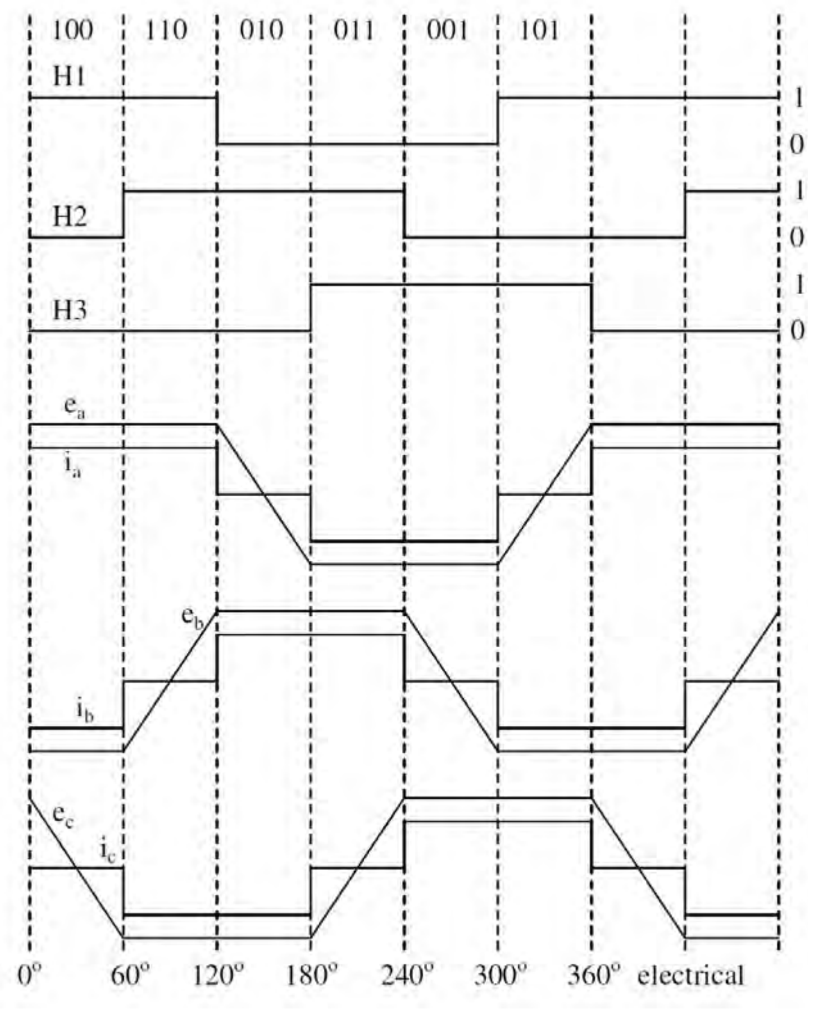


Fig 17. Trapezoidal control with Hall sensor feedback

6. References

- UM10398 - LPC11xx and LPC11Cxx User manual
- LPC1111/12/13/14 data sheet
- LPC11Cx2/Cx4 data sheet
- AN10661 – Brushless DC motor control using LPC2141
- AN10898 – BLDC motor control with LPC1700
- Cortex Microcontroller Software Interface Standard - http://www.onarm.com/
- LPCXpresso Motor Control Kit by Embedded Artists product page: http://www.embeddedartists.com/products/lpcxpresso/xpr_motor.php
- LPCXpresso_Motor_Control Users_Guide_Rev_PA4.pdf[1] - Users Guide for the LPCXpresso Motor Control kit
- 42BLF01 datasheet.pdf[1] - LPCXpresso Motor Control Kit datasheet
- LPCXpressoLPC1114revA.pdf[1] - LPC1114 LPCXpresso board schematics
- LPCXpresso LPC11C24 rev B_schematic.pdf[1] - LPC11C24 LPCXpresso board schematics
- Motor_Control_Evaluation_Board_rev_A_SCHEMATICS.pdf[1] - LPCXpresso motor control board schematics

1. These documents are available in the example code bundle. Open up the example project in the LPCXpresso IDE and look in the _Documentation folder.

7. Legal information

7.1 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.

7.2 Disclaimers

7.3 Trademarks

8. Contents

Date of release: 1 August 2011
Document identifier: AN11094