

C868

Brushless DC Trainer Description

Microcontrollers

Never stop thinking.

Revision History:	2003-09	**V 0.97**
Previous Version:	-	

Page	Subjects (major changes since last revision)

Controller Area Network (CAN): License of Robert Bosch GmbH

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all? Your feedback will help us to continuously improve the quality of this document. Please send your proposal (including a reference to this document) to:
mcdocu.comments@infineon.com



Table of Contents **Page**

1 Introduction 4

2 Software Description 5
2.1 C868 Code Description 5
2.1.1 Basic Code 5
2.1.2 Hall Sensor Mode 9
2.1.3 Hall Sensor Mode with Recording 12
2.1.4 Automatic Hall Sensor Mode 12
2.1.5 Sensorless Mode 12
2.1.5.1 Normal function 13
2.1.5.2 Start-Up 17
2.1.6 Automatic Sensorless Mode 20
2.2 Windows GUI Description 21

3 Hardware Description 23
3.1 Hall Sensor Mode Configuration 23
3.2 Sensor-less Mode Configuration 24

4 Getting Started 26
4.1 Requirements 26
4.2 Software Installation 26
4.3 Design Example 27
4.3.1 Basic Settings 27
4.3.2 Motor Configuration 28
4.3.3 Running the Motor with Hall Sensors 29
4.3.4 Configuration for Sensor-less operation 30
4.3.5 Running the Motor Sensor-less 31
4.3.6 Generate Files for Your Project 32

5 Improvements 33

6 Conclusion 34

7 References 35

8 Appendices 36
8.1 UART Protocol 36
8.2 BLDC_Params files description 38
8.3 C868 Used Ressources 40

1 Introduction

TO BE MODIFIED !!

The aim of this software is to easily develop a new application (or migrate an already existing one) for brushless-DC motors using a C868 Microcontroller.

With the software, motor system parameters such as inertia, load or dynamic behaviour can be tested practically in a matter of minutes. This step is necessary before a sensorless (without hall sensors) control method can be performed.

The motor is started using a "Boost-mode". This asynchronous mode locks onto a normal mode when sufficient feedback is read from the motor. The feedback signal is for instance in this software the Hall sensors and will be in a further version, zero-crossing events detected from Back-EMF voltage.

2 Software Description

The software is composed of two main different parts.

The first part is the C868 code, it is meant to work in Hall sensor and Sensor-less mode.

The other part is the Windows Gui which helps the user to develop a working application in a very fast way without having to code a single C line.

2.1 C868 Code Description

The C868 code can be compiled in 5 different modes.

Each mode is used for different purpose and work around the same base. To compile in one mode or the other, preprocessor defines are used. In Keil, different targets are defined and it is really easy to switch between on and another.

2.1.1 Basic Code

The program is nearly all interrupt driven, the main loop is therefore very small.

The two main functions of the “root” software is to handle serial communications and to call a function every ms (for control purposes).

a) Serial communication

A flag is checked inside the main loop to see if a command has arrived from the UART (Gui). If this is the case, the command is handled.

These commands allows the Gui interface to communicate with the program and modify important parameters. All the commands are sum-up in the next table.

Command Value	Command Name	Description
0x00	Reset	Reinitialize the system. The output stage is put into idle state and the motor is stopped.
0x10	Start	Starts the motor.
0x40	Increase Control Value	The PWM duty cycle is increased, the motor turns slowly.
0x50	Decrease Control Value	The PWM duty cycle is decreased, the motors turns faster.
0x60	Get Status	Returns the current system status.
0xC0	Set Variable	Writes to a value stored in the memory. This is used to modify system and motor parameters.
0xF0	Get Buffer	In the Recording stage, retrieves the Hall Events timings.

Table 1 : Gui UART Commands

A complete description of the commands and the protocol used is described in Appendix 1.

In answer to certain commands or to inform the Gui that something has happened, the C868 code can send a Status to the Gui. This is sumed-up in the following table.

Status Value	Status Name	Description
0x00	Status	Returns the running/stopped status of the board. During start-up this gives the actual state (positioning, accelerating or locked)
0x10	Speeds	Returns the actual speed of the motor.
0x20	Control Value	Returns the duty-cycle value. This is usefull to display it into the Gui.
0xE0	End Hall Log	Tells the Gui that acceleration data is recorded and can be retrieved (cf. Hall Sensor mode with Recording).
0xF0	Record Buffer	Returns all the recorded data.

Table 2 : Gui UART Status

b) Timer function

To derive a system clock, the Timer 0 is used. This timer overflows every 1ms and calls the *BLDC_TimerFunction*. This is described on Figure 1. The current Speed is sent out to the Gui every 100ms.

The Duty Cycle is modified in this function every 1ms. It is updated with the "*Control_Value*" variable. This is the perfect place to put a speed regulation (i.e. PID). This was not done because every motor is different and everyone wants a different control loop.

Last but not least in this function is the call to the Start-Up function. This will be described in the Sensor-less method and is used when we want to start the motor from standstill.

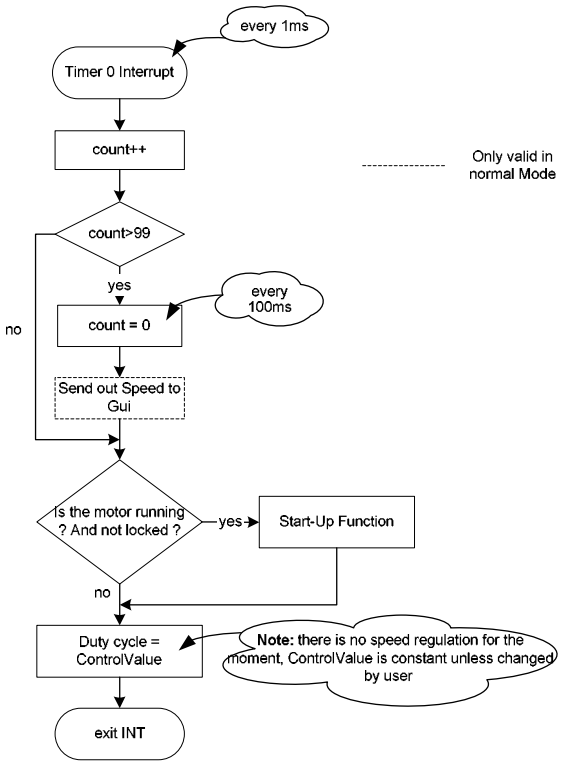

Figure 1 : Timer 0 Overflow Interrupt

2.1.2 Hall Sensor Mode

In this mode, the motor will work in the usual Hall Sensor mode with a communication with the Gui. The user is able to modify each parameter of the program and test the configuration directly on the motor.

The Hall Sensor mode is really straight-forward and uses a minimum software configuration. The control method is described on Figure 2.

Each time the motor reaches a position where the Hall Sensors change, the speed is captured in Channel 0 and the Timer T12 is reset.

The *MCMOUTSx* shadow register holds the next value for the next pattern. These values are transferred (Shadow Transfer) when Timer T12 reaches Channel 1 value which is 0 in the Hall Sensor Mode (this value is set to 30 degrees for sensor-less mode).

When Timer T12 reaches Channel 2 which is 1, an interrupt occurs and the next patterns are loaded in the shadow registers. This is the simpler case, if a T13 synchronization is needed, Channel 2 must be adjusted to at least 2 Timer 13 periods (was done in the previous version but removed as was not needed).

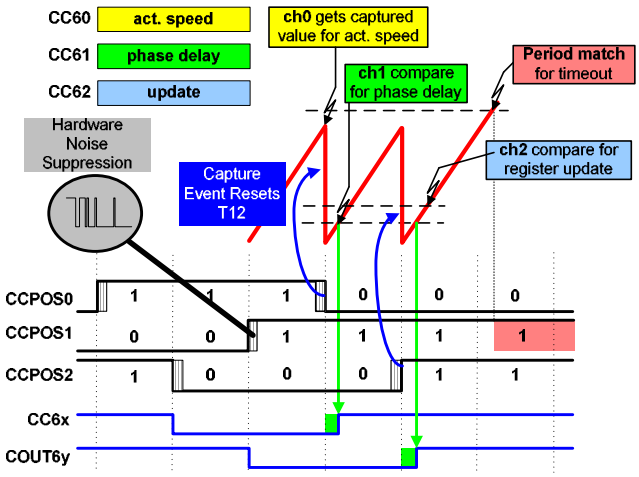

Figure 2 : Hall Sensor Mode Control Method

The Channel 2 Compare Match interrupt is mapped to the CapCom6 Node 0. It is described on Figure 3. The PWM and Hall pattern which are used for updating the

MCMOUTSx registers are stored in 4 tables (two registers with two rotating directions possible). These tables are stored in the "*BLDC_Params.c*" file. This file holds all system and motor dependent parameters. It is automatically generated by the Gui interface, this will be described later on.

The CH2 Flag is set each time this function is called. This flag is needed because of a silicon bug. Indeed, events like Channel 2 Compare match can be lost if using a high rate interrupt (T13 period match) with a multiple node configuration. In this case, the update would not be made and the motor hang (or even stop).

To prevent this bug, the Correct Hall Event interrupt is used to check if the CH2 interrupt has occurred (with the CH2 flag). If this is not the case, the update is done in the CHE. The CHE interrupt is described in Figure 4. The speed is automatically captured in Channel 0, this speed is checked whether it is too slow or too fast. In each case, the timer T12 clock divider is adjusted. This assures that the speed value is always in the range 0x6000 to 0xF800 and gives the best accuracy according to the motor speed.

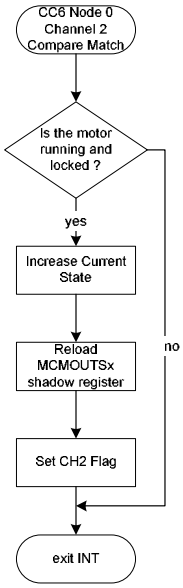

Figure 3 : Channel 2 Compare Match Interrupt

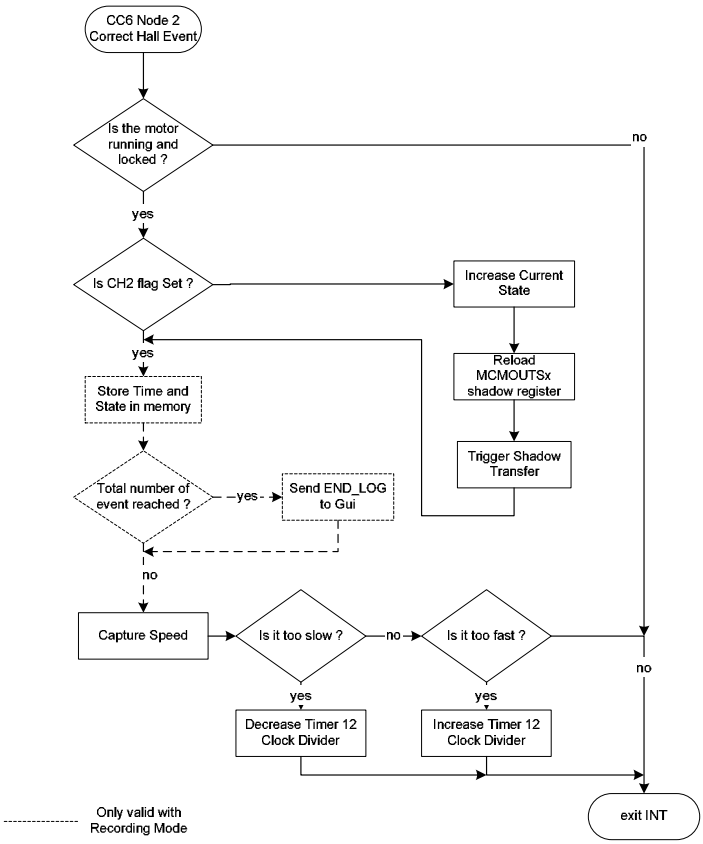

Figure 4 : Correct Hall Event Interrupt

CCU6 Node 3 is used for error handling. There's not much to say on this issue, whenever a Trap Condition is reached, the program returns to idle state (reset) and the motor is stopped. If a Wrong Hall Event occurs, nothing is done yet. This will be in charge of the developer to decide whether he wants to stop the motor or try to recover from this error.

The Timer T13 Period Match and Timer T12 Period match interrupts are not very interesting for the Hall Sensor Mode and will be described later on in the Sensor-less Mode chapter. In Hall Sensor Mode, the Timer T12 Period Match interrupt is used as a timeout and resets the system.

2.1.3 Hall Sensor Mode with Recording

This mode is exactly the same as the previous one. There is only a slight difference in Correct Hall Event interrupt. Every time this interrupts occurs, the time and the current state are stored in SRAM. This is done using memory context switching (to be able to write in code memory) and will not be described here. For further details please refer to the boot-loader source code.

The purpose of recording those states is to be able to work Sensor-less. This will be described furthermore in the next chapter.

Another difference is that the main loop doesn't send out speed information to the Gui. This is done to minimize the CPU load and record exact CHE timing. When all the events are stored in the table, a "*End Log*" status message is sent to the Gui. The Gui will then send a "*Get Buffer*" command to retrieve the buffer.

2.1.4 Automatic Hall Sensor Mode

When all the system parameters are tuned to the specific needs of the application, the GUI link can be removed and Automatic mode used. This mode uses data generated by the GUI and has no UART link. The motors starts-up directly after reset using the ramp-up parameters.

This is the base for a new project and shall be used as such.

There is no need to describe anything as it is the exact same program as the normal Hall Sensor Mode.

2.1.5 Sensorless Mode

The main aim of the project is to be able to work with a Brushless DC Sensor-less. The method used to control the motor without the help of Hall Sensor uses the BEMF (Back Electro-Motrice Force). This BEMF crosses the zero voltage every 60 degrees and is 30 degrees earlier than the switching event.

When using sensor-less control, there exist two different working modes: normal and start-up. This is explained in this chapter.

2.1.5.1 Normal function

Normal function here means the motor is turning and BEMF is sufficient enough for the Sensor-less algorithm to work. This is not the case when starting from standstill and is described next.

The phase voltage is sampled every T13 period and checked against a threshold (equivalent to the zero voltage). This threshold was defined in the first 3 T13 periods by measuring the DC-Link voltage (via a fourth ADC channel). This threshold is set to half of this voltage. Whenever this threshold is reached, the speed is captured (i.e. Timer T12 value) and the timer is reset (manually). The speed is equivalent to 60 degrees; it is therefore easy to configure Channel 1 to 30 degrees as the switching should occur 30 degrees after the Zero Crossing Event (ZCE). This method is summarized in Figure 5.

**********Say something about the fact that the method is a bit more complicated !!!

(acceleration problem)

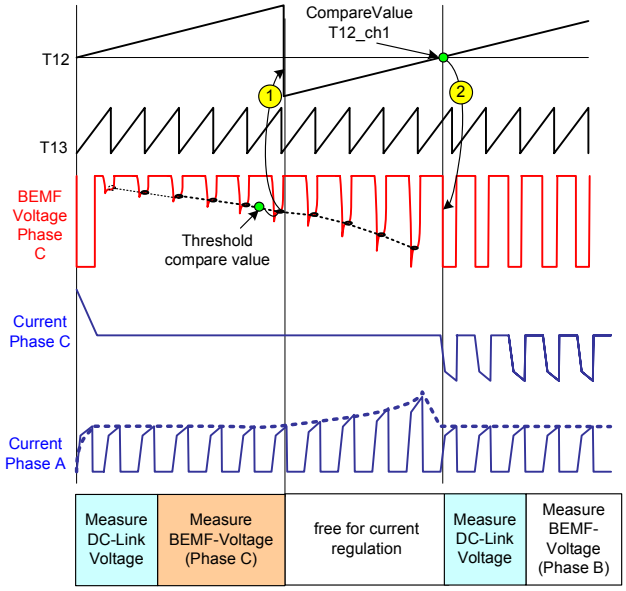

Figure 5 : Sensor-less Method Description

The algorithm is started every time a switching on the output has occured, this is detected by Timer 12 Channel 1 Compare match interrupt and is described on Figure 6.

When the switching is detected, the ADC is setup to measure the DC-Link voltage. Then the Sensorless state machine is started (*Sensorless_State* = 1). An additional counter (*NonSamplingPeriodCount*) was added to prevent from false ZCE detection at start-up, this will be described in the next chapter. Finally in this function, the *ZCE_Detected* flag is checked. If it was false, no zero crossing was detected in the last period and the *ZCE_Count* is reset. This is used for start-up.

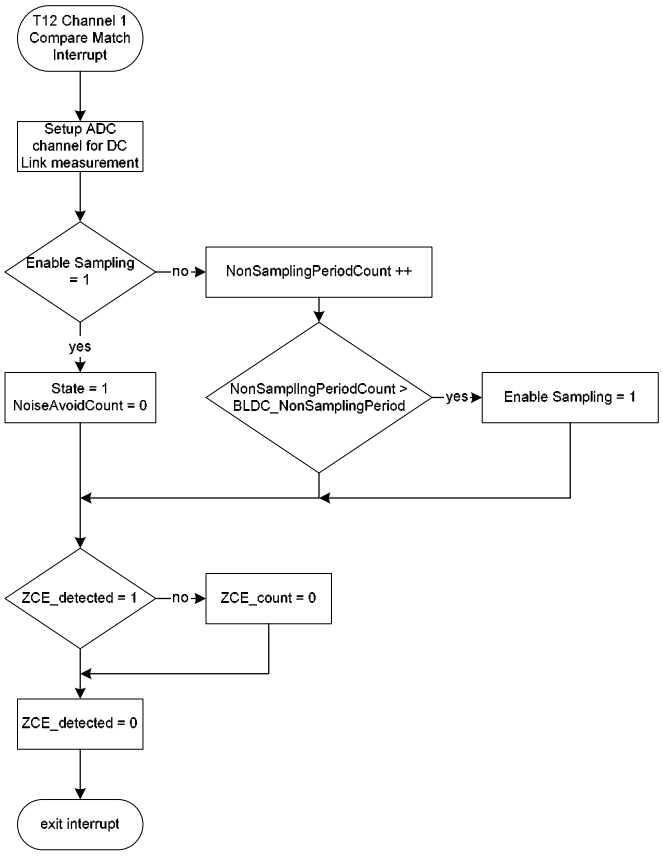

Figure 6 : T12 Channel 1 Compare Match Interrupt

The Sensorless state machine is handled within the Timer 13 Period match interrupt. This interrupt occurs according to the PWM output. The ADC is triggered by the T13 period match and the converted value is therefore available in this function. The state machine is described on Figure 7.

The first state is used to measure the DC Link voltage. This state is started just after switching, there could be some noise on the DC Link due to this. The *NoiseAvoid_Count* counter is used to avaoid this noise. The DC Link is effectively measured 5 periods after switching. The ADC is then setup for BEMF measurement, this is extracted from a table stored in the *"BLDC_Params"* file. The direction (upwards or downwards) is also retrieved and the next state set according to this value: State 2 for upwards and State 4 for downwards.

The algorithm have to detect a zero crossing event, therefore the BEMF is first tested to be below the threshold (for upwards BEMF). If this is the case, the state is set to State 3. In this state, the BEMF is sampled until it is above the threshold. When this is the case, a Zero Crossing Event is generated. The ZCE function is described on Figure 8. States 4 and 5 are used the same way as 2 and 3 but for downwards BEMF detection.

This function is fairly simple: in normal mode (not start-up), the speed is captured and Channel 1 set to half of the value(30 degrees). The *MCMOUT* registers are prepared for the next state. In case the speed is too slow or too fasr, the clock divider is adjusted just as in the Hall Sensors Mode.

The Accelerating mode is explained is the next chapter, start-up.

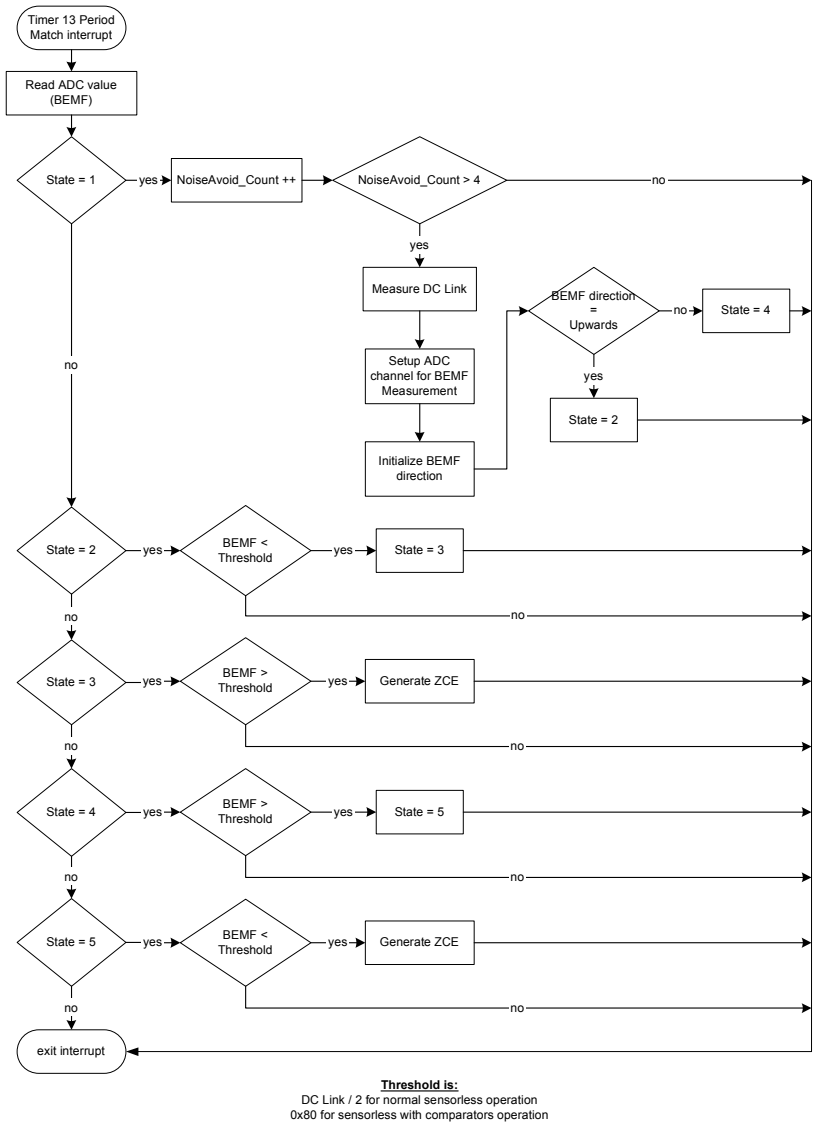

Figure 7 : Timer T13 Period Match Interrupt

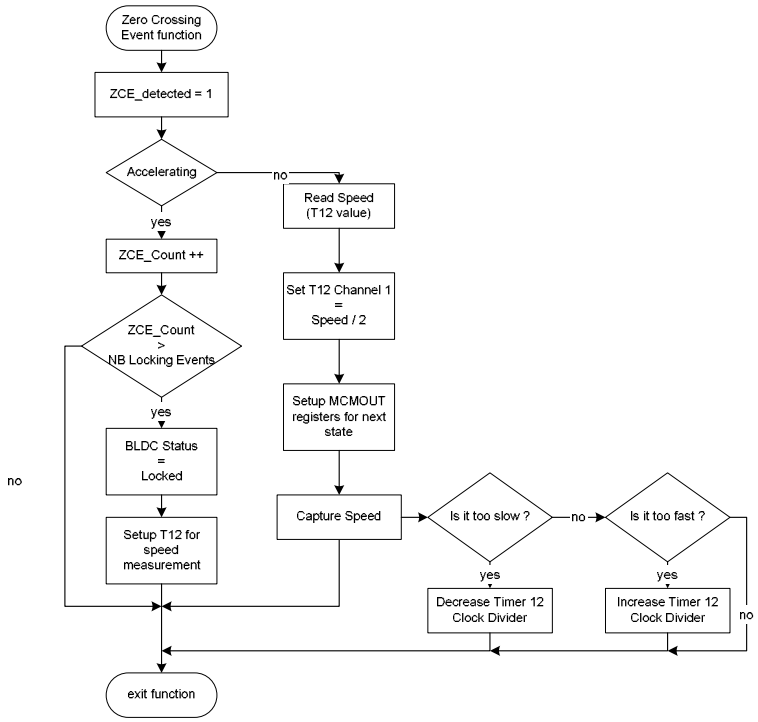

Figure 8 : Zero Crossing Event function

2.1.5.2 Start-Up

When the motor is at standstill or running at very low speed, the BEMF is very small or inexistent. It is therefore impossible to use it to detect the ZCE. A way should be found to start the motor until the BEMF is big enough so ZCE could be detected.

This is called the start-up function in this software.

First the motor is put in a known and well defined position. It is then accelerated using a recorded table (from Hall Sensor mode with recording). Until the end of this phase, the BEMF should increase enough in order to detect ZCE and switch to the normal mode.

The start-up function is a state machine and is described on Figure 9. This is common for Hall Sensor mode and Sensor-less mode. In Hall Sensor mode, there is no

accelerating part as we can use the signals from the Hall Sensors directly. This is not the case when working Sensor-less.

1. Positioning:

   The CAPCOM is configured to energize two coils. The duty cycle is progressively increased. This has the effect of putting the motor in a precise position which is always the same (at least electrically speaking).

2. Accelerating:

   The Timer T12 is configured with a period value extracted from a table stored in the "*BLDC_Params.c*". Each time the timer overflows, the period register is updated with a new value. These period values are extracted from the Hall Sensor Mode with recording. In this mode, the motor was started normally using Hall Sensors. The exact instant where the CHE occurred were recorded and stored in this table. Each time the timer overflows, the switching on the output occurs. This simulates the behavior the motor had with Hall Sensors but without it. This method, however very simple is quite efficient. It unfortunately doesn't work perfectly on high inertia motors. The Timer 12 period match interrupt is described on Figure 10.

3. Locked:

   This state is reached either directly from the Positioning state in Hall Sensor mode or when the BEMF was big enough to detect sufficient ZCE. It waits for the Ramp-up to finish and switches to normal mode.

During the Accelerating Phase, the same algorithm for ZCE detection is running. If a ZCE is detected, a counter is incremented (described on Figure 8). Once this counter reaches a specific value (user defined); the programs switches to normal mode. The Timer T12 is then setup for speed measurement (maximum period value 0xFFFF).

To prevent a lot of false ZCE at the beginning (when the motor turns very slowly), a counter is used: *NonSamplingPeriodCount*. After a few simulated periods (Timer 12 periods), this counter reaches a threshold (user defined) and the *EnableSampling* flag is set; allowing the ZCE to be generated.

We see that the user can define two values for the accelerating phase:

BLDC_Nb_Locking_Events and *BLDC_NonSamplingPeriod*

The user will have to play with those values to find the better combination.

Typical values are 2-5 for *BLDC_Nb_Locking_Events* and *5-10* for *BLDC_NonSamplingPeriod.*

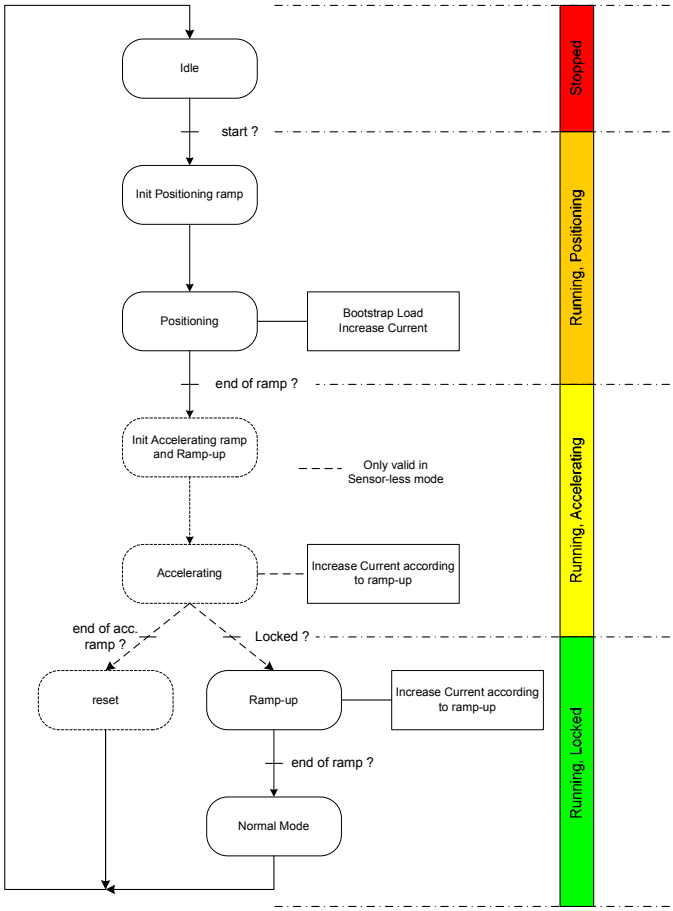

Figure 9 : Start-Up function state machine

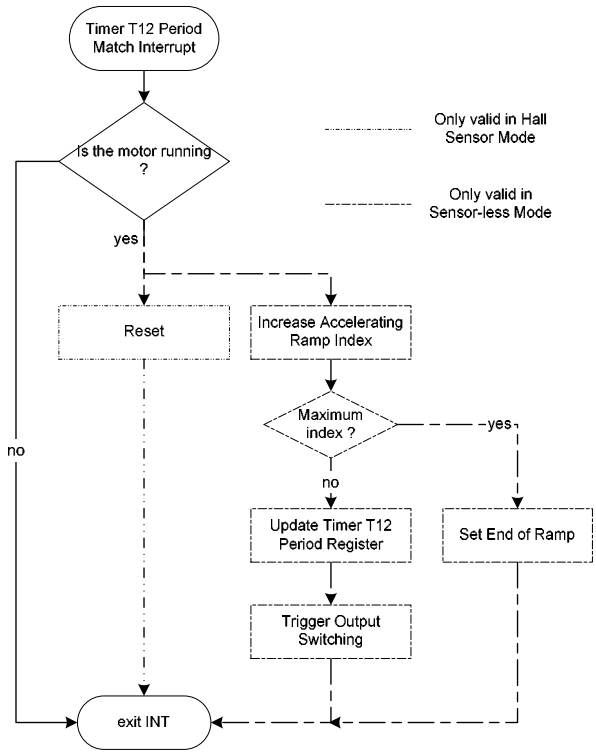

Figure 10 : Timer T12 period match interrupt

2.1.6 Automatic Sensorless Mode

The Automatic Sensor-less mode has exactly the same behavior as the Hall Sensors Automatic mode. The motor starts-up and ramps-up, using the specified parameters.

No connection to the Gui is needed.

2.2 Windows GUI Description

The PC Gui software was developed with Microsoft Visual C++ v6.0. You also need the *MSComm* control (Serial port handling).

The main idea of this interface is to be able to control each parameter of the Brushless DC software. By controlling those parameters, it is then really easy to test a motor for proper operation.

It is organized around multiple windows:

- Main:

  This is where you see the status of the board, and where you can control its behavior. In particular you control the speed of the motor and its rotating direction.

- System properties:

  This window controls the system behavior of the board, i.e. what kind of pattern is used for the motor and what kind of modulation is used. It is really easy to change a parameter according to your specific needs. Another important choice is the Mode (Hall Sensor or Sensor-less).

- Motor properties:

  In this window, you can control the parameters specific to the motor and especially the positioning, accelerating and ramp-up.

- Debug:

  This is not really useful and is only used for debug purposes 9will be removed in further versions).

- Settings:

  This is an important window to setup the link you are using (COM1 or COM2). It is also very important to use the correct path to the C868 files which will be downloaded to the board.

The main window is *CBLDCSheet* (a *CPropertySheet*) which holds all the windows:

- *CBLDCMainPage*
- *CBLDCSystemPropertiesPage*
- *CBLDCMotorPropertiesPage*
- *CBLDCDebugPage*
- *CBLDCSettingsPage*

The serial port is part of the main window and sends all commands (and receives status). Whenever a status message is received, the associated data is dispatched to the corresponding windows which hold the value to be displayed.

Every information shown on the windows are part of one big class called "*CBLDC*". This class handles all the motor specific routines. The buttons created on the main window call the routines inside this class:

- Load/Save Parameters
- Reload C868 (with a .hex file)
- Reset
- Start
- Get Status
- Change Speed
- Generate *BLDC_Params* files

3 Hardware Description

Because the program can function with or without Hall Sensors, different hardware topologies are needed for each mode. This is described in this chapter.

3.1 Hall Sensor Mode Configuration

In the Hall Sensor Mode, the basic well known topology is used. This topology is shown on Figure 11.
The Hall Sensors coming from the motor are connected directly to the CCPOS inputs of the C868. Hall Sensors are usually open collector transistors, it is easy to interface it with the microcontroller. A pull-up resistor connected to the +3.3V is enough (don't connect the resistor to +12V or more !).
The CTRAP hardware protection is connected to a comparator which measures the current in the bridge. In case of an over current, the bridge is put in a passive state immediately.

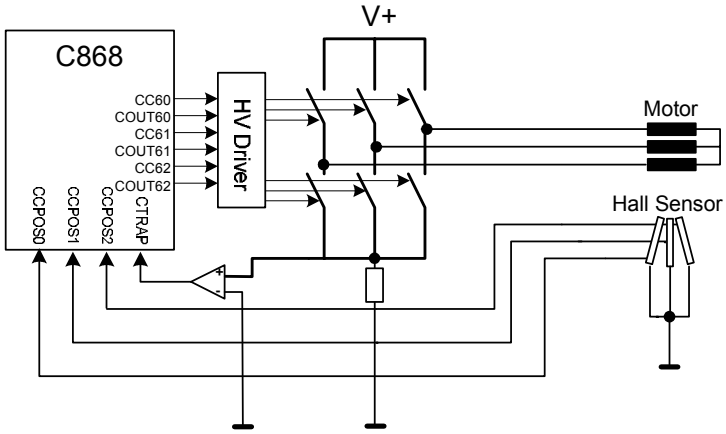

Figure 11 : Hall Sensor Mode Topology

3.2 Sensor-less Mode Configuration

In Sensor-less Mode, the rotor position is derived from the BEMF which is measured from the phase voltage. The ADC is used to do so and therefore must be connected to the phase voltage. Because the motor is usually driven with 12V or 24V we need an adapter stage which is in this case a simple voltage divider. This is shown on Figure 12.

This method would not work properly with high voltage (at least supposed). Indeed the input capacitance of the ADC combined with the high impedance of the voltage divider stage (because of high voltage for current limitation) creates an RC filter with a high time constant. This destroys the signal and makes it impossible to use.

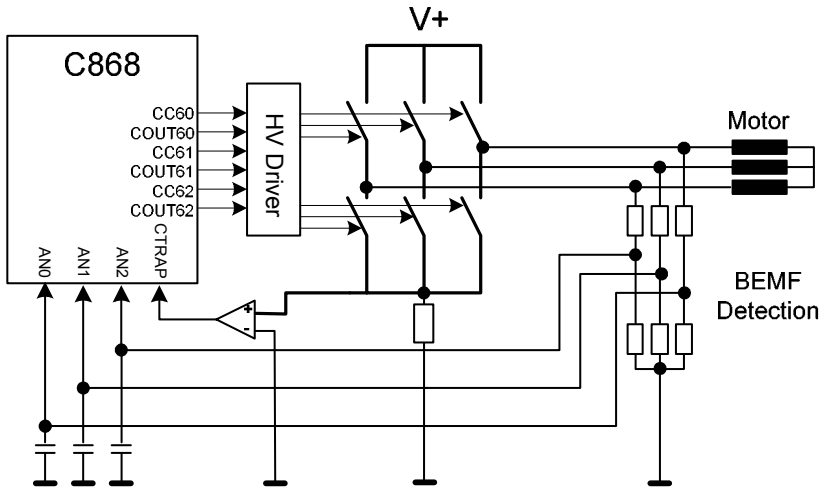

Figure 12 : Sensor-less Mode Topology

The resistors should be chosen such as the DC Link (maximum voltage on the phases) falls into a 3.3V range (ADC input). For example, using a 24V motor:

R1 = 80kΩ

R2 = 13kΩ

→ 24V is converted to 3.3V

Another sensor-less topology can be used with the program. This topology uses comparators, this is shown on Figure 13.

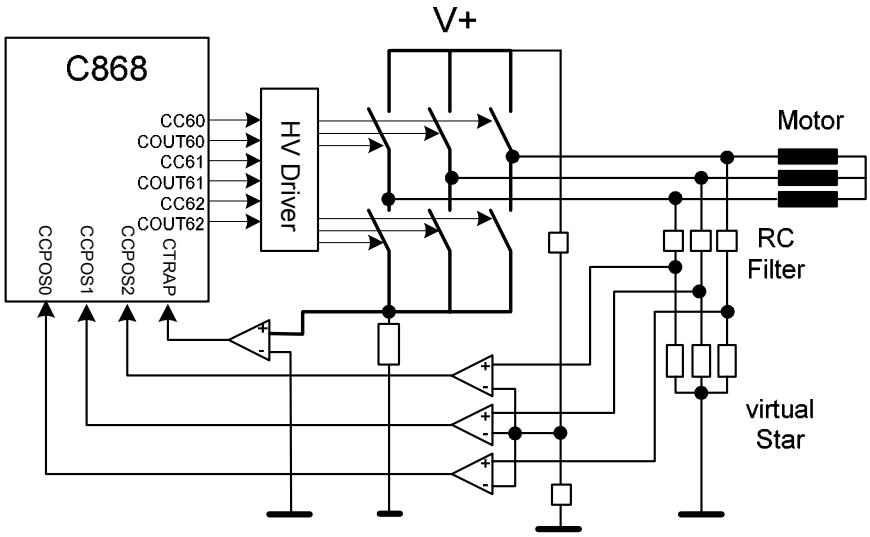

Figure 13 : Sensor-less Mode Topology with Comparators

The voltage dividers for the phase voltage is computed the same way as before.

The voltage divider for the comparators must be the same divided by 2. Indeed we want to detect the zero crossing which is DC Link / 2.

For example, using a 24V motor:

R1 = 80kΩ

R2 = 6kΩ

→ 24V is converted to 1.67V (3.3V/2)

4 Getting Started

4.1 Requirements

To be able to run the software you need Windows 98/NT/2000/XP.

You need a board with a C868. This can either be reference designs from Infineon Technologies or a specific design. Currently only IR driver IC is supported, further version will support TLE driver IC.

To communicate with the board you need a free serial port.

And of course you need a brushless DC motor connected to your board.

4.2 Software Installation

To run the GUI you need to install the serial port communication control.

First copy all the files in your working directory and double click on "register.bat" in the bin directory, this will install the serial port control if not yet installed.

If you want to compile the generated code for you own application you will need to install Keil µVision for C868. The full version might be needed as some parts are bigger than the evaluation size limit.

4.3 Design Example

In this example we will try to run a brushless DC sensor-less. We will undergo several steps. This explains the methodology you should use in order to work sensor-less.

4.3.1 Basic Settings

- Launch "*BLDC Trainer.exe*"

  The Gui is shown onFigure 14. It is composed of different tabs (on the upper part) and buttons on the right side for communicating with the board.

  You can load and save parameters for your specific motor into files.



Figure 14 : Graphical User Interface

- Push load button and select the example file:

  "*your-directory*\BLDC v0.96\bin\bldc_maxon.parameters"

  The data stored in this file is using absolute path, therefore you need to change this path to point to the right one and save it.

- Go to the settings tab and select the com port you are using (don't change speed...)

- Click on select to choose the location of the C868 files and select:

  "*your-directory*\BLDC v0.96\bin\c868\bldc_hallsensors.hex"

- click save and select (or any other, this is your choice) :

  "*your-directory*\BLDC v0.96\bin\bldc_maxon.parameters"

You are now ready to start playing with BLDC Trainer.

4.3.2 Motor Configuration

In this steps, you will configure the system to work with your specific motor and specific hardware. You can save this data to your file and start with a new configuration at any time.

- Go to the System properties tab (this is the tab shown on the Figure 14)
- Change the Hall patterns and PWM patterns according to the motor you are using. The PWM patterns should be the same as the one already configured. The hall patterns are different from a motor to another one. To change a value, double-click in the upper window. No action will be taken if you click elsewhere.
- Select the bridge modulation, inactive state and so on according to your hardware. For example on the High-Voltage reference design board, the driver IC (IR) is low active. The modulation on this board can be chosen either high or low.
- Go to the Motor properties tab
- Select a positioning ramp.

  If you have a low inertia motor, you can choose exponential. In case you have a high inertia, you should choose linear.

  The duty cycle should be chosen so the motor will be put in a known position without too much current.

  The Settling Time is also important with high inertia motors.

  You must try different values in order to find the right parameters.

- Select the ramp-up you want to apply to your motor.

  In this case I choose the same value as the positioning sequence. This will give no ramp-up, this is for test purposes.

  We'll see the accelerating part later on.

4.3.3 Running the Motor with Hall Sensors

The hall sensors should be connected to the C868 the usual way (refer to a reference design for this). An example connection was shown on Figure 11.

- Select the Hall sensor mode in the System Properties tab
- Go to the Main tab and push "Reload C868" (you must press reset on your board). This will upload the C868 software to the board.
- Press Start

  If your parameters were correct, the motor should start to move into a specific position and then start to turn.

  If it does not, you will have to go back to the last chapter and try different parameters for the PWM and Hall patterns or the positioning.

  If the motor turns, a graph will plot the current speed of the motor. It is easy to change the speed using the “Duty Cycle” buttons.

  You can also turns “clockwise” or “counter-clockwise” (you need to stop and restart the motor).

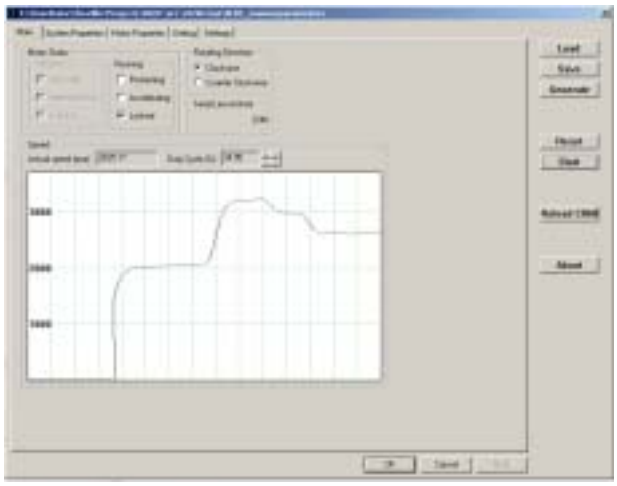

Figure 15: Gui When Running the motor

4.3.4 Configuration for Sensor-less operation

Now we want to run sensor-less, first you need to keep the hall sensor configuration.

- Go to system properties and select “record” mode with hall sensors and push reload (hit reset on the board).

  This will upload a software which records the accelerating phase of the motor.

  The same behavior will be given to the motor when running sensor-less. This is needed because the BEMF is too small when motor is at standstill or low speed, therefore we need to accelerate it before.
- Go to the Motor properties tab and push “clear table” and “clear curves”
- Push start, the motor will turn just as it did before.

  After a while you should see a red curve in the graph. This is the curve recorded from the motor itself. This curve is extracted from the Hall Sensors; each time a new CHE occurs, the time is recorded and stored in a table inside the C868.
- Push “Use curve”

A blue curve appears, this is the curve which will be stored in the C868 for sensor-less operation.

You are now ready to go sensor-less.

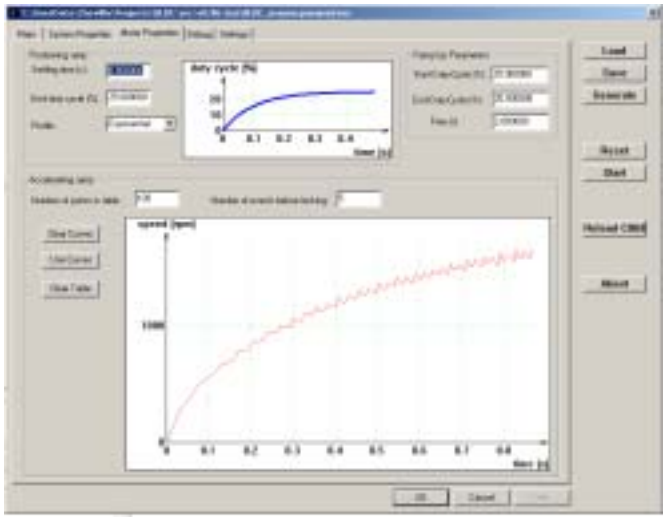

Figure 16 : Typical Recorded Curve

4.3.5 Running the Motor Sensor-less

The sensor-less software needs to measure the BEMF from the Motor phases, therefore you need to add voltage dividers as shown on Figure 12. These are connected in place of the hall sensors to the SAME inputs.

You need to compute the values so that the full voltage extent falls into a 3V range in the end (i.e. a 12V motor is connected through a 3/12 voltage divider). You also need to connect the DC Link voltage through the same kind of voltage divider to channel 3 (this should be removed in the next version).

To sum up you need 4 voltage dividers connected to the 4 ADC inputs. 3 of them for BEMF measurement connected to the motor phases and 1 for DC link measurement.

- Go to the system properties tab and select Sensor-less (without record)
- Push reload and try start !

  The motor should turn, if not try again. Please be careful not to change parameters like the DC Link Voltage. Indeed the Sensor-less control is very sensible with this issue.

4.3.6 Generate Files for Your Project

Now that you have played with the Gui interface and that your motor works in the mode you have chosen (with or without sensors), you want to have your own application. This is possible using one of the “Automatic” mode of the code.

This mode only uses data from the “*BLDC_Params*” files and do not use any serial link. You can either use the provided project to test it or copy it and develop your application around it.

The “*BLDC_Params*” files are automatically generated by the Gui:

- You can now push the generate button

  This creates 2 files in the “bin\C868” directory (where you selected the Hex files for the C868).
- Copy these files into the “src\C868” directory and double click on BLDC.dpt

  You must have Keil µVision installed to do so.
- You can select the “Sensorless, Automatic” target and load it.

  Once the code is loaded, the motor should start automatically and ramp-up as you selected before.

5 Improvements

TO BE MODIFIED !!

This software is really basic and works only for motors with Hall sensors. The aim of this software is to be able to run the motor in a sensorless mode. This will be the next version.

For the moment, no speed or current regulation is built in the software. It was decided to do so as every motor are different and would need different types of regulation. A very basic speed loop control will be embedded in the next version.

Positioning and accelerating tables are transfered all the time before a start command. In the next version, the tables will be put in the code memory space so there will be no need to download them every time. They're will still be modifiable though.

6 Conclusion

TO BE MODIFIED !!

Dfsdf

7 References

Loader gui source code, Arno Rabenstein, Infineon Technologies, Munich, 2003

Bootloader gui source code, ?, Infineon Technologies, Munich, 20??

Indirect Sensors for electric drives, M. Jufer, Swiss Federal Institute of Technology, Switzerland, 1995

8 Appendices

8.1 UART Protocol

a) Commands: Gui to C868

The C868 uses a 8 bytes buffer to receive commands. The first byte is used for synchronization and the next byte is the command/size. Depending on the size field, the software waits for all bytes to be received. This is summarized in Table 2.

All the commands are described in Table 3. For the “Set Variable” command, a 16 bits variable is considered as a 2 bytes array.

Byte n°	Name	Description	Value	
1	Synch	Used to synchronize the receiver	0xFF	
2	CommSize	Identifies the command and the number of bytes associated with it	command	size
3	Data n	Corresponding data for the command	?	
…	…			
3+size-1	Data n+size-1		?	

Table 2 : UART Commands protocol

Name	Byte n°				Description
	2	3	4	5	
Reset	0x00	-	-	-	Resets the motor
Start	0x10	-	-	-	Starts the motor
IncCtrlVal	0x42	IncrH	IncrL	-	Increment the duty cycle
DecCtrlVal	0x52	DecrH	DecrL	-	Decrement the duty cycle
GetStatus	0x60	-	-	-	Retrieves the current state (cf. Status)
SetVariable	0xc4	Index of variable	Array index in bytes	Val	Initialize a parameter variable (cf. BLDC_Params description)
GetBuff	0xf0	-	-	-	Retrieves the data buffer

Table 3 : UART Commands Description

c) Status: C868 to Gui

The C868 can send back data to the Gui. It uses a very similar protocol but is able to send more than 128 bytes. If the “size” field is less than 0x0F, it uses the same protocol as the command one. If this field is equal to 0x0F then the next 2 bytes defines a 16 bits size. This is summarized in Table 4.

The Status messages are described in Table 5.

Byte n°	Name	Description	Value
1	Synch	Used to synchronize the Gui	0xFF
2	StatusSize	Identifies the Status and the number of bytes associated with it	Status | 0x0F
3	SizeMSB	Most significant part of the 16 bits size	sizeH
4	SizeLSB	Less significant part of the 16 bits size	sizeL
5	Data n	Corresponding data for the status	?
…	…		
5+size-1	Data n +(sizeH<<8)+sizeL-1		?

Table 4 : UART Status protocol

Name	Byte n°					Description	
	2	3	4	5	6		
Status	0x02	State	Info	-	-	Sends the current state of the software:	
						State = 0 : Motor is stopped	State = 1 : Motor is running
						Info: 0 = by user 1 = by hardware 2 = error 3 = time-out	Info: 0 = start 1 = positioning 2= accelerating 3 = locked
Speed	0x10	SpdH	SpdL	T12Clk	-	Current Motor Speed T12Clk is the current clock divider	
CtrlVal	0x22	CtrlH	CtrlL	-	-	Current Value of the duty cycle	
EndLog	0xe0	-	-	-	-	Tells the Gui the log has finished	
DataBuffer	0xff	SizeH	SizeL	Data n	…	Retrieves the current state (cf. Status)	

Table 5 : UART Status Description

8.2 BLDC_Params files description

The BLDC_Params files (.c and .h) hold all the parameters of the software. Depending of the motor and the software you will be using, this file might change. It is generated automatically by the Gui and should not be modified by the user.

The files are used when compiling „Automatic“ parts. When using the Gui, the file are not used (because you need to recompile). Therefore to be able to modify it, the Gui issues „Set Variable“ commands.

All the parameters are summarized in the next table.

Name	Size (bytes)	"Set Variable" Command		Description
		Index	Array Index	
HallPatternsClockWise	6	0	0-5	Holds the values that will be stored in the MCMOUTH register. Controls the Hall Sensors state machine. Refer to chapter 4.8.4.2 of the C868 User's Manual.
HallPatternsCounterClockWise	6	0	5-11	Same as previously but for opposite rotation operation.
PWMPatternsClockWise	6	1	0-5	Holds the values that will be stored in the MCMOUTL register. Controls the behavior of the bridge according to the Hall Sensor state. Refer to chapter 4.8.4.2 of the C868 User's Manual.
PWMPatternsCounterClockWise	6	1	5-11	Same as previously but for opposite rotation operation.
PositioningTable	100	2	0-99	Holds duty cycle values which will be used during the positioning phase.
PositioningTableCountLimit	1	3	-	Count in ms between each values of the table.
AcceleratingTable	200	4	0-199	Each value of the table is a Timer T12 count (clock divider default is 7). Timer is loaded with this value and when an overflow occurs, the output pattern is changed and timer reloaded with next value.

BLDC_PassiveState	1	6	-	Holds bridge passive level value stored in PSLRL register. Refer to chapter 4.8.4.1 of the C868 User's Manual.
BridgeModulation	1	7	-	Defines the type of bridge modulation. 0 is Low-side, 1 is High-side.
BLDC_Nb_Locking_Events	1	8	-	Number of ZCE events before locking onto normal mode.
BLDC_Direction_ClockWise	bit	9	-	Defines the motor direction: 0 is Counter Clockwise, 1 is Clockwise
BEMFADCChannel	6	10	0-5	Defines which ADC channel to use for BEMF measurement according to the current state.
BEMFDirection	6	11	0-5	Defines in which direction the BEMF is going, up (1) or down (0).
RampUpTable	100	13	0-99	Defines the duty-cycle while in the accelerating phase (start-up).
RampUpCountLimit	1	14	-	Count in ms between each values of the table.
Timer13_PeriodValue	2	16	0-1	Defines the PWM frequency along with the clock divider
Timer13_ClockDivider	1	18	-	-

Table 6 : BLDC Parameters description

8.3 C868 Used Ressources

	Hall Sensors Mode			Sensorless Mode	
Ressource	Normal	Record	Automatic	Normal	Automatic
Watchdog Timer					
Timer 0	✓*(2)*	✓*(2)*	✓*(2)*	✓*(2)*	✓*(2)*
Timer 1	✓*(3)*	✓*(3)*		✓*(3)*	
Timer 2					
UART	✓	✓		✓	
CAPCOM6	✓	✓	✓	✓	✓
AN0/CCPOS0/INT0	✓*(5)*	✓*(5)*	✓*(5)*	✓*(6)*	✓*(6)*
AN1/CCPOS1/INT1	✓*(5)*	✓*(5)*	✓*(5)*	✓*(6)*	✓*(6)*
AN2/CCPOS2/INT2	✓*(5)*	✓*(5)*	✓*(5)*	✓*(6)*	✓*(6)*
AN3				✓*(4)*	✓*(4)*
AN4					
P3.0	✓*(1)*	✓*(1)*	✓*(1)*	✓*(1)*	✓*(1)*
P3.1	✓*(1)*	✓*(1)*	✓*(1)*	✓*(1)*	✓*(1)*
P3.2	✓*(1)*	✓*(1)*	✓*(1)*	✓*(1)*	✓*(1)*
P3.3	✓*(1)*	✓*(1)*	✓*(1)*	✓*(1)*	✓*(1)*
P3.4	✓*(1)*	✓*(1)*	✓*(1)*	✓*(1)*	✓*(1)*
P3.5	✓*(1)*	✓*(1)*	✓*(1)*	✓*(1)*	✓*(1)*
P3.6	✓*(1)*	✓*(1)*	✓*(1)*	✓*(1)*	✓*(1)*
P3.7					
P1.0	✓*(7)*	✓*(7)*		✓*(7)*	
P1.1	✓*(8)*	✓*(8)*	✓*(8)*	✓*(8)*	✓*(8)*
P1.2	✓*(8)*	✓*(8)*	✓*(8)*	✓*(8)*	✓*(8)*
P1.3	✓*(8)*	✓*(8)*	✓*(8)*	✓*(8)*	✓*(8)*
P1.4	✓*(7)*	✓*(7)*		✓*(7)*	

(1) Used by the CAPCOM6
(2) Used as a 1ms timer function
(3) Used as a baudrate generator
(4) Used to measure the DC Link voltage
(5) Used as CCPOS inputs
(6) Used as analog inputs
(7) UART Pins
(8) EEPROM Pins

Infineon goes for Business Excellence

“Business excellence means intelligent approaches and clearly defined processes, which are both constantly under review and ultimately lead to good operating results.
Better operating results and business excellence mean less idleness and wastefulness for all of us, more professional success, more accurate information, a better overview and, thereby, less frustration and more satisfaction.”

Dr. Ulrich Schumacher

http://www.infineon.com

Published by Infineon Technologies AG