User manual

Getting started with the AEK-MOT-3P99081 CAN-controlled brushless motor evaluation board based on SPC560P and L9908

Introduction

The AEK-MOT-3P99081 evaluation board is based on the SPC560P Pictus 32-bit MCU and the L9908 gate driver allowing the control of 6 N-channel FETs for brushless motors in automotive applications.

The AEK-MOT-3P99081 supports independent encoder inputs and Hall sensors to detect and control motor speed.

The L9908 independently controls each N-channel FET through a dedicated PWM input pin. L9908 configuration, protection and diagnostic functions are controlled via SPI by the SPC560P microcontroller.

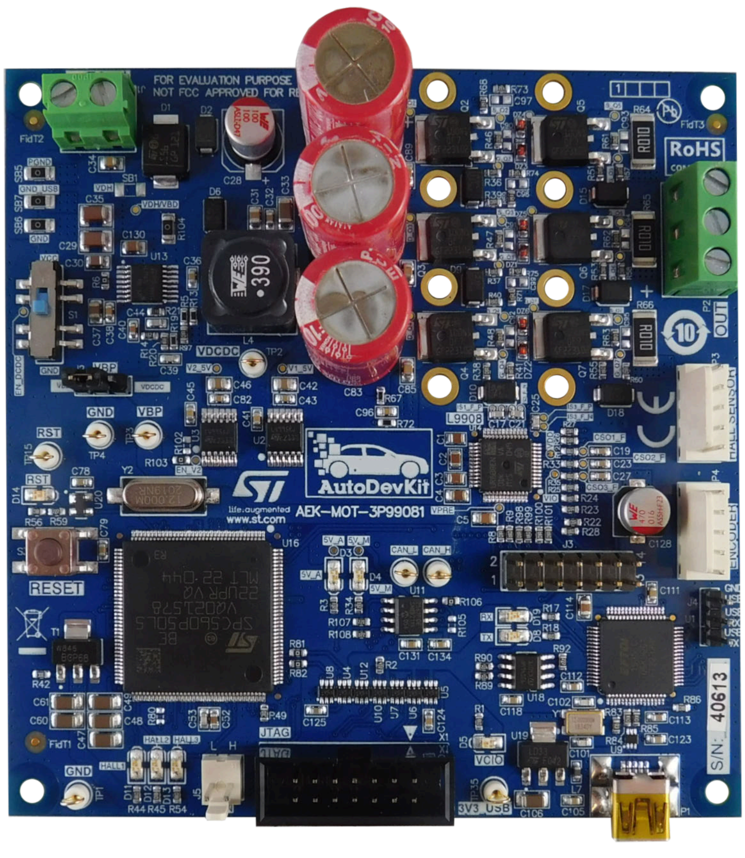
Firmware is preloaded and can be externally driven via CAN bus. The STSW-AUTODEVKIT contains a CAN bus driving example based on SPC58 Chorus 4M, named “SPC58ECxx_RLA_MainEcuForBLDCControl-L9908 - Test Application”. In the project folder, a readme file explains how to use the demo which works only with a BLDC motor with Hall sensors.

To change the motor characteristics or the control firmware on the SPC560P50L5, you need to install the SPC5-MCTK-LIB motor control plug-in in SPC5-STUDIO.

Once the motor control plug-in is installed, select the “SPC560Pxx_RLA_AEK_MOT_3P99081_3Phase_Motor_Control_L9908_via_CAN ” demo and make your customizations in the Motor Control Component section.

Update the Motor Settings section according to the motor used and, if the motor sensing is not based on Hall sensors, update also the type of sensor used in the Speed Sensor Selection menu of the Drive Management section.

Figure 1. AEK-MOT-3P99081 evaluation board



1 Getting started

1.1 System overview

The AEK-MOT-3P99081 evaluation board belongs to the AutoDevKit ecosystem.

In its simplest configuration, the board is considered as a standalone motor drive controllable via CAN by a host microcontroller. An example of this configuration is included in the AutoDevKit example libraries. The source code example demonstrates the CAN protocol setting and usage for an SPC58EC Chorus microcontroller acting as communication master towards an AEK-MOT-3P99081 board connected to a BLDC motor acting as a slave.

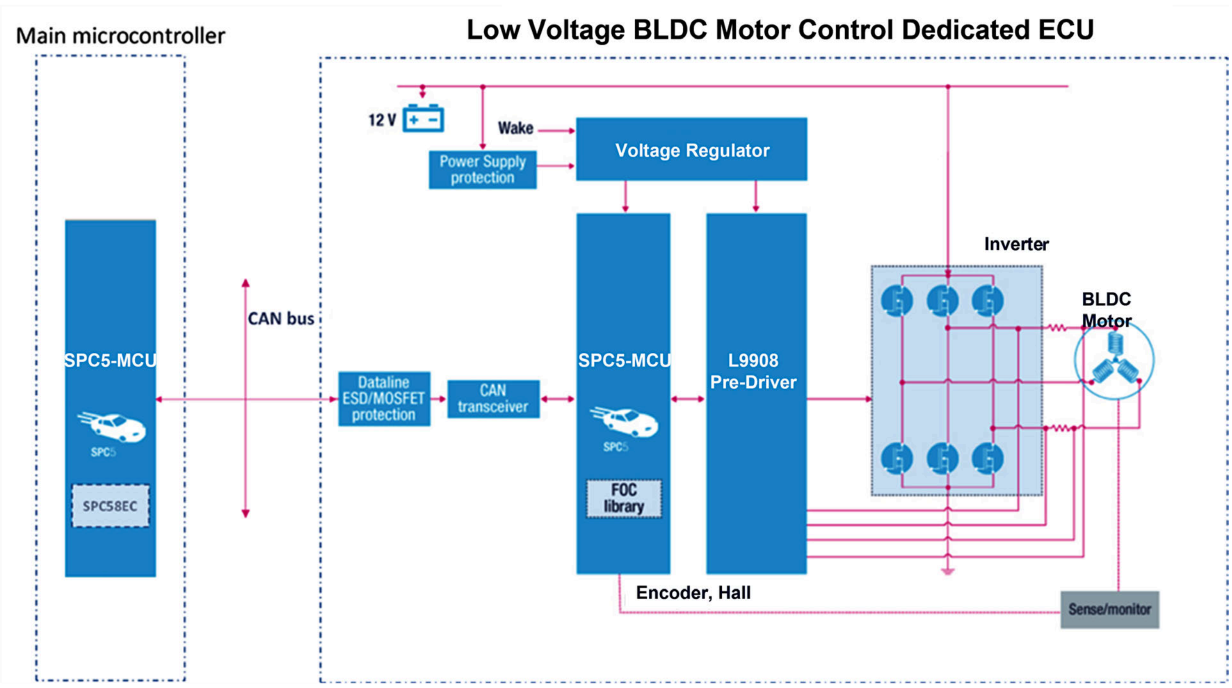
The figure below highlights the CAN communication architecture underlying the distributed system solution to control BLDC motors.

Important:
In the current temporary implementation of low voltage BLDC motor control, only one AEK-MOT-3P99081 can be connected as the employed SIDs are the same and decoding of the motor ID is not implemented.

In this architecture, the master board is represented by an AEK-MCU-C4MLIT1, while the slave system board is represented by the AEK-MOT-3P99081. The latter manages the BLDC motor control application through a dedicated firmware based on field-oriented control (FOC) algorithm.

The dedicated CAN communication protocol allows real-time data exchange between master and slave boards. You can send commands to start/stop the BLDC motor, to increase/decrease the BLDC motor target speed, to reverse the BLDC motor rotation direction. You can also make the system receive error messages when a fault condition occurs.

Figure 2. System overview describing a generic high-level system block diagram for Hall-effect sensor three-phase BLDC motor control



Warning:
The AEK-MOT-3P99081 evaluation board has not to be used in a vehicle as it is designed for R&D laboratory use only.

1.2 Features

- Hosts an automotive-grade L9908 gate driver to control 6 N-channel FETs and SPC560P Pictus 32-bit automotive microcontroller
- Works with 12 V, 24 V and 48 V battery bus
- Independent encoder inputs and Hall sensors
- Gate driver configurable through dedicated SPI bus
- CAN bus interface for remote control

1.3 Board components

The AEK-MOT-3P99081 is composed of:

- **Microcontroller unit** (SPC560P50L5) - 32-bit power architecture MCU for automotive chassis and safety applications. The microcontroller can be flashed either using the integrated programmer/debugger (FTDI module, FT2232HL) or using an adapter connected to the JTAG connector
- **Gate driver** (L9908) - automotive three-phase motor gate driver fully supporting all features related to BLDC motor control applications. The device features three differential high accuracy current monitors for ground referred current measurements
- **Power stage** (inverter) - made up of six automotive-grade N-channel 100 V,6.8 mΩ typ., 80 A STripFET F7 power MOSFETs in a DPAK package (STD105N10F7AG). It uses input bus voltage VBD as main power supply.

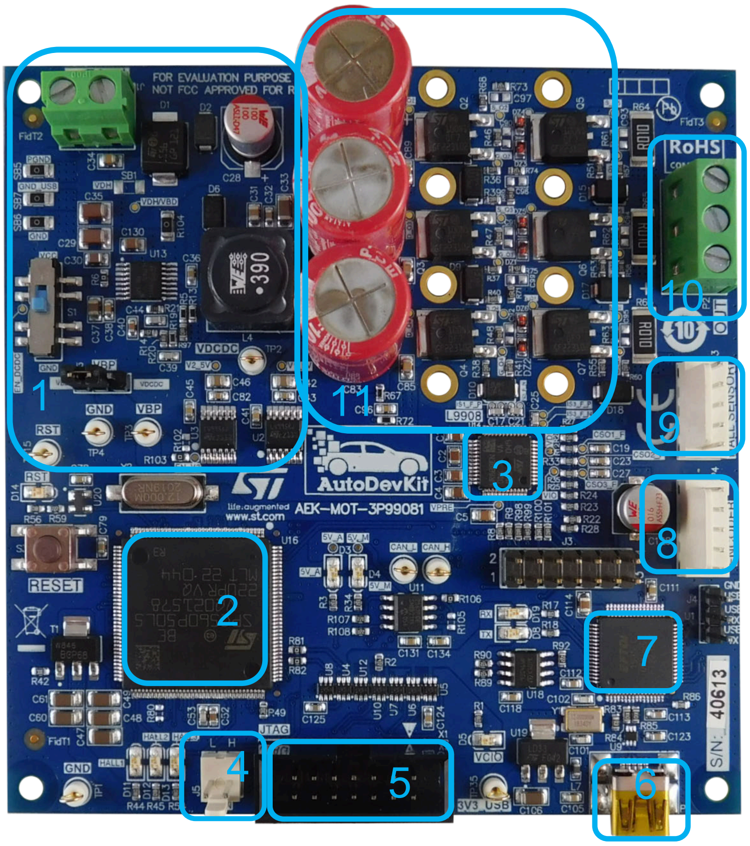
Note: Due to component shortage, some AEK-MOT-3P99081 could mount the feature-equivalent industrial-grade STD100N10F7.

- **Power supply unit** - for safety functions related to battery polarity inversion and reliable generation of board power supplies. It can be divided in: DC-DC converter (L7987L), used to generate 10.9 V reference voltage starting from the input bus voltage (VBD), and U2 (L4995RJ) and U3 (L4995AJ) voltage regulators, used to generate 5 V to power, respectively, the microcontroller unit and the gate driver.
- **High-speed CAN flexible data rate transceiver** (L9616) - to manage CAN communication with the master board (AEK-MCU-C4MLIT1).

The BLDC motor three phases are connected to the P2 screw connector. Encoder sensors, used to track the BLDC motor rotor position, are connected to the P4 header. Instead, alternative Hall sensors to track the rotor position of the BLDC motor can be connected to the P3 header.

Figure 3. **AEK-MOT-3P99081 components**

1. Power supply unit
2. SPC560P50L5 microcontroller
3. L9908 gate-driver
4. CAN connector
5. JTAG connector
6. Mini-USB port
7. FTDI module
8. Encoder sensor header
9. Hall sensor header
10. Three-phase BLDC motor screw connector
11. Power stage inverter



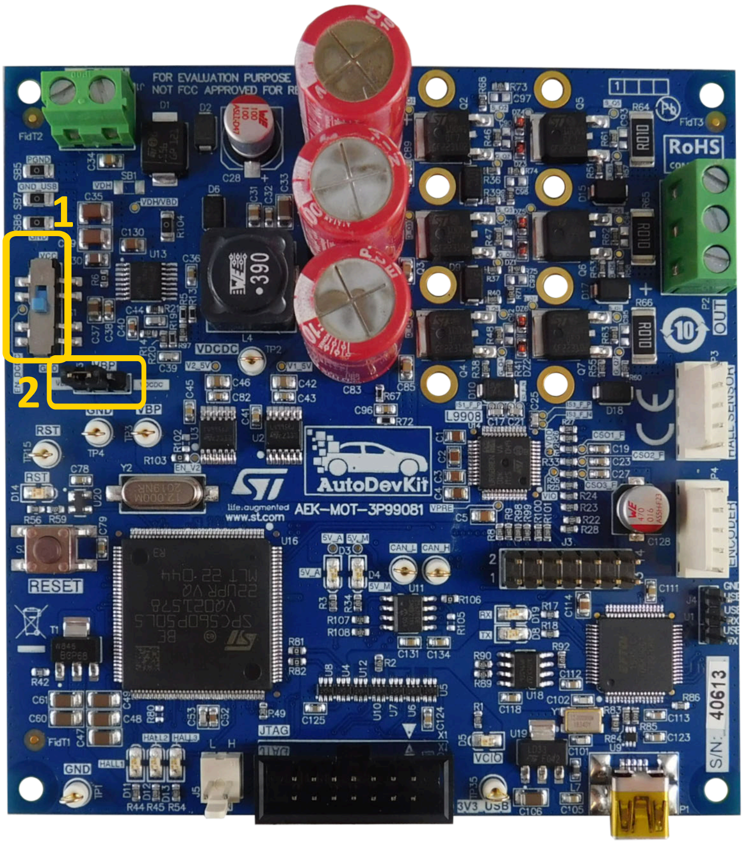
2 How to set up the AEK-MOT-3P99081

Step 1. Set switch S1 to VCC to enable the DC-DC converter (U13).

Step 2. Set jumper J2 by choosing one of the following options:
- for 12 V and 24 V systems: J2 short-circuits VBP with VBD (pins 1 and 2);
- for a 48 V system: J2 short-circuits VBP with VDCDC (pins 2 and 3).

Figure 4. AEK-MOT-3P99081 switch and jumper

1. S1 switch
2. J2 jumper

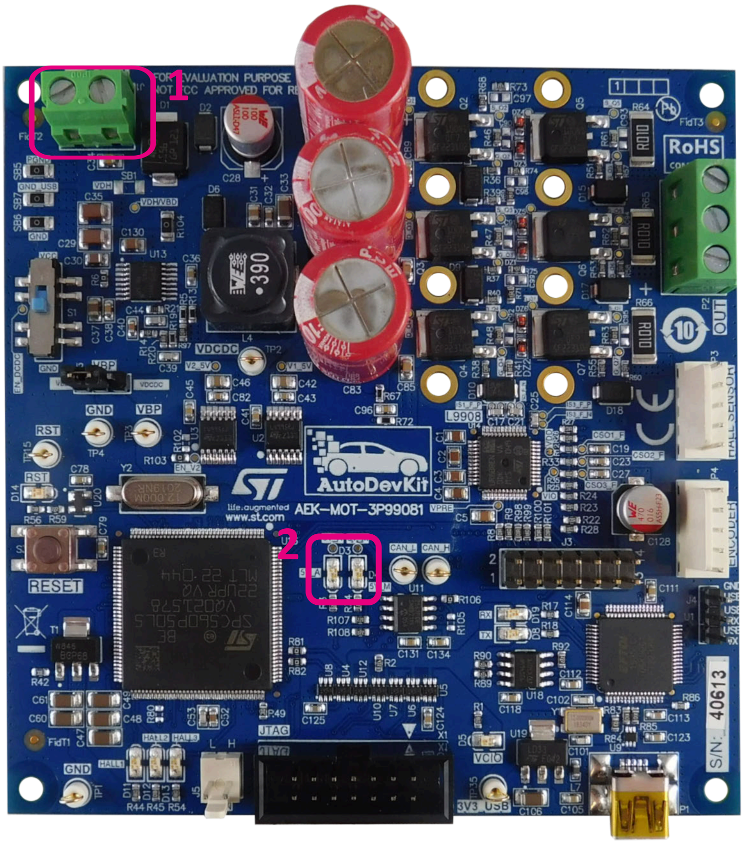


Step 3. Through J1 screw connector, provide external power supply to the AEK-MOT-3P99081 board by connecting (–) to 0 V and (+) to, for example, 14 V, which is the typical value of battery voltage (VB) in a 12 V system.

When the board is powered using the external power supply, LED D3, D4, D5 turn on.

Figure 5. AEK-MOT-3P99081 power supply connection and LEDs

1. Power supply connector
2. LEDs



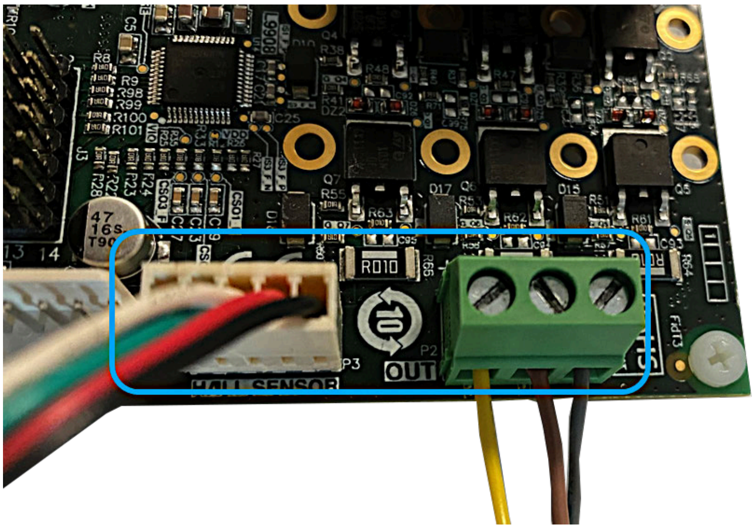
Step 4. Connect the three phases of the motor (i.e., phases U, V, W) to OUT1, OUT2, OUT3 of P2 screw connector.

In this demo application, a Nanotec motor (model: DF45L024048-A2, Brushless DC Servo Motor) is used.

Step 5. Connect Hall-effect sensors, used to track rotor positions for Nanotec BLDC motor, in P3 connector.

Important:
- *U phase (the grey wire in the figure below) has to be linked to OUT1*
- *V phase (the brown wire in the figure below) has to be linked to OUT2*
- *W phase (the yellow wire in the figure below) has to be linked to OUT3*
- *GND pin of the Hall-effect sensor cable (coming from the Nanotec BLDC motor) has to be linked to the first pin of the P3 connector close to P2 screw connector*

Figure 6. AEK-MOT-3P99081 three-phase BLDC motor and Hall-effect sensor connections



3 How to set up the AEK-MCU-C4MLIT1

Step 1. For jumpers JP37, JP36 and JP35, close pin 2 and 3.

Step 2. For jumper JP34, close pin 1 and 2.

Step 3. For jumpers JP6, JP7 and JP2, close pin 2 and 3.

Step 4. For jumper JP32, close pin 1 and 2.

Step 5. For jumpers JP42, JP43, JP44, JP47, JP49, JP45, JP46 and JP48, close the two pins.

Figure 7. AEK-MCU-C4MLIT1 JP37, JP36, JP35 and JP34 jumpers

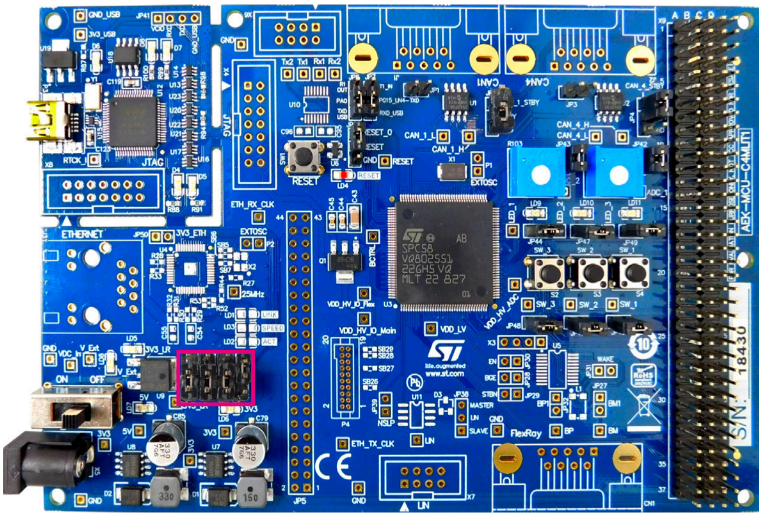
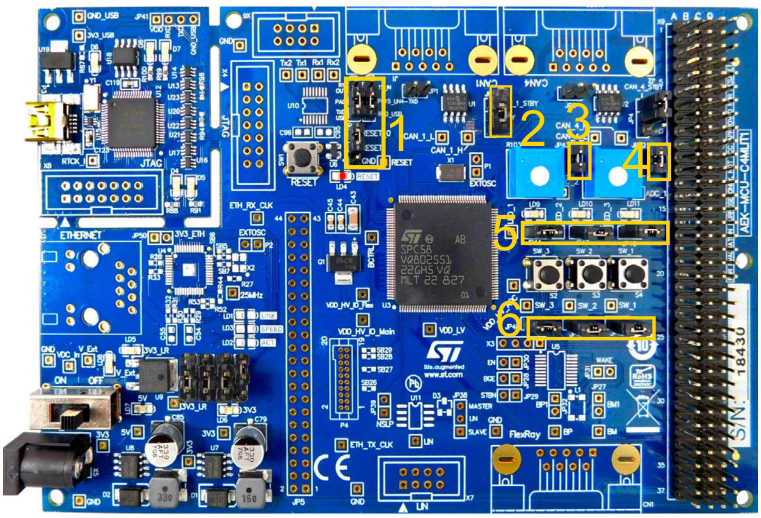


Figure 8. AEK-MCU-C4MLIT1 other jumpers

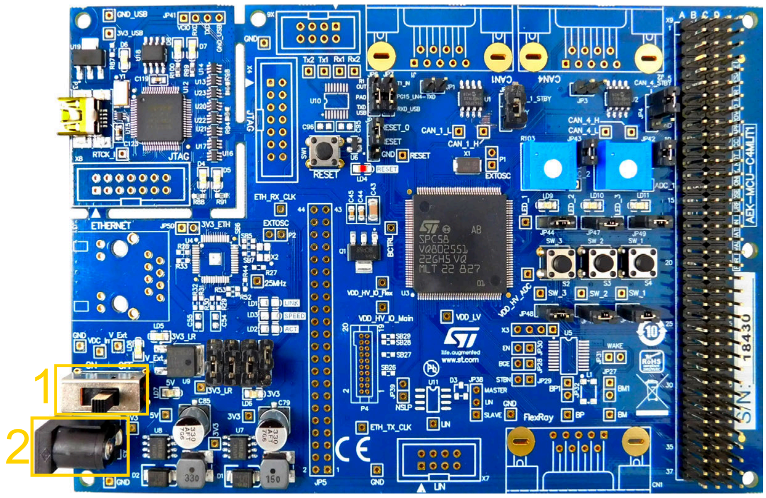
1. JP6, JP7, JP33
2. JP2
3. JP42
4. JP43
5. JP44, JP47, JP49
6. JP45, JP46, JP48



Step 6. Connect the plug of the DC supply (12 V, 2.0 A) and turn on the board by switching S1 to ON state.

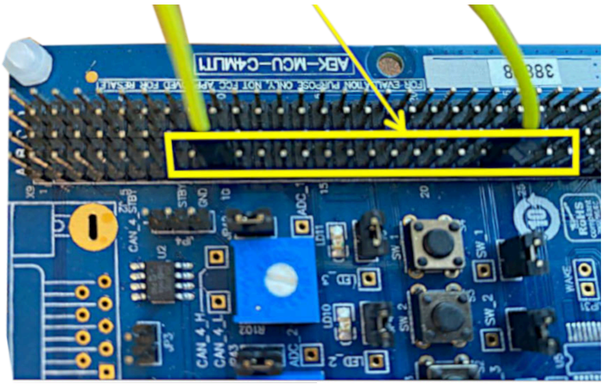
Figure 9. **AEK-MCU-C4MLIT1 plug connector for DC supply and power switch**

1. Power switch
2. Plug connector for DC supply



Step 7. Connect pin A10 with pin A23 through wires.
A10 and A23 pins belong to the 4x37 pins "X9" connector.

Figure 10. **AEK-MCU-C4MLIT1 physical connection between pin A10 and pin A23**

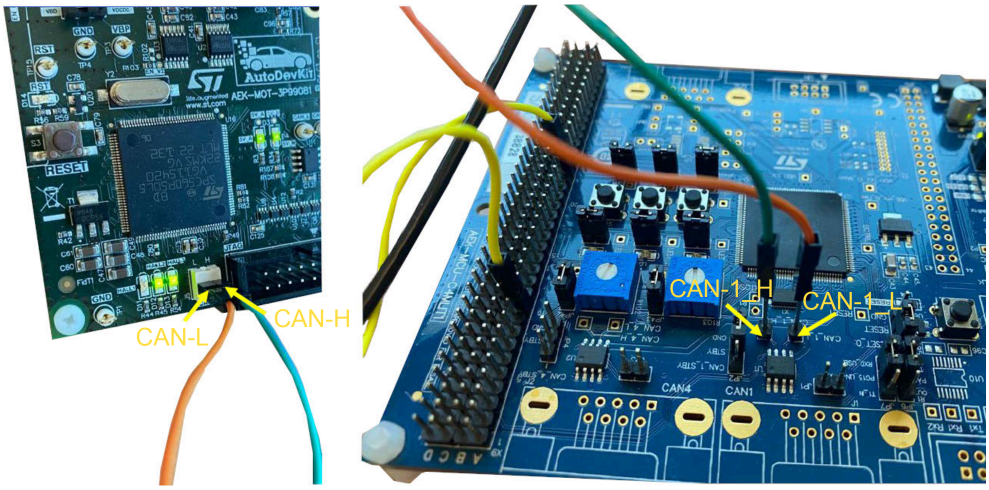


Step 8. Connect CAN_1_L of the AEK-MCU-C4MLIT1 system master board to the CAN_L of the AEK-MOT-3P99081 system slave board.

Step 9. Connect CAN_1_H of the AEK-MCU-C4MLIT1 to the CAN_H of the AEK-MOT-3P99081

To ease CAN wire connection between the two boards, two male connector-pins have been welded on the CAN_1_H and CAN_1_L outputs of the AEK-MCU-C4MLIT1 board.

Figure 11. AEK-MCU-C4MLIT1 and AEK-MOT-3P99081 wire connections



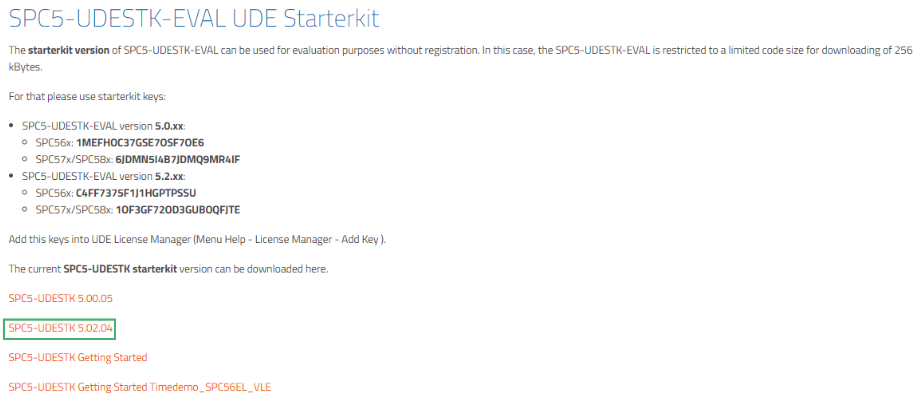
4 How to install UDE visual platform and PLS USB JTAG drivers

The following procedure describes how to download and install the latest version of SPC5-UDESTK. You can skip this procedure if the latest version is already installed on your PC.

Step 1. Go to the tool download page.

Step 2. Download SPC5-UDESTK 5.02.04 (or greater version available), save udestk-5-02-04-spc5-stk.exe and run it as an administrator.

Figure 12. SPC5-UDESTK starter-kit


SPC5-UDESTK-EVAL UDE Starterkit

The **starterkit version** of SPC5-UDESTK-EVAL can be used for evaluation purposes without registration. In this case, the SPC5-UDESTK-EVAL is restricted to a limited code size for downloading of 256 kBytes.

For that please use starterkit keys:

- SPC5-UDESTK-EVAL version **5.0.xx**:
  - SPC56x: **1MEFHOC37GSE7OSF7OE6**
  - SPC57x/SPC58x: **6JDMN5I4B7JDMQ9MR4IF**
- SPC5-UDESTK-EVAL version **5.2.xx**:
  - SPC56x: **C4FF7375F1J1HGPTPSSU**
  - SPC57x/SPC58x: **1OF3GF72OD3GUBOQFJTE**

Add this keys into UDE License Manager (Menu Help - License Manager - Add Key).

The current **SPC5-UDESTK starterkit** version can be downloaded here.

SPC5-UDESTK 5.00.05

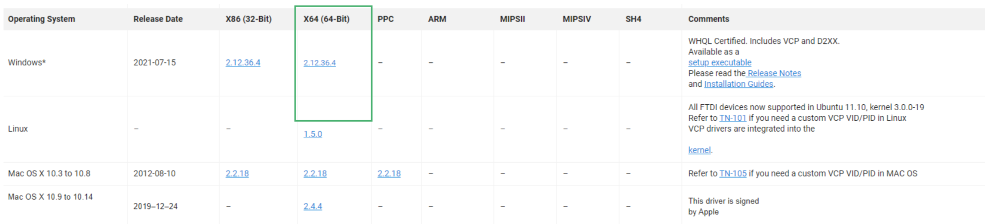
SPC5-UDESTK 5.02.04

SPC5-UDESTK Getting Started

SPC5-UDESTK Getting Started Timedemo_SPC56EL_VLE

Step 3. Install PLS USB JTAG Driver Package (v. 2.12.36.4) and PLS USB JTAG Driver Package-VCP driver (v. 2.12.36.4) from FTDIC Chip.

Figure 13. VCP driver library

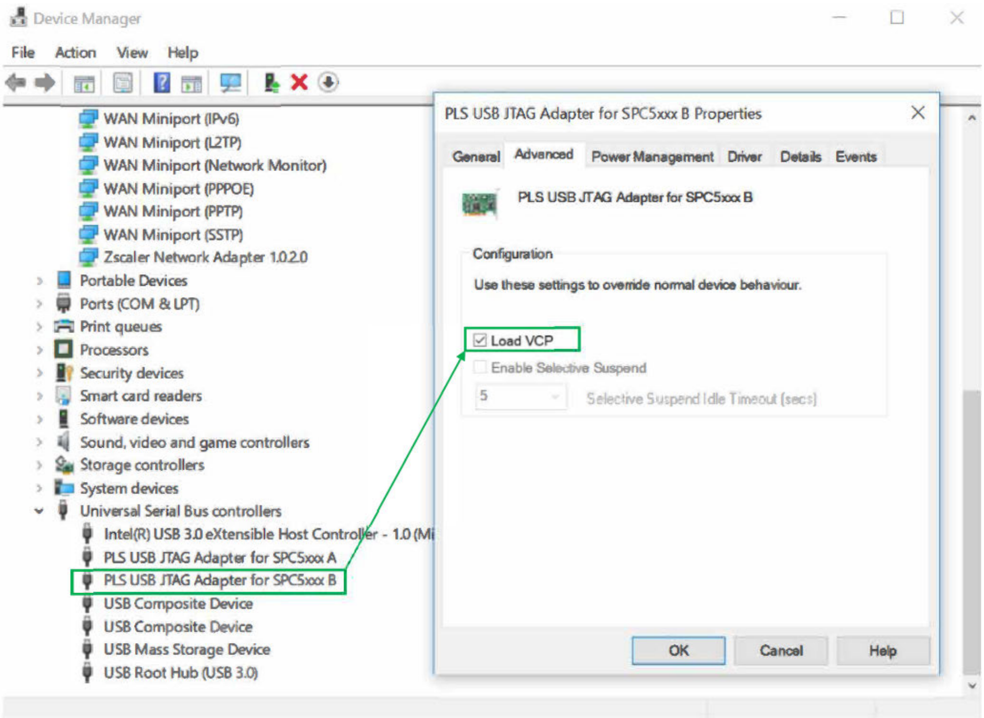


Operating System	Release Date	X86 (32-Bit)	X64 (64-Bit)	PPC	ARM	MIPSII	MIPSIV	SH4	Comments
Windows*	2021-07-15	2.12.36.4	2.12.36.4	–	–	–	–	–	WHQL Certified. Includes VCP and D2XX. Available as a setup executable Please read the Release Notes and Installation Guides.
Linux	–	–	1.5.0	–	–	–	–	–	All FTDI devices now supported in Ubuntu 11.10, kernel 3.0.0-19 Refer to TN-101 if you need a custom VCP VID/PID in Linux VCP drivers are integrated into the kernel.
Mac OS X 10.3 to 10.8	2012-08-10	2.2.18	2.2.18	2.2.18	–	–	–	–	Refer to TN-105 if you need a custom VCP VID/PID in MAC OS
Mac OS X 10.9 to 10.14	2019–12–24	–	2.4.4	–	–	–	–	–	This driver is signed by Apple

Step 4. Once the installation is completed, connect the USB cable between the PC and the USB port on the microcontroller board. From [**Start**] menu, right click on [**Computer**], then select [**Manage**]. Once the Computer management pop-up appears, from [**System Tools**] menu, select [**Device Manager**].

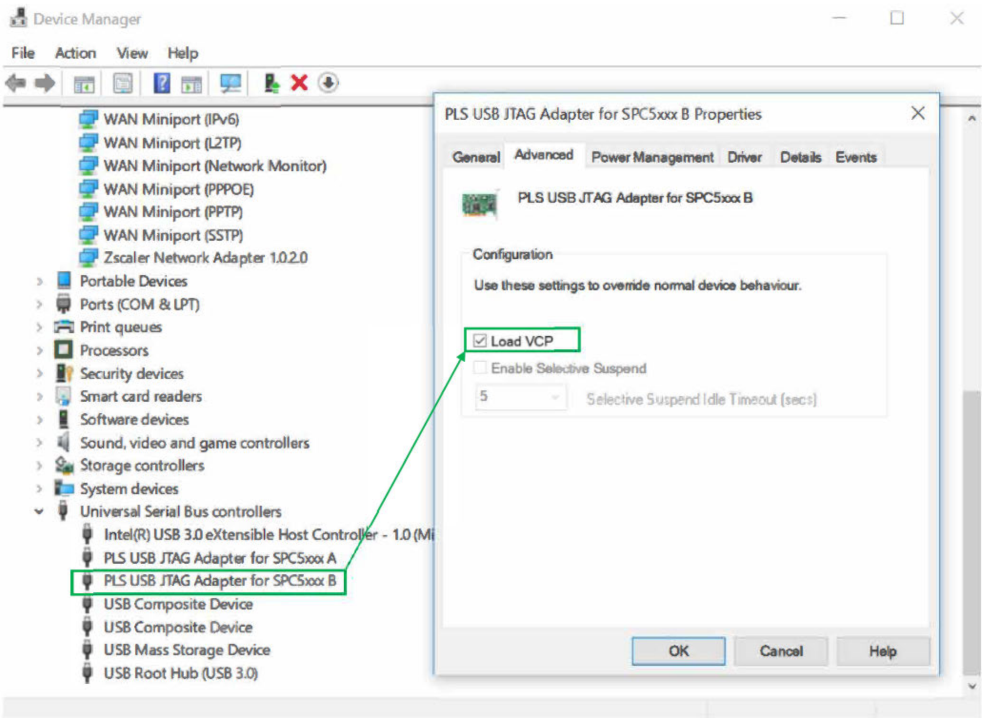
Step 5. Expand the Universal Serial Bus controllers and click on [**PLS USB JTAG Adapter for SPC5xxx B**]. Then, click on Advanced tab and check that Load VCP is flagged.

Figure 14. **PLS JTAG Adapter for SPC5xxx B**



Step 6. Go back to Universal Serial Bus controllers and click on PLS USB JTAG Adapter for SPC5xxx A. Click on Advanced tab and check that Load VCP is NOT flagged.

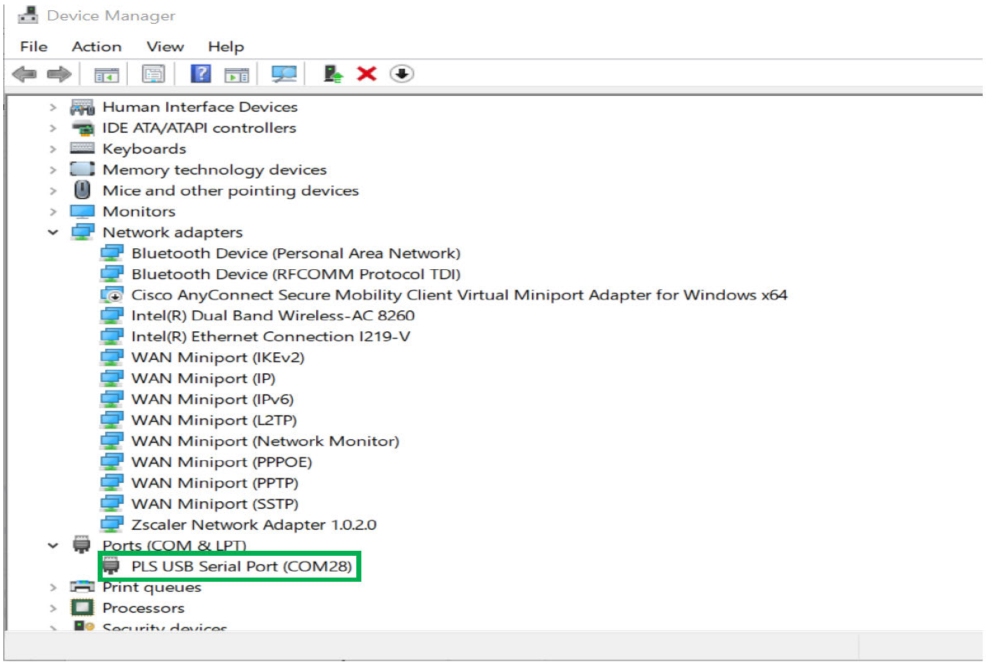
Figure 15. **PLS JTAG Adapter for SPC5xxx A**



Step 7. Disconnect the USB cable from the PC, then reconnect it again.

A new COM port appears in the [**Device Manager**] window. You should see it as a new COM port available in [**Ports (COM & LPT)**]. The COM port is configured and available to be used for serial communication with the PC.

Figure 16. New COM port available



5 How to flash the microcontroller firmware using UDE visual platform

5.1 How to flash the microcontroller firmware on the AEK-MOT-3P99081

The demo works with a 24 V supply, using a Nanotec brushless DC servo motor (model: DF45L024048-A2), and with a three-shunt resistor (ground-referred) configuration. Hall-effect sensors perform rotor position sensing.

Important:
If you want to change the supply voltage from 24 V to 12 V or 48 V or other parameters in the drive, you have to configure, generate and recompile a new firmware version.

After installing SPC5Studio (version 6.0.0 Clarke or greater version), importing, generating, and compiling the *SPC560Pxx_RLA_AEK_MOT_3P99081_3Phase_Motor_Control_L9908_via_CAN* demo, follow the procedure below to flash the firmware on the SPC560P50L5 microcontroller embedded in the AEK-MOT-3P99081 board.

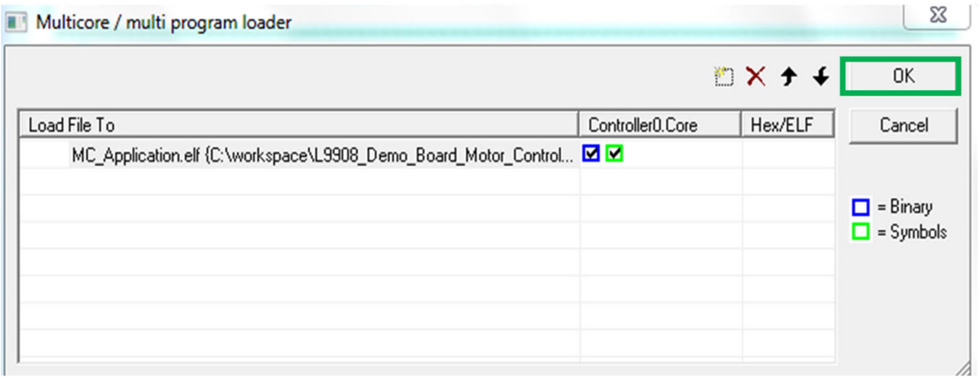
Step 1. Plug the USB cable to the PC and turn on the power supply connected to the demo board.

Step 2. Open *UDE-STK-5.2* and click on [**File**]>[**Open Workspace**]>[**SPC560Pxx_RLA_AEK_MOT_3P99081_3Phase_Motor_Control_L9908_via_CAN**]>[**UDE**]>[**debug.wsx**].

Step 3. In the left corner, click on [**File**]>[**Load Program**].

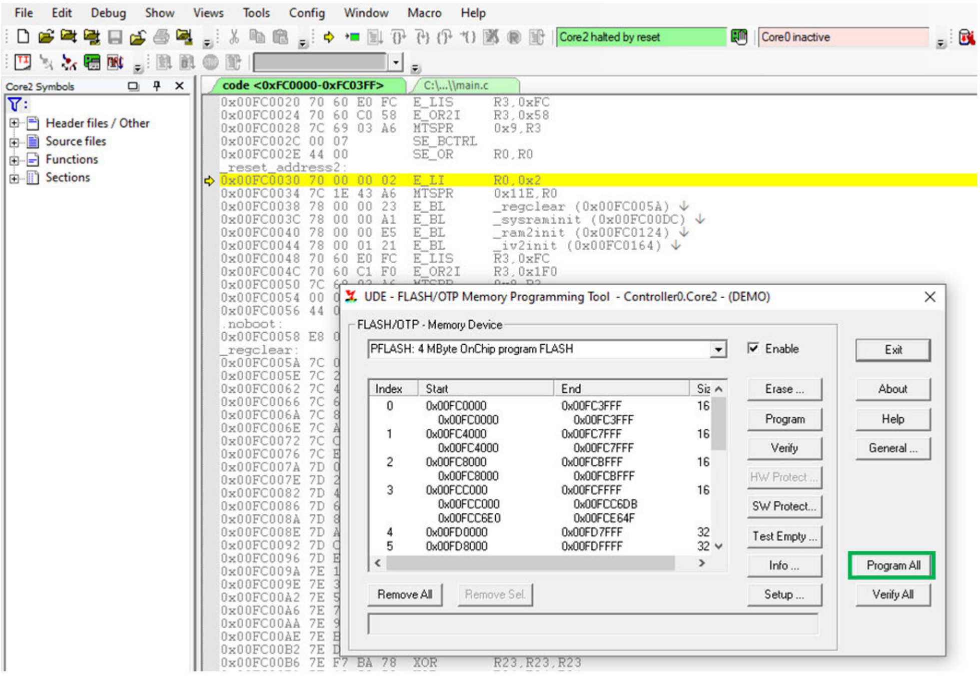
Step 4. Click [**OK**] to load *MC_Application.elf*, which is the binary file to be loaded on the microcontroller and is located in the build folder of the demo project: C:\Workspace\SPC560Pxx_RLA_AEK_MOT_3P99081_3Phase_Motor_Control_L9908_via_CAN\build .

Figure 17. Loading the binary file onto the microcontroller



Step 5. Click on [**Program All**] and wait for the program loading to complete (i.e., the loading bar reaches 100%). At end click on [**Exit**].

Figure 18. Flashing the firmware using UDE-STK



Step 6. Press the [**Start**] button in the toolbar.

Note: The motor starts running as soon as you push the AEK-MOT-3P99081 reset button.

Step 7. Close UDE STK 5.2.

5.2 How to flash the microcontroller firmware on the AEK-MCU-C4MLIT1

After installing SPC5Studio (version 6.0.0 Clarke or greater version), importing, generating, and compiling the *SPC58ECxx_RLA_MainEcuForBLDCControl-L9908-Test Application* demo, follow the procedure below to flash the firmware on the SPC58EC80E5 microcontroller embedded in the AEK-MCU-C4MLIT1 board.

Step 1. Plug the USB cable to the PC and turn on the power supply connected to the demo board.

Step 2. Open *UDE-STK-5.2* and click on [**File**]>[**Open Workspace**]>[**SPC58ECxx_RLA_MainEcuForBLDCControl-L9908-Test Application**]>[**UDE**]>[**debug.wsx**].

Step 3. In the left corner, click on [**File**]>[**Load Program**].

Step 4. Click [**OK**] to load *out.elf*, which is the binary file to be loaded on the microcontroller and is located in the build folder of the demo project: C:\Workspace\SPC58ECxx_RLA_MainEcuForBLDCControl-L9908-TestApplication\build.

Step 5. Press the [**Start**] button in the toolbar.

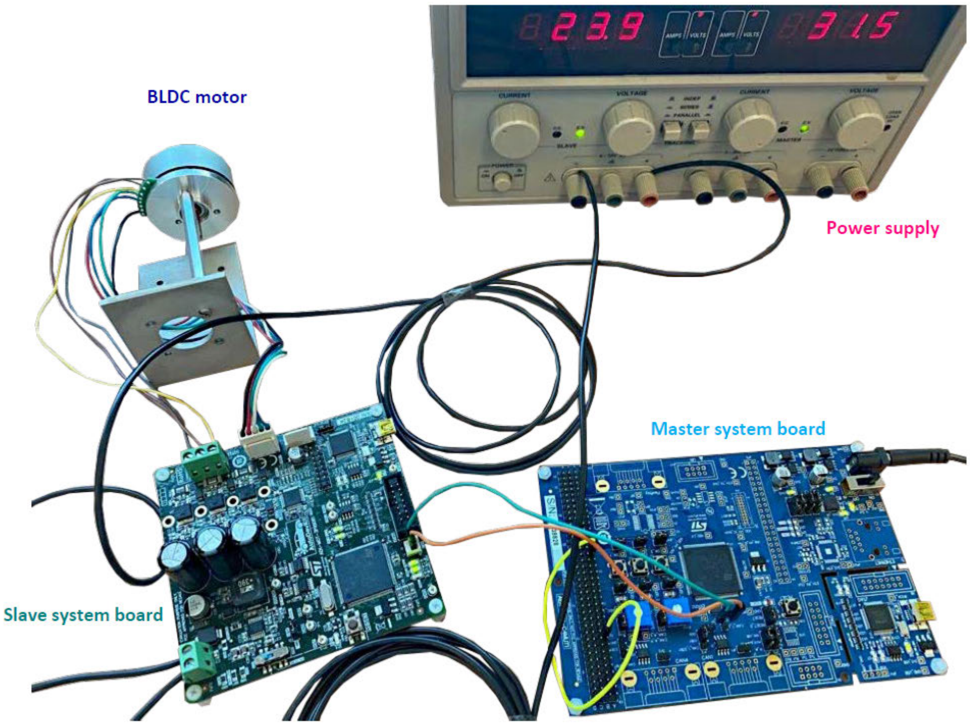
Step 6. Close UDE STK 5.2.

Note: For further details on how to import a demo in SPC5Studio, refer to UM2719, Section 7.2.1.

6 How to manage the system demo using CAN communication protocol between the AEK-MCU-C4MLIT1 and the AEK-MOT-3P99081

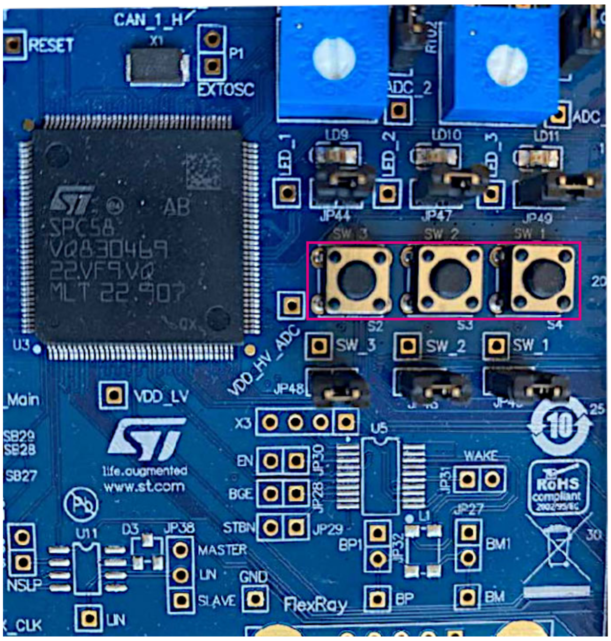
The figure below shows the main hardware components for a Hall-sensor three-phase BLDC motor control.

Figure 19. **BLDC drive system configuration**



Step 1. After setting up AEK-MOT-3P99081 and AEK-MCU-C4MLIT1, supply the boards.
The AEK-MCU-C4MLIT1 SW1, SW2 and SW3 buttons will be enabled.

Figure 20. AEK-MCU-C4MLIT1 board SW1, SW2 and SW3 (highlighted below)



Step 2. Push the buttons above to send CAN commands to the BLDC motor drive:
- SW1 button increases the target speed of the BLDC motor control;
- SW2 button decreases the target speed of the BLDC motor control;
- SW3 button turns the BLDC motor control algorithm ON/OFF;
- SW2 button pressed several times decreases the target speed of the BLDC motor control to reverse the rotation direction of the BLDC motor.

Note: As soon as the supply voltage is provided to both boards, the BLDC motor starts spinning and is controlled at a fixed target speed. If upon voltage supply, the previously described start operative condition does not occur, reset the AEK-MOT-3P99081 board MCU.

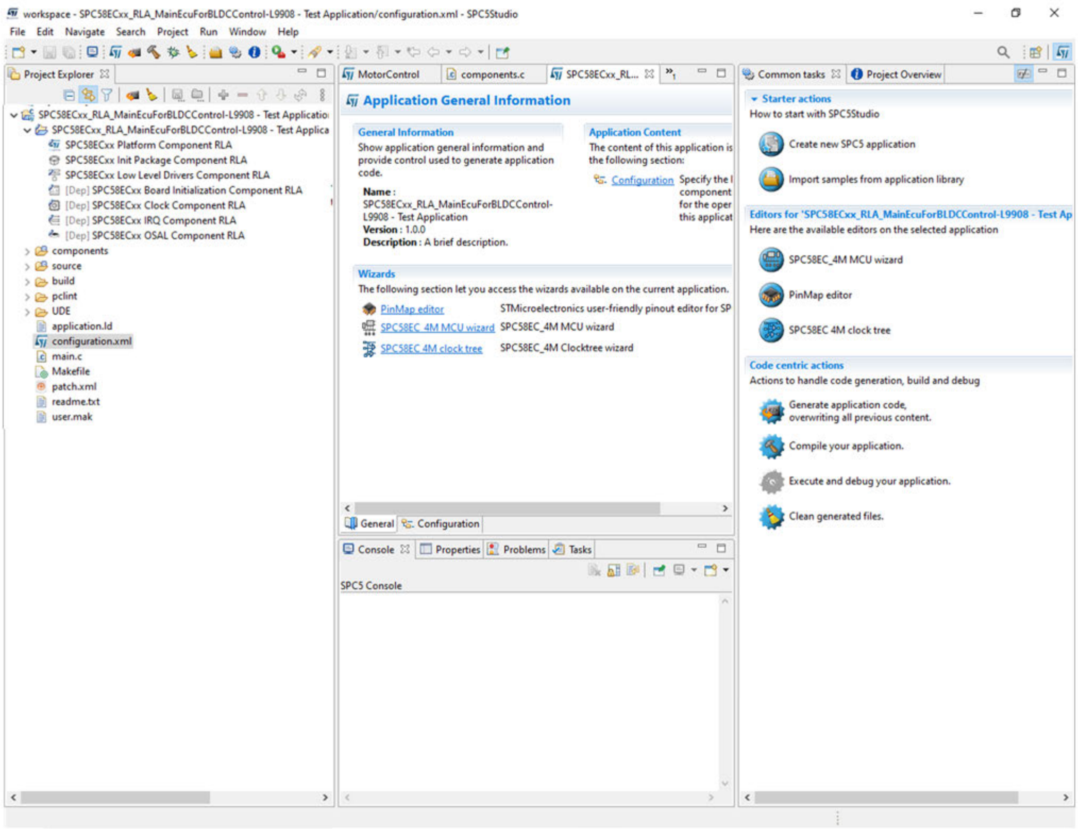
7 SPC5-STUDIO overview

SPC5-STUDIO is an integrated development environment (IDE) based on Eclipse. It consists of a standard workspace and an extensible plug-in system environment customization.

This IDE helps to maximize embedded application developer productivity with SPC5 Power Architecture 32-bit microcontrollers employing a single tool for evaluation, development, design, and production.

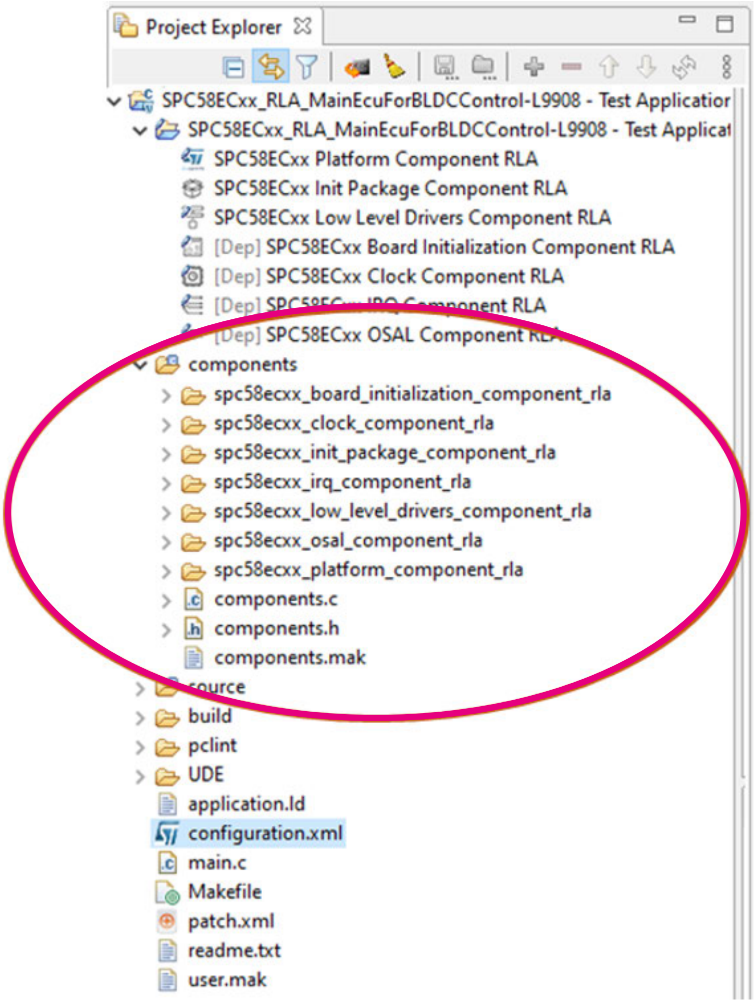
SPC5-STUDIO includes an application wizard to simplify project creation and configuration; it automatically solves component dependencies and generates support files.

Figure 21. **SPC5-STUDIO - SPC58ECxx_RLA_MainEcuForBLDCControl-L9908-Test Application project workspace**



The application wizard integrates the initial components into the project, together with the key elements employed by SPC5-STUDIO to generate the final application source code. The services of one component are provided to other components. Moreover, each component configuration is supported by an intuitive GUI.

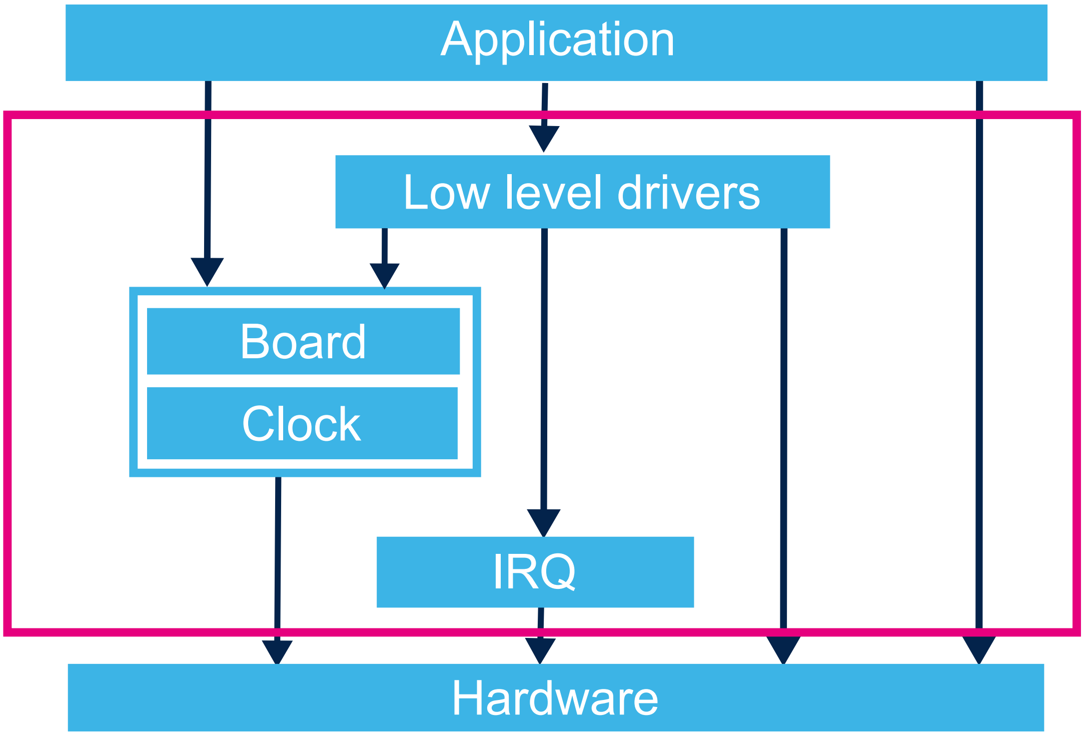
Figure 22. **SPC5-STUDIO - SPC58ECxx_RLA_MainEcuForBLDCControl-L9908-Test Application project components**



Register Level Access (RLA) components are low-level drivers with direct access to the MCU and peripherals such as CAN, Ethernet, DSPI, ADC, programmable interrupt timer (PIT) and advanced timing and synchronization unit (GTM). The RLA components can be added and configured via GUIs.

Note: *The RLA source code is MISRA 2021 checked.*

Figure 23. **SPC5-STUDIO - software architecture**



The FreeRTOS open source real-time operating system is available on request as a separate component compatible with the rest of the environment.

SPC5-STUDIO also contains straightforward software examples for each MCU peripheral that can help developers to become familiar with the specific code involved.

SPC5-STUDIO also features:

- the possibility of integrating other software products from the Eclipse standard marketplace
- availability of a free license GCC GNU C Compiler component
- support for industry-standard compilers
- support for multi-core microcontrollers
- a PinMap graphical editor to facilitate MCU pin configuration

7.1 How to create a new project in SPC5-STUDIO

Step 1. Install SPC5-STUDIO version 6.0.0 Clarke or greater version.

Step 2. Create a new SPC5 application.

– Select [**File**]>[**New**]>[**SPC5 C/C++ Application**] or

– Select the [**Create a new SPC5 application**] icon in the [**Starter actions**] tab

A window for application details pops up.

Step 3. Fill in the application details and then click on the [**Next**] button.

Step 4. Select the SPC58ECxx component and select the [**Finish**] button.

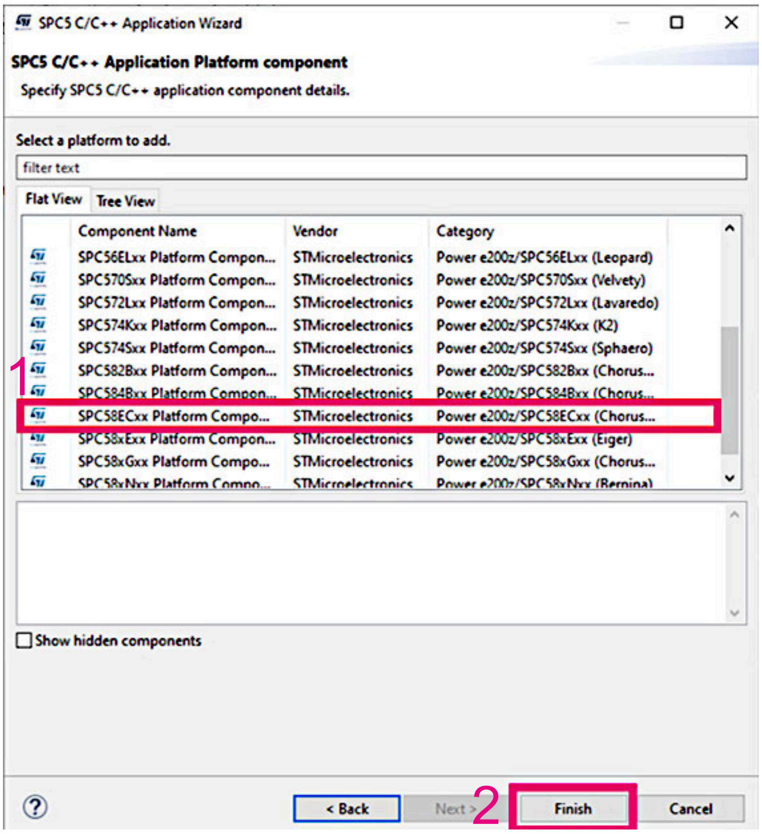
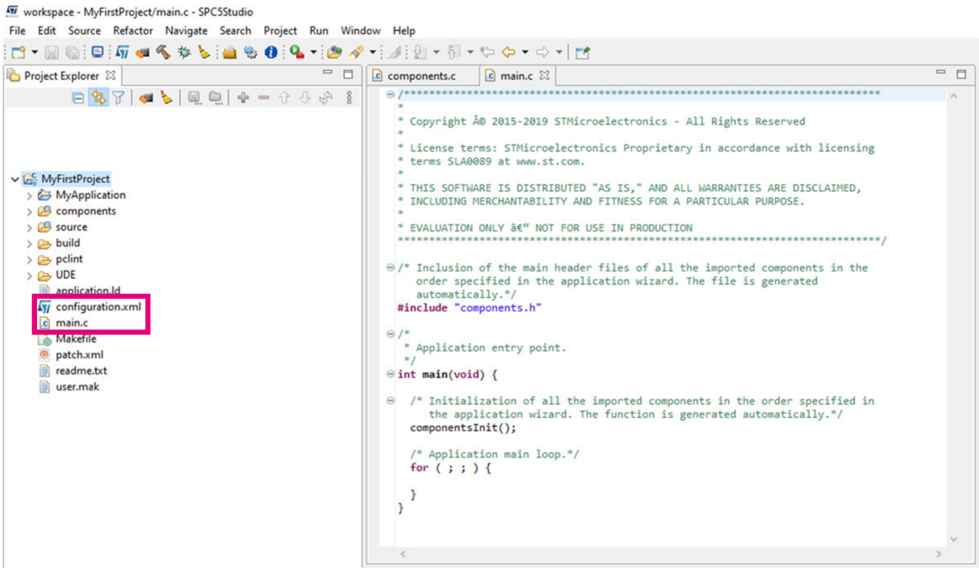


Figure 24. **SPC5-STUDIO new application MCU platform**

Step 5. Click on the [**Generate application code**] button to create a base C project in the workspace.
The following files are visible in the application tree view:
- *configuration.xml* containing the project configuration information updated each time the project is changed;
- *main.c* where the actual application is implemented. Upon creation, the *main.c* file only contains a basic initialization section and an infinite loop.

Figure 25. SPC5-STUDIO new application files in outline view and main.c file automatically generated



7.2 How to add components to an SPC5-STUDIO project

Step 1. In the [**Project Explorer**] tab, select the SPC58Cxx Platform component RLA.

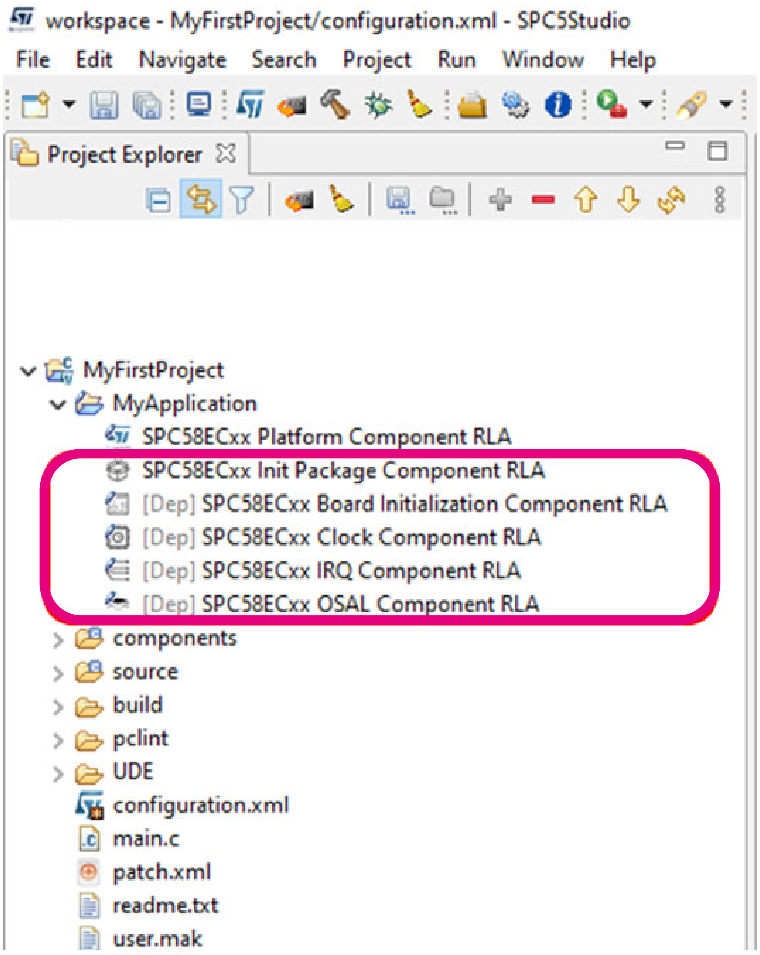
Step 2. Open the available components window for the chosen platform:
- right-click and select add, or
- select the + icon in the project explorer

Step 3. Select the components to add and click [**OK**].

You should add at least the following components to your SPC5-STUDIO project:

- SPC58Cxx Init Package Component;
- Low Level Driver;
- SPC58Cxx Board Initialization Component RLA: to initialize and configure the selected board;
- SPC58Cxx Clock Component RLA: to configure the MCU clock tree;
- SPC58Cxx IRQ Component RLA: to set and configure interrupt the request QUEUE;
- SPC58Cxx OSAL Component RLA: operating system abstraction.

Figure 26. SPC5-STUDIO new components visible in the project tree



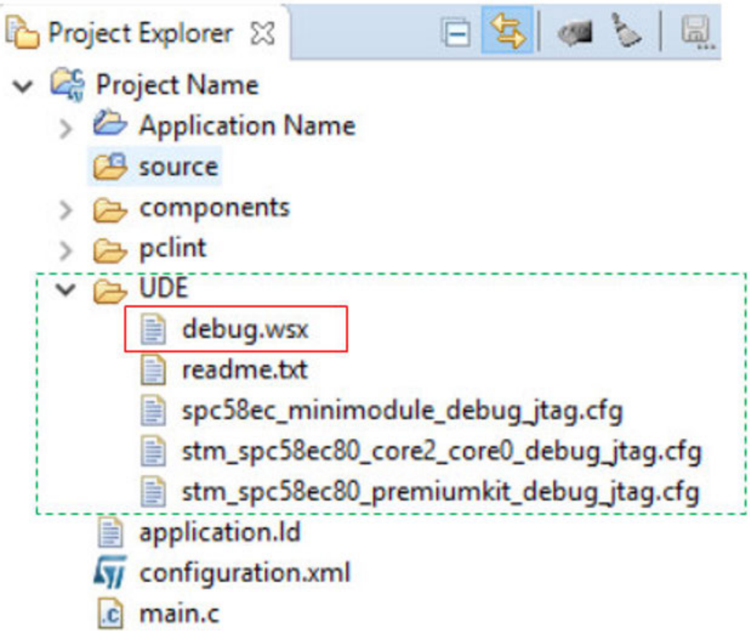
7.3 How to generate and compile application source code

Step 1. Save the project.

Step 2. Click on the [**Generate**] icon to update the configuration files.

Step 3. Click on the [**Compile**] icon to compile the project source and produce the MCU Flash image.
Once the source code is compiled and linked, the *debug.wsx* file appears in the UDE folder.

Figure 27. **SPC5-STUDIO compiled project files**



7.4 AutoDevKit plug-in

The AutoDevKit plug-in extends SPC5-STUDIO into a straightforward, low-cost, time saving tool designed to help automotive application engineers evaluate, prototype, develop and deploy complex embedded systems.

AutoDevKit features:

- fast prototyping: with integrated hardware and software components, component compatibility checking, and MCU and peripheral configuration tools
- flexibility: allowing the creation of new system solutions from existing ones by adding dedicated boards for further functionalities or removing unused boards
- hardware abstraction: if, for example, the user swaps the SPC58EC Chorus 4M MCU with an SPC584B Chorus 2M, or an SPC582B Chorus 1M, the tool will automatically regenerate the proper pin allocation
- high-level application APIs: to control specific functional boards and exploit the chosen hardware functionality to facilitate communication with the MCU

AutoDevKit provides a GUI to facilitate the configuration and setup for each supported component, as well as sample code demonstrating typical usage and applications.

Another key feature is the BoardView graphical tool showing the electrical wiring between the microcontroller and board connectors and the connection between the MCU board and other functional boards.

AutoDevKit adds new items to the list of the components that can be used in an SPC5-STUDIO project. Each functional board component appears in the list of the MCUs able to support it only. Refer to RN0118 for all the components installable with the AutoDevKit plug-in.

Before using AutoDevKit, download the latest release of AutoDevKit plug-in (STSW-AUTODEVKIT).

For more details about the AutoDevKit plug-in installation, see available videos.

8 How to set up BLDC motor parameters

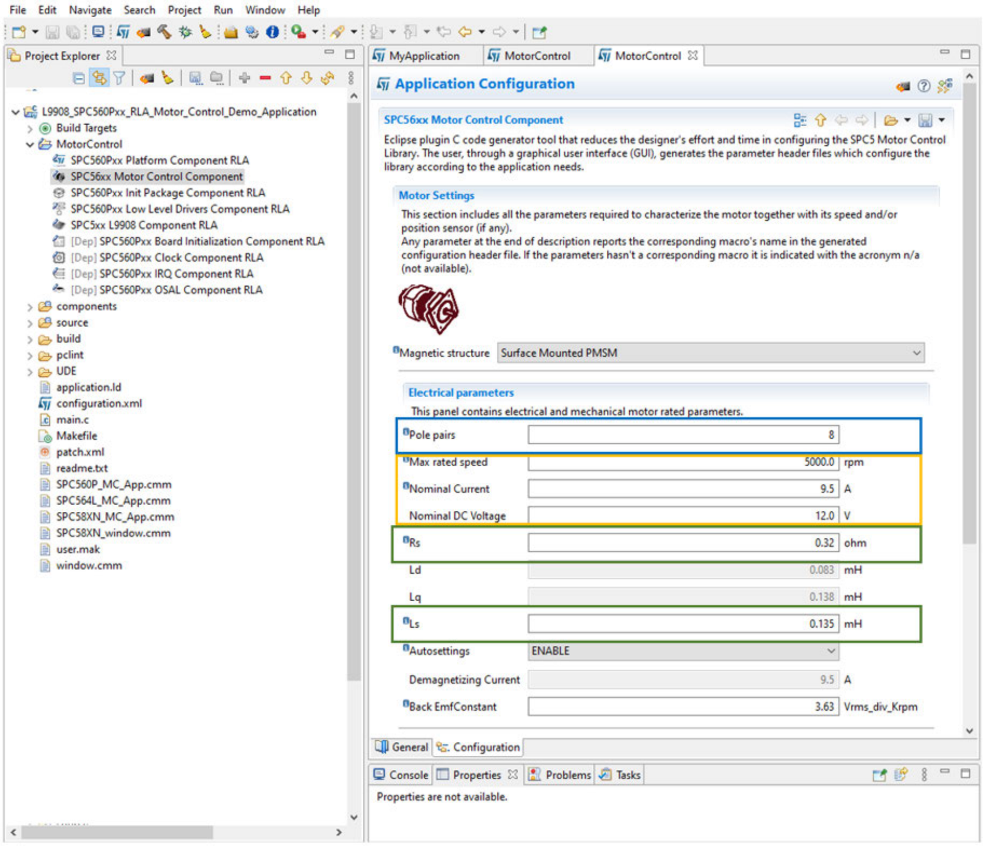
The AEK-MOT-3P99081 drive kit is built with a Nanotec brushless DC servo motor (model: DF45L024048-A2).

You can use different Hall-effect sensor three-phase BLDC motors by customizing the motor drive settings with the actual values of the following parameters:

- nominal parameters (max. rated speed, nominal current, nominal DC voltage);
- number of pole pairs;
- electrical parameters (stator resistance and inductance).

These parameters can be used to update the BLDC motor parameters in the SP56xx-Motor-Control-Component of theSPC560Pxx_RLA_AEK_MOT_3P99081_3Phase_Motor_Control_L9908_via_CAN project (the firmware to be flashed into the SPC560P MCU of AEK-MOT-3P99081 board).

Figure 28. SP56xx-Motor-Control-Component parameters to be updated to use a different Hall-effect sensor three-phase BLDC motor



Step 1. Save the *SPC560Pxx_RLA_AEK_MOT_3P99081_3Phase_Motor_Control_L9908_via_CAN* project.

Step 2. Click on [**Generate**] to update the configuration files related to the project, and then click on [**Compile**] to get the MCU flash image.

Step 3. Download the new firmware on the SPC560P MCU of the AEK-MOT-3P99081 board.

For the new BLDC motor, the nominal parameters are:

– *Max Rated Speed* that is the maximum motor speed according to the application level requirements;

– *Nominal Current* that is the peak current provided to each of the motor phases according to the BLDC motor specifications;

– *Nominal DC Voltage* that is the value of the DC bus to be provided to the inverter.

8.1 How to determine the number of pole pairs of a BLDC motor

The number of pole pairs of a BLDC motor is usually provided by the motor supplier. If this parameter is not available, considering that the number of pole pairs corresponds to rotor stable positions in one mechanical turn, you can determine them by following the procedure below.

Step 1. Connect a DC power supply between two (of the three) motor phases and provide up to 5% of the expected nominal DC bus voltage.

Step 2. Gradually rotate the motor by hand up to a full mechanical turn of the rotor.

During the mechanical turn of the rotor from a “jamming” rotor position to the next, you should notice a little resistance, otherwise:

- if you are not able to rotate the motor, decrease the applied voltage;
- if the motor does not generate any resistance, increase the applied voltage.

The number of “jamming” positions of the rotor detected during the procedure corresponds to the number of pole pairs.

8.2 How to calculate stator resistance and inductance

The stator resistance and inductance of the BLDC motor are parameters usually provided by the BLDC motor technical documents. If these parameters are not available, you can determine them by following the steps below.

Step 1. Using a multimeter, measure the DC stator resistance phase-to-phase (Rs) and divide it by two.

Step 2. Connect the DC voltage between two motor phases.

Step 3. Connect the oscilloscope probes to monitor voltage and current.

Step 4. Increase the voltage up to where the current equals the nominal value, so the rotor will align with the generated flux.

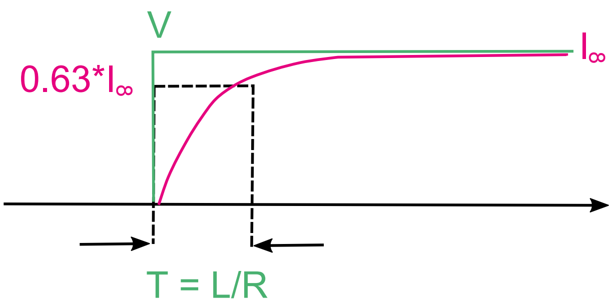
Step 5. Do not move the rotor anymore.

Step 6. Disable the current protection of DC voltage source.

Step 7. Unplug one terminal of the voltage source cable without switching it off.

Step 8. Plug the voltage source rapidly and monitor the voltage and current waveform on the oscilloscope.

Figure 29. Nominal shape of voltage and current during the procedure to determine the stator resistance and inductance of the BLDC motor



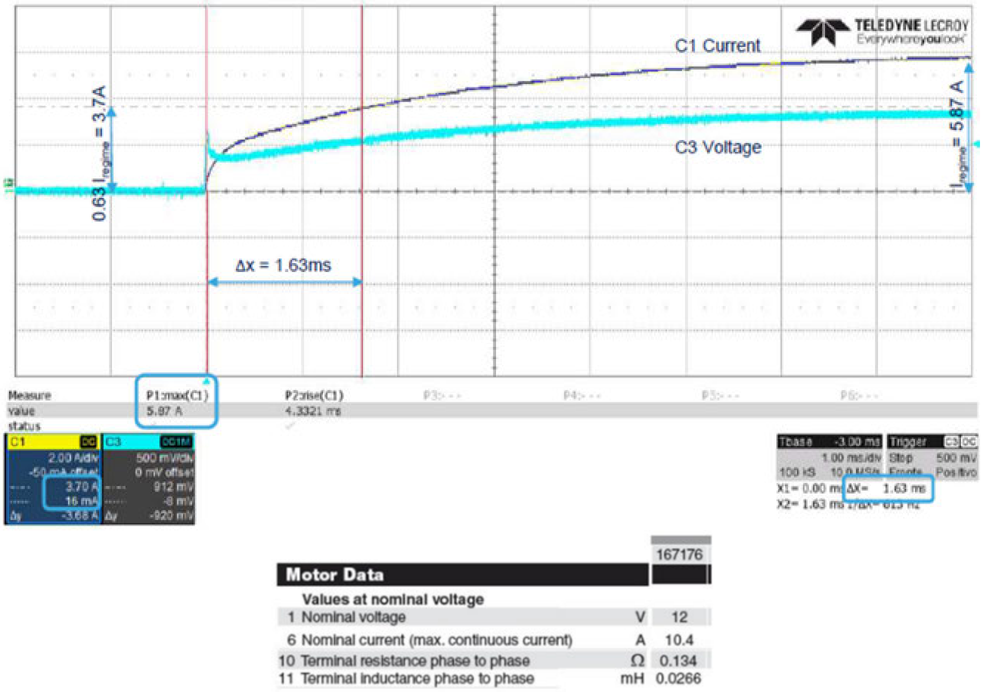
The measurement is useful if the voltage can be assimilated to a step and the current increase as $I_{\infty} = \left(1 - e^{-t * L/R]}\right)$

Step 9. Measure the time needed by the current waveform to rise up to 63% of the plateau value.

This value corresponds to the ratio Ls/Rs. Multiplying it by Rs, it is possible to get the Ls value.

The following figure shows the results of the above procedure when applied to the MAXON EC40 BLDC motor.

Figure 30. Results of the procedure to determine the stator resistance and inductance of the BLDC motor, when applied to the BLDC motor MAXON EC40



- Δx is the time needed by the current to reach 63% of the plateau value = T
- T = L/R≥ L = T*R where R = 0.134 Ω and L = 0.00163*0.134 = 0.218 mH

Note: The nominal current of the motor is 10.4 A and 6 A is the limit value determined by the power supply.

Warning:
*To avoid issues when restarting the motor, it is important to set the "Lowside PWM idle state" parameter to "Turn_on" in the [**Driver Settings**] section, as shown in the image below.*

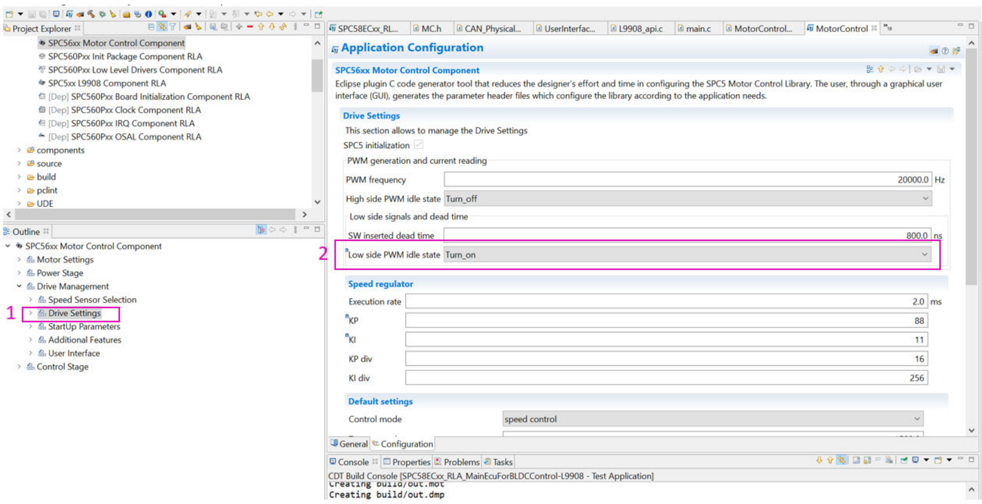


Figure 31. Drive Settings section

9 Motor control protocol

9.1 Overview

The distributed system solution proposed to control BLDC motors in this demonstration kit is a dedicated CAN communication protocol allowing real-time data exchange. This protocol allows sending commands to start/stop the BLDC motor, to increase/decrease the motor target speed, and to reverse the rotation direction of the motor.

Market applications which require to drive an electrical motor usually embeds a domain control ECU and a motor drive ECU. To drive the system correctly, the domain controller requires a method to send a command to the motor drive board and get a feedback. To meet these requirements, we have implemented a Motor Control Protocol (MCP).

The MCP enables commands to start/stop motor, to set the target speed of the FOC motor control firmware, and to tune control variables, such as PI coefficients, in real-time. The MCP also allows monitoring the speed of the motor or the bus voltage.

The master starts the communication any time by sending the first communication frame to the slave which answers with a frame for acknowledgement.

The CAN message defined in the MCP consists in:

- a **SID**: to identify the command requested to the motor drive
- a **DATA FRAME**: a set of bytes split in:
  - **PAYLOAD_LENGTH**: the total number of bytes that compose the frame payload
  - **PAYLOAD_ID**: first byte of the payload that contains the payload identifier (optional - depends on the frame type)
  - **PAYLOAD[X]**: the remaining payload content (optional - depends on the frame type)
  - **CRC**: for cyclic redundancy parity check.

Note: The smallest DATA frame is made of PAYLOAD_LENGTH = 0 and CRC.

Note: The CRC byte is computed by:

$$CRC = (unsigned8bit)(HighByte(Total) + LowByte(Total))$$

$$Total = (unsigned16bit)\left(SID + PAYLOAD\_LENGTH + PAYLOAD\_ID + \Sigma_{i=0}^{n} PAYLOAD[i]\right)$$

Figure 32. Generic data frame

Data Frame =

PAYLOAD_LENGHT	PAYLOAD_ID	PAYLOAD[0]	....	PAYLOAD[n]	CRC

The current CAN messages available in the MCP are listed in the following table.

Table 1. Starting frame codes managed by MCP

SID	Description
0x01	Set register frame: to write a value into a relevant motor control variable.
0x02	Get register frame: to read a value from a relevant motor control variable.
0x03	Execute command frame: to send a command to the motor control object.
0x06	Get board info: to retrieve information about the firmware currently running on the microcontroller. Payload length is 0.
0x07	Exec ramp: to execute a speed ramp.
0x0A	Set current references: to set the current reference.

9.2 Set register frame

The set register frame is sent by the master to write a value into a relevant motor control variable. The payload length depends on REG_ID (see Table 3 for more details) which indicates the register to be updated. The remaining payload contains the value to be updated starting from the least significant byte to the most significant byte.

The acknowledgment frame can be of two types:

- data acknowledgment frame - if the operation has been successfully completed. The payload is zero.
- error acknowledgment frame - if the operation has not been successfully completed by the firmware. The payload is always 1.

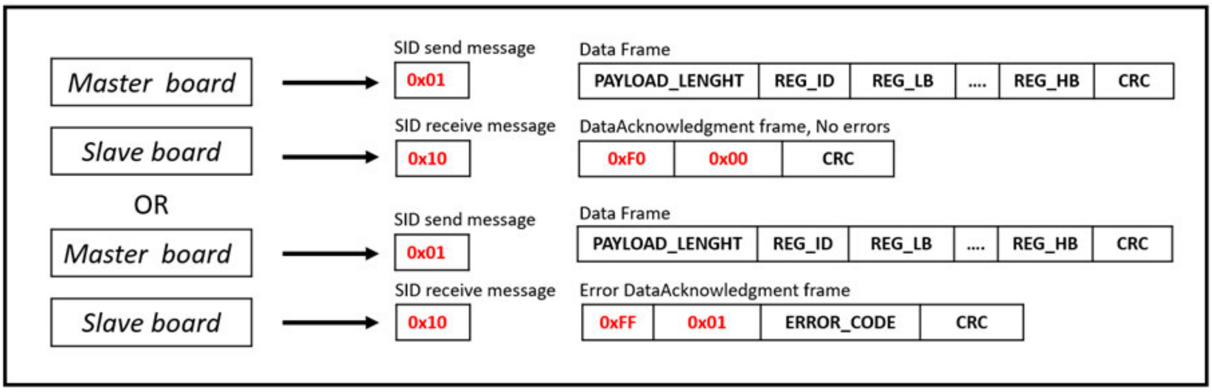


Figure 33. Set register frame

Table 2. List of error codes managed by MCP

Error name	Error code	Description
ERROR_BAD_FRAME_ID	0x01	Bad Frame ID- the Frame ID has not been recognized by the firmware.
ERROR_CODE_SET_READ_ONLY	0x02	Write on read-only - the master wants to write on a read-only register.
ERROR_CODE_GET_WRITE_ONLY	0x03	Read not allowed - the value cannot be read.
ERROR_CODE_WRONG_SET	0x05	Out of range - the value used in the frame is outside the range expected by the firmware.
ERROR_CODE_WRONG_CMD	0x07	Bad command ID - the command ID has not been recognized.
ERROR_CODE_OVERRUN	0x08	Overrun error - the frame has not been received correctly as the transmission speed is too fast.
ERROR_CODE_TIMEOUT	0x09	Timeout error - the frame has not been received correctly and a timeout occurred. This kind of error usually occurs when the frame is not correct or is not correctly recognized by the firmware.
ERROR_CODE_BAD_CRC	0x0A	Bad CRC - the computed CRC is not equal to the received CRC byte.

Table 3. List of relevant motor control registers managed by MCP

For each register, the table lists:

- Type: u8 means 8-bit unsigned, u16 means 16-bit unsigned, u32 means 32-bit unsigned, s16 means 16-bit signed, s32 means 32-bit signed)
- Payload length in set register frame
- Allowed access: R for read, W for write, RW for read and write
- REG_ID

Register name	Type	Payload length	Access	REG_ID
Target motor	u8	2	RW	0x00

Register name	Type	Payload length	Access	REG_ID
Flags	u32	5	R	0x01
Status	u8	2	R	0x02
Control mode	u8	2	RW	0x03
Speed reference	s32	5	R	0x04
Speed KP	u16	3	RW	0x05
Speed KI	u16	3	RW	0x06
Speed KD	u16	3	RW	0x07
Torque reference (Iq)	s16	3	RW	0x08
Torque KP	u16	3	RW	0x09
Torque KI	u16	3	RW	0x0A
Torque KD	u16	3	RW	0x0B
Flux reference (Id)	s16	3	RW	0x0C
Flux KP	u16	3	RW	0x0D
Flux KI	u16	3	RW	0x0E
Flux KD	u16	3	RW	0x0F
Observer C1	s16	3	RW	0x10
Observer C2	s16	3	RW	0x11
Cordic Observer C1	s16	3	RW	0x12
Cordic Observer C2	s16	3	RW	0x13
PLL KI	u16	3	RW	0x14
PLL KP	u16	3	RW	0x15
Flux weakening KP	u16	3	RW	0x16
Flux weakening KI	u16	3	RW	0x17
Flux weakening BUS Voltage allowed percentage reference	u16	3	RW	0x18
Bus Voltage	u16	3	R	0x19
Heat sink temperature	u16	3	R	0x1A
Motor power	u16	3	R	0x1B
DAC Out 1	u8	2	RW	0x1C
DAC Out 2	u8	2	RW	0x1D
Speed measured	s32	5	R	0x1E
Torque measured (Iq)	s16	3	R	0x1F
Flux measured (Id)	s16	3	R	0x20
Flux weakening BUS Voltage allowed percentage measured	u16	3	R	0x21
Revup stage numbers	u8	2	R	0x22
Stator Current Ia	s16	3	R	0x23
Stator Current Ib	s16	3	R	0x24
Stator Current Ialpha	s16	3	R	0x25
Stator Current Ibeta	s16	3	R	0x26

Register name	Type	Payload length	Access	REG_ID
Stator Current Iq	s16	3	R	0x27
Stator Current Id	s16	3	R	0x28
Stator Current Iqref	s16	3	R	0x29
Stator Current Idref	s16	3	R	0x2A
Stator Voltage Vq	s16	3	R	0x2B
Stator Voltage Vd	s16	3	R	0x2C
Stator Voltage Valpha	s16	3	R	0x2D
Stator Voltage Vbeta	s16	3	R	0x2E
Electrical Angle measured	s32	5	R	0x2F
Mechanical rotor speed	s32	5	R	0x30
Observed Electrical Angle	s32	5	R	0x31
Observed Mechanical rotor speed	s32	5	R	0x32
Observed Ialpha	s16	3	R	0x33
Observed Ibeta	s16	3	R	0x34
Observed Back Emf alpha	s16	3	R	0x35
Observed Back Emf beta	s16	3	R	0x36
Cordic Observed Electrical Angle	s32	5	R	0x37
Cordic Observed Mechanical rotor speed	s32	5	R	0x38
Cordic Observed Ialpha	s16	3	R	0x39
Cordic Observed Ibeta	s16	3	R	0x3A
Cordic Observed Back Emf alpha	s16	3	R	0x3B
Cordic Observed Back Emf beta	s16	3	R	0x3C
DAC User 1	s16	3	R	0x3D
DAC User 2	s16	3	R	0x3E
Maximum application speed	u32	5	R	0x3F
Minimum application speed	u32	5	R	0x40
Iq reference in speed mode	s16	3	W	0x41
Expected BEMF level (PLL)	s16	3	R	0x42
Observed BEMF level (PLL)	s16	3	R	0x43
Expected BEMF level (CORDIC)	s16	3	R	0x44

Register name	Type	Payload length	Access	REG_ID
Observed BEMF level (CORDIC)	s16	3	R	0x45
Feedforward (1Q)	s32	5	RW	0x46
Feedforward (1D)	s32	5	RW	0x47
Feedforward (2)	s32	5	RW	0x48
Feedforward (VQ)	s16	3	R	0x49
Feedforward (VD)	s16	3	R	0x4A
Feedforward (VQ PI out)	s16	3	R	0x4B
Ramp final speed	s32	5	RW	0x5B
Ramp duration	u16	3	RW	0x5C
External fault 1	u32	5	RW	0x66
External fault 2	u32	5	R	0x67
External fault 3	u32	5	R	0x68
External fault 4	u32	5	R	0x69
External fault 5	u32	5	R	0x6A
External fault 6	u32	5	R	0x6B
External fault 7	u32	5	R	0x6C
External fault 8	u32	5	R	0x6D
External fault 9	u32	5	R	0x6E
External fault 10	u32	5	R	0x6F
External fault 11	u32	5	R	0x70

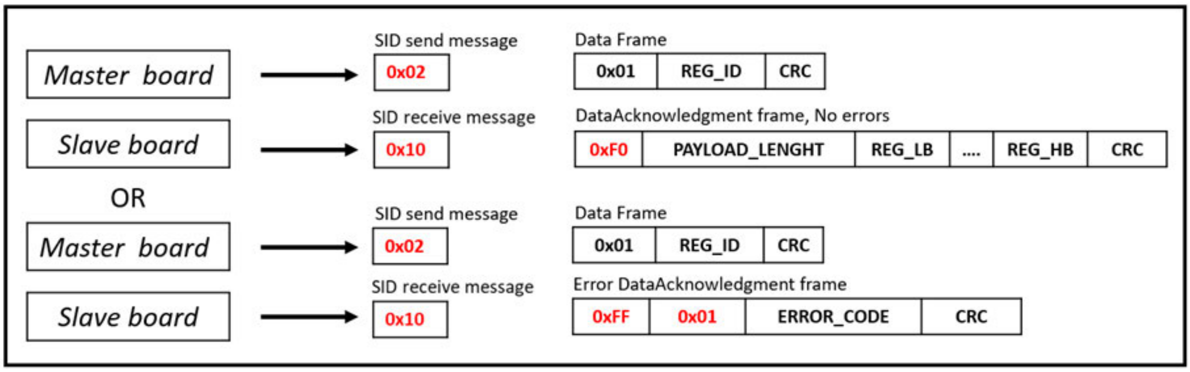
Table 4. List of external faults for L9908 managed by MCP

External fault	Reg returned from external device	Description
1	`GEN_STATUS1`	GEN_STATUS1 Register
2	`GEN_STATUS2`	GEN_STATUS1 Register
3	`GEN_STATUS3`	GEN_STATUS3 Register
4	`GEN_TEMP_STATUS`	GEN_TEMP_STATUS Register
5	`WDT_STATUS`	WDT_STATUS Register
6	`CH1_STATUS1`	CH1_STATUS1 Register
7	`CH1_STATUS2`	CH1_STATUS2 Register
8	`CH2_STATUS1`	CH2_STATUS1 Register
9	`CH2_STATUS2`	CH2_STATUS2 Register
10	`CH3_STATUS1`	CH3_STATUS1 Register
11	`CH3_STATUS2`	CH4_STATUS2 Register

9.3 Get register frame

The get register frame is sent by the master to read a value from a relevant motor control variable.

Figure 34. **Get register frame**



Payload length is always 1.

`REG_ID` indicates the register to be queried.

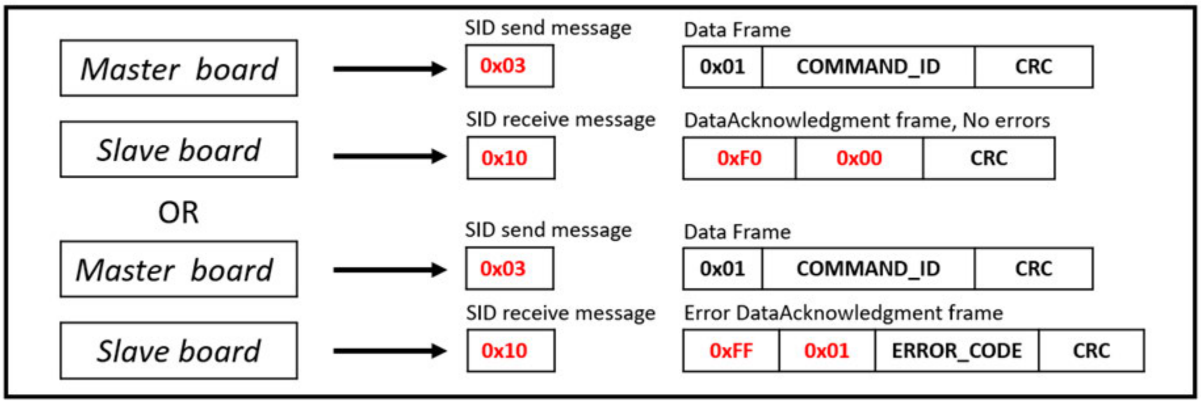
The acknowledgment frame can be of two types:

- data acknowledgment frame - if the operation has been successfully completed. In this case, the returned value is embedded in the data acknowledgment frame. The size of the payload depends on `REG_ID` and is equal to the payload length of Table 3 minus 1. The value is returned starting from the least significant byte to the most significant byte
- error acknowledgment frame - if the operation has not been successfully completed by the firmware. The payload is always 1. The list of error codes is shown in Table 2.

9.4 Execute command frame

The execute command frame is sent by the master to the motor control firmware to request the execution of a specific command.

Figure 35. **Execute command frame**



Payload length is always 1.

`COMMAND_ID` indicates the requested command.

Table 5. **List of commands**

Command	Command ID	Description
Start Motor	0x01	Indicates the user request to start the motor regardless of the state of the motor.
Stop Motor	0x02	Indicates the user request to stop the motor regardless of the state of the motor.

Command	Command ID	Description
Stop Ramp	0x03	Indicates the user request to stop the execution of the speed ramp that is currently executed.
Start/Stop	0x06	Indicates the user request to start the motor if the motor is still, or to stop the motor if it is running.
Fault Ack	0x07	Communicates the user acknowledgement of the occurred fault conditions.
Encoder Align	0x08	Indicates the user request to perform the encoder alignment procedure.

The acknowledgment frame can be of two types:

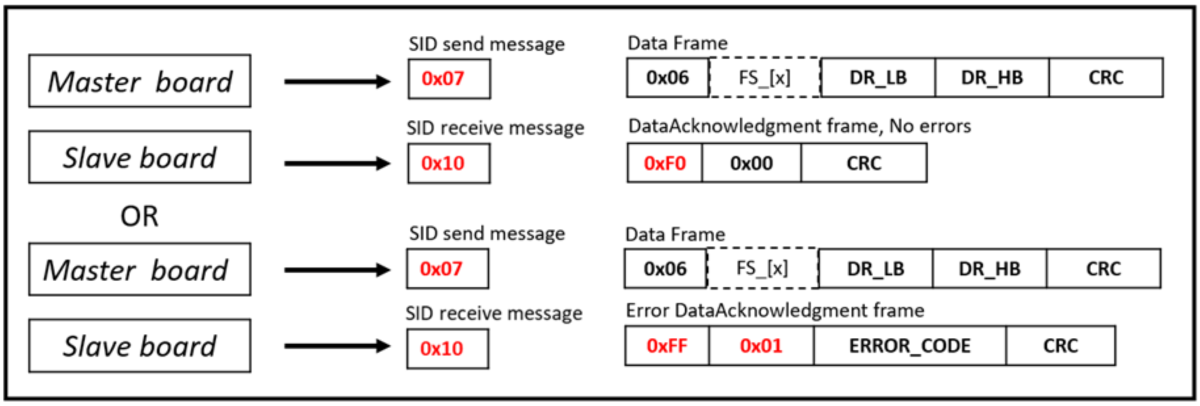
- data acknowledgment frame - if the operation has been successfully completed. In this case, the returned value embedded in the data acknowledgment frame is an echo of the same `COMMAND_ID`. The size of payload is always 1
- error acknowledgment frame - if the operation has not been successfully completed by the firmware. The payload is always 1. The list of error codes is shown in Table 2.

9.5 Execute ramp frame

The execute ramp frame is sent by the master to the motor control firmware, to request the execution of a speed ramp.

A speed ramp always starts from the current motor speed. The ramp is defined by a duration and a final speed

Figure 36. Execute ramp frame



Payload length is always 6.

The four bytes FS[x] represent the final speed expressed in rpm starting from the least significant byte followed by the most significant byte.

`DR_LB` and `DR_HB` represent the duration expressed in milliseconds, respectively least significant byte and most significant byte.

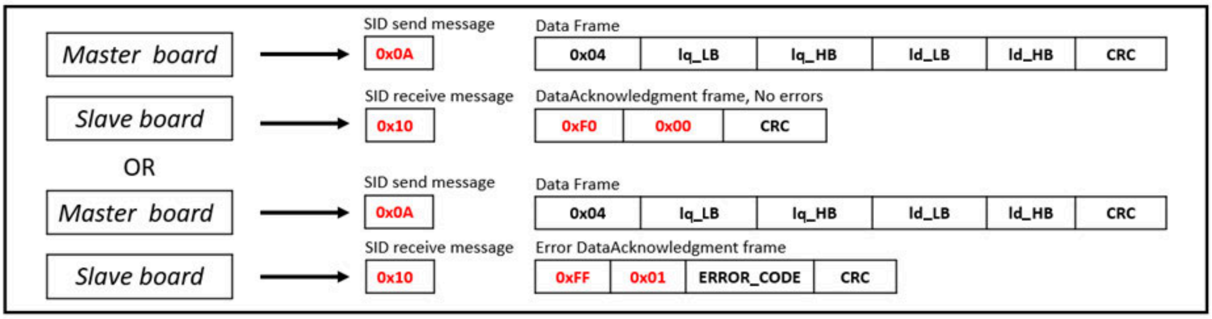
The acknowledgment frame can be of two types:

- data acknowledgment frame - if the operation has been successfully completed. The payload is zero
- error acknowledgment frame - if the operation has not been successfully completed by the firmware. The payload is always 1. The list of error codes is shown in Table 2.

9.6 Set current reference frame

The set current references frame is sent by the master to modify the current references (Iq and Id).

Figure 37. **Set current reference frame**



Payload length is always 4.

`Iq_LB` and `Iq_HB` are the requested new Iq references expressed in digits (respectively, the least significant byte and the most significant byte).

`Id_LB` and `Id_HB` are the requested new Id reference expressed in digits (respectively, the least significant byte and the most significant byte).

To convert current expressed in amps to current expressed in digits, use the following formula:

$$Current(digit) = [Current(Amp) \times 65536 \times R\_Shunt \times A\_OP]/V_{DD}micro$$

The acknowledgment frame can be of two types:

- data acknowledgment frame - if the operation has been successfully completed. The payload is zero
- error acknowledgment frame - if the operation has not been successfully completed by the firmware. The payload is always 1. The list of error codes is shown in Table 2.

10 Source code for the master board MCU

10.1 Overview

The source code to be flashed on the AEK-MCU-C4MLIT1 board MCU is the *SPC58ECxx_RLA_MainEcuForBLDCControl-L9908-Test Application* software package.

After loading it into SPC5-STUDIO, you can verify:

- how to configure the CAN protocol in the AEK-MCU-C4MLIT1 through SPC5-STUDIO;
- the *MotorControlCommunication* library;
- the Main.c file containing an example on how to interact via CAN messages with the slave board (AEK-MOT-3P99081).

10.2 CAN protocol configuration

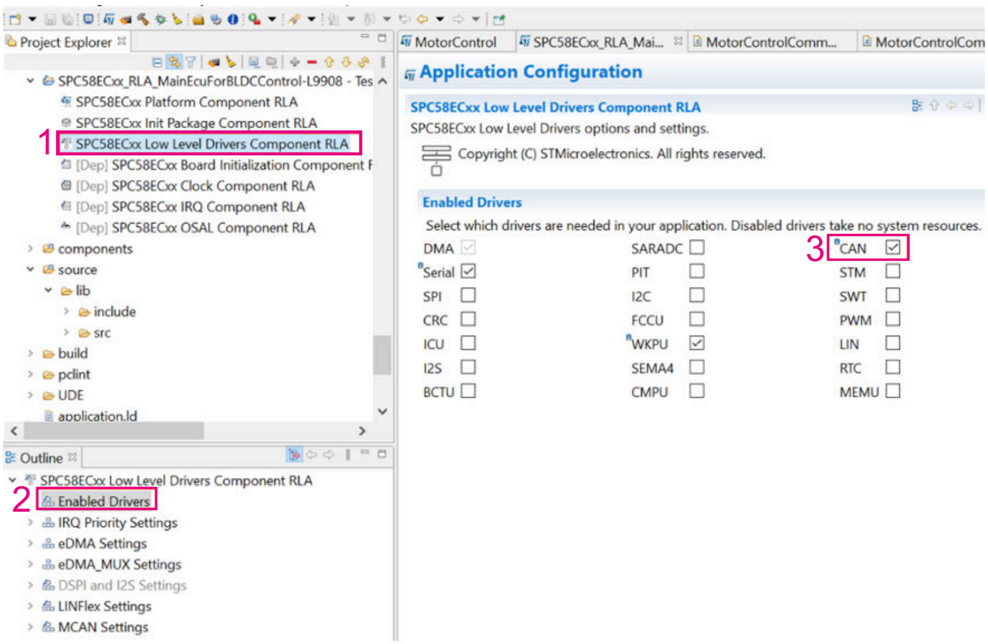
To configure the CAN protocol in an SPC5-STUDIO project follow the procedure below.

Step 1. Double-click on the low-level driver.

Step 2. Double-click on [**Enabled drivers**].

Step 3. Tick [**CAN**] drivers.

Figure 38. CAN protocol project driver selection

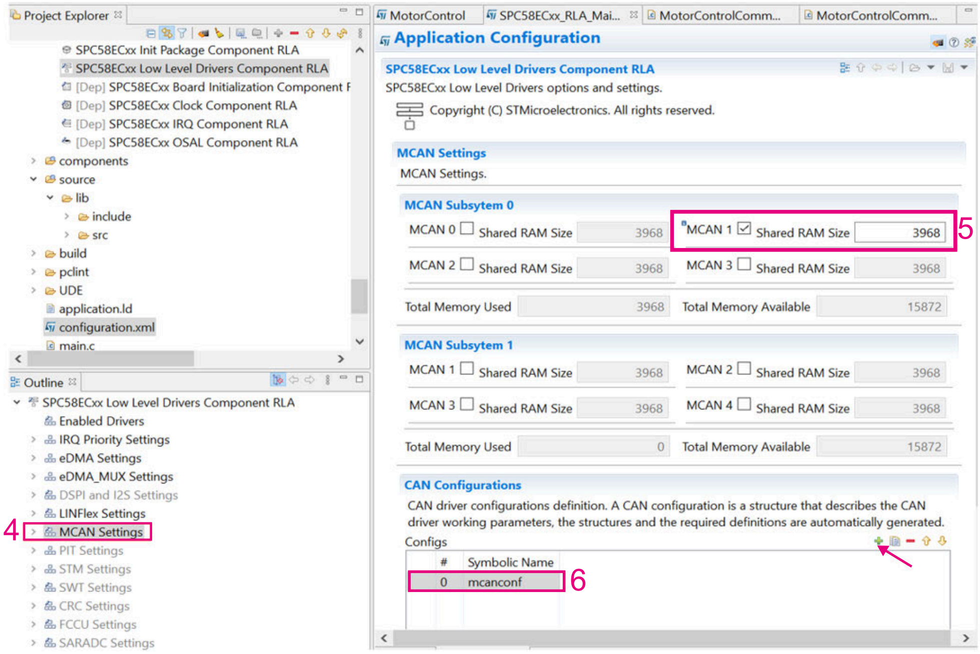


Step 4. Double-click on [**MCAN Settings**].

Step 5. Choose the CAN controllers for at least one subsystem.

Step 6. Double-click on [**Configs**] in CAN Configurations to open the CAN Configuration Window. If the configs object is not present, use the green "+" to add it.

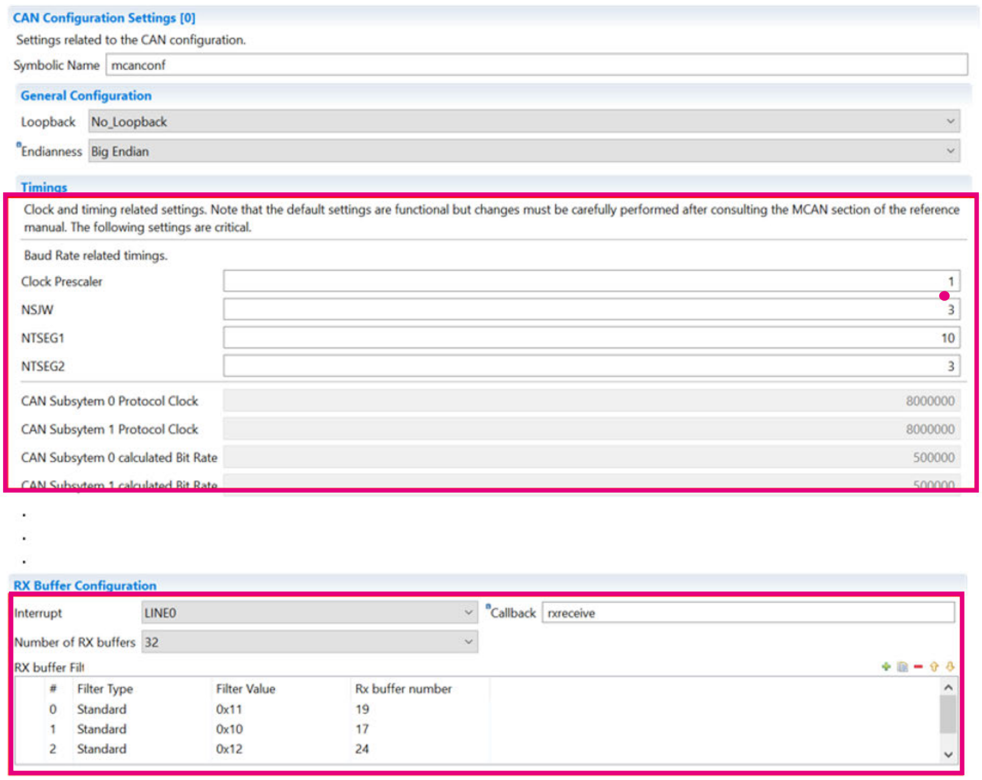
Figure 39. CAN configuration



Step 7. In the CAN configuration window, configure the settings according to your application requirements.

Note:
- *Pay attention to the timing parameters: choose the "Bit Rate" value taking into account the "Bit Rate" of the system with which you want to exchange CAN messages.*
- *In [**RX Buffer Configuration**], select the interrupt and the name of the callback that will be associated to the selected interrupt. Pay attention to the "RX buffer Filter" to add taking into account that the [**Filter Value**] represents the CAN message SID.*
- *Only the CAN messages that pass the CAN controller acceptance filters will be stored in the memory and processed by the callback previously defined.*

Figure 40. CAN configuration settings

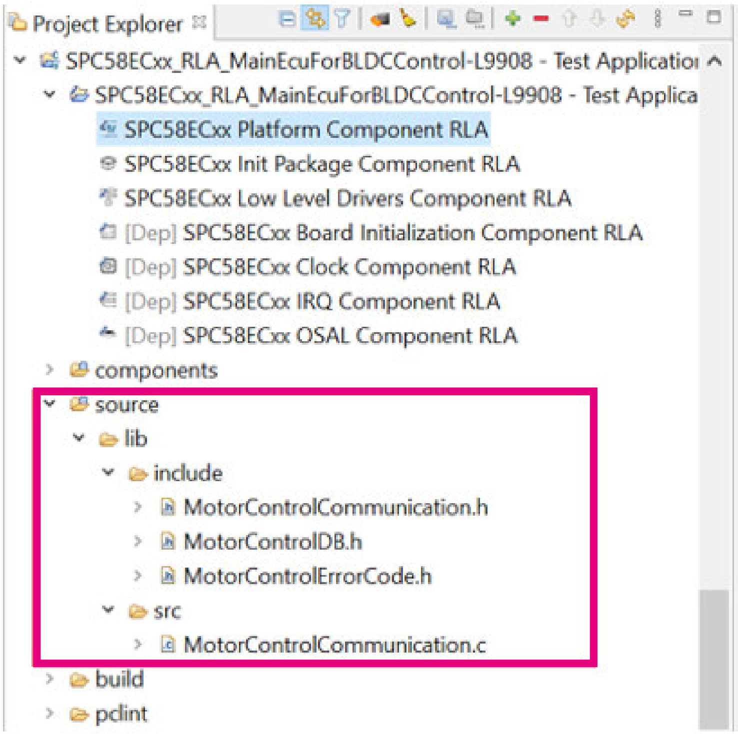


Note: *For more details about the CAN protocol and configuration see* Section Appendix A *.*

10.3 Motor Control Communication library

The MCP is included in the *SPC58ECxx_RLA_MainEcuForBLDCControl-L9908-Test Application* demo under the *MotorControlCommunication* library of the source folder.

Figure 41. **Motor Control Communication library**



The library exploits a set of functions implemented in the file MotorControlCommunication.c to request the AEK-MOT-3P99081 board to:

- set register frame
- get register frame
- execute command frame
- execute ramp frame
- set current reference frame

The .h files are:

- MotorControl.h - to define the constants and structure
- MotorControlDB.h - to define a DB containing all the relevant motor control registers
- MotorControlErrorCode - to define a DB containing error codes managed by the MCP

10.4 Demo implemented in main.c file

The source code demo consists of the following functions:

1. Init components
2. Enable Isr
3. Set up application

The first two instructions manage the initialization of the components related to MCU standard functions and the low-level drivers for the MCU peripherals, such as clock, external interrupt request queue (EIRQ), port configurations, etc. These two functions are automatically generated by SPC5-STUDIO from the configuration information provided during the creation/loading of the SPC5-STUDIO project.

The third function manages the initialization of all AutoDevKit components and MCU functionality implemented by the application.

Figure 42. **SPC58ECxx_RLA_MainEcuForBLDCControl-L9908-Test Application - main.c file extract**

```
/*
 * Application entry point.
 */
int main(void)
{

  /* Initialization of all the imported components in the order specified in
     the application wizard. The function is generated automatically.*/
  componentsInit();

  /* Enable Interrupts */
  irqIsrEnable();

  MotorControlCommunicationInit(&CAND2, &can_config_mcanconf);
  sd_lld_start(&SD5,&serial_config_debug);
  /* Configure and start WKPU Low Level Driver.*/
  wkpu_lld_start(&WKPUD1, &wkpu_config_configuration_name);

  /* Application main loop.*/
  for ( ; ; )
  {
    switch(state)
    {
        case SET_REGISTER:
            setRegister(registerString, value);
            state = GET_REGISTER;
             break;
        case EXEC_COMMAND:
            executeCommand(execCommandValue);
            state = 0;
             break;
        case GET_REGISTER:
            returnMotControlValue = getRegister(registerString);
            trace_value("Ramp_final_speed",returnMotControlValue->returnValue);
            value = returnMotControlValue->returnValue;
            state = 0 ;
             break;
        default:
             break;
    }
  }
}
```

The main.c file of the *SPC58ECxx_RLA_MainEcuForBLDCControl-L9908-Test Application* software package contains some macro-functions designed to drive, on a dedicated CAN-based communication protocol, the Hall-effect sensor three-phase BLDC motor control:

- key-press - to switch the application between the START/STOP of the BLDC motor when pushing the AEK-MCU-C4MLIT1 board SW3 button;
- turnUpRampSpeedBLDC - to increase the target speed of the BLDC motor when pushing the AEK-MCU-C4MLIT1 board SW1 button;
- turnDownRampSpeedBLDC - to decrease the target speed of the BLDC motor when pushing the AEK-MCU-C4MLIT1 board SW2 button;
- trace-error - to track the error coming from the communication between the master and the slave system boards;
- trace-value - to track the value of a register coming from the communication between the master and the slave system boards.

Note: The trace-error function, even if implemented, is not used in the main.c file.
The code below shows how to use this function in order to log the error code in a console.

```
returnValues_t * returnValue;
returnValue = setRegister("Ramp_final_speed", 500);
if(returnValue->status != CMD_OK)
{
trace_error((int8_t)(returnValue->returnValue));
}
returnValue = getRegister("Ramp_final_speed");
trace_value("Ramp_final_speed",returnValue->returnValue);
```

11 Schematic diagrams

Figure 43. AEK-MOT-3P99081 circuit schematic (1 of 5)

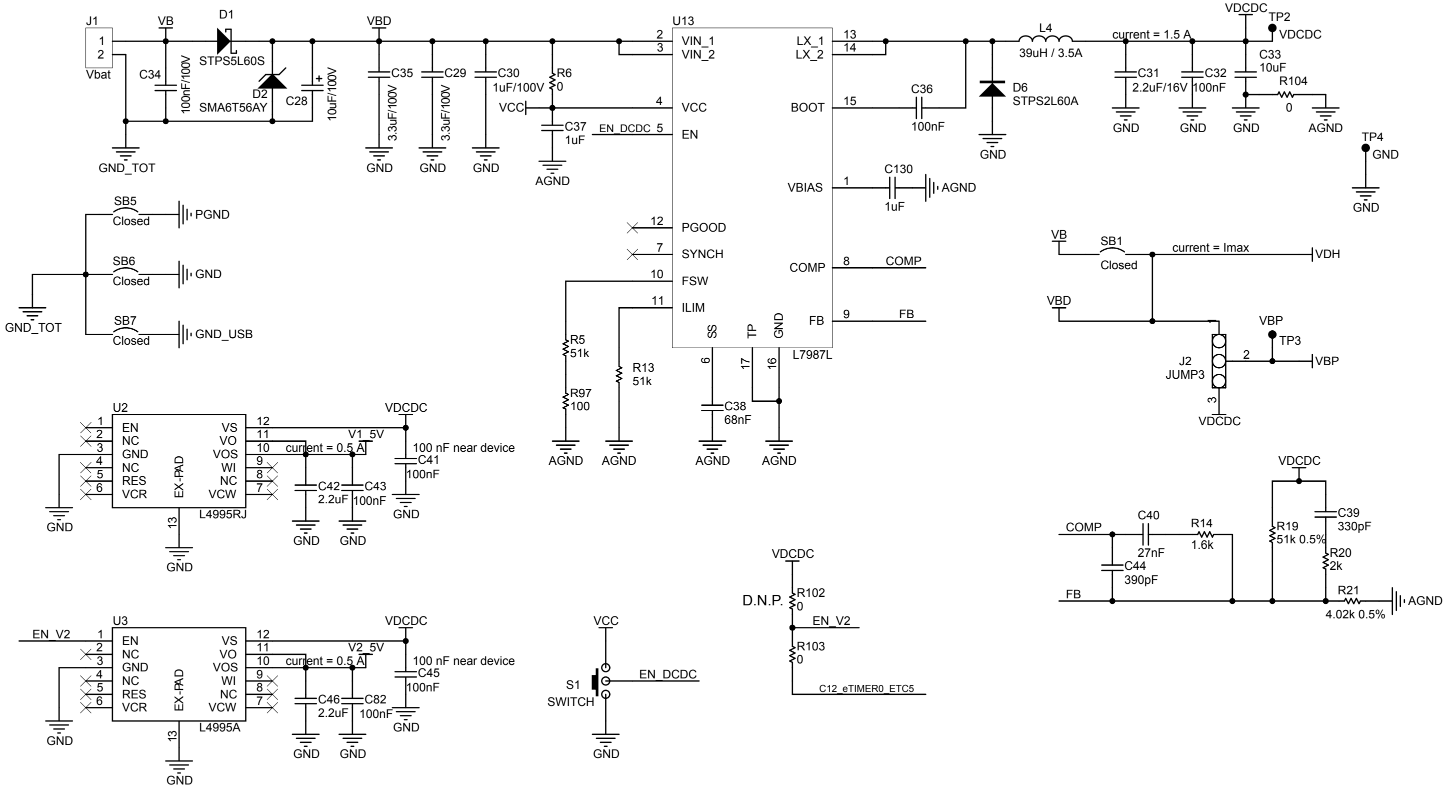


Figure 44. AEK-MOT-3P99081 circuit schematic (2 of 5)

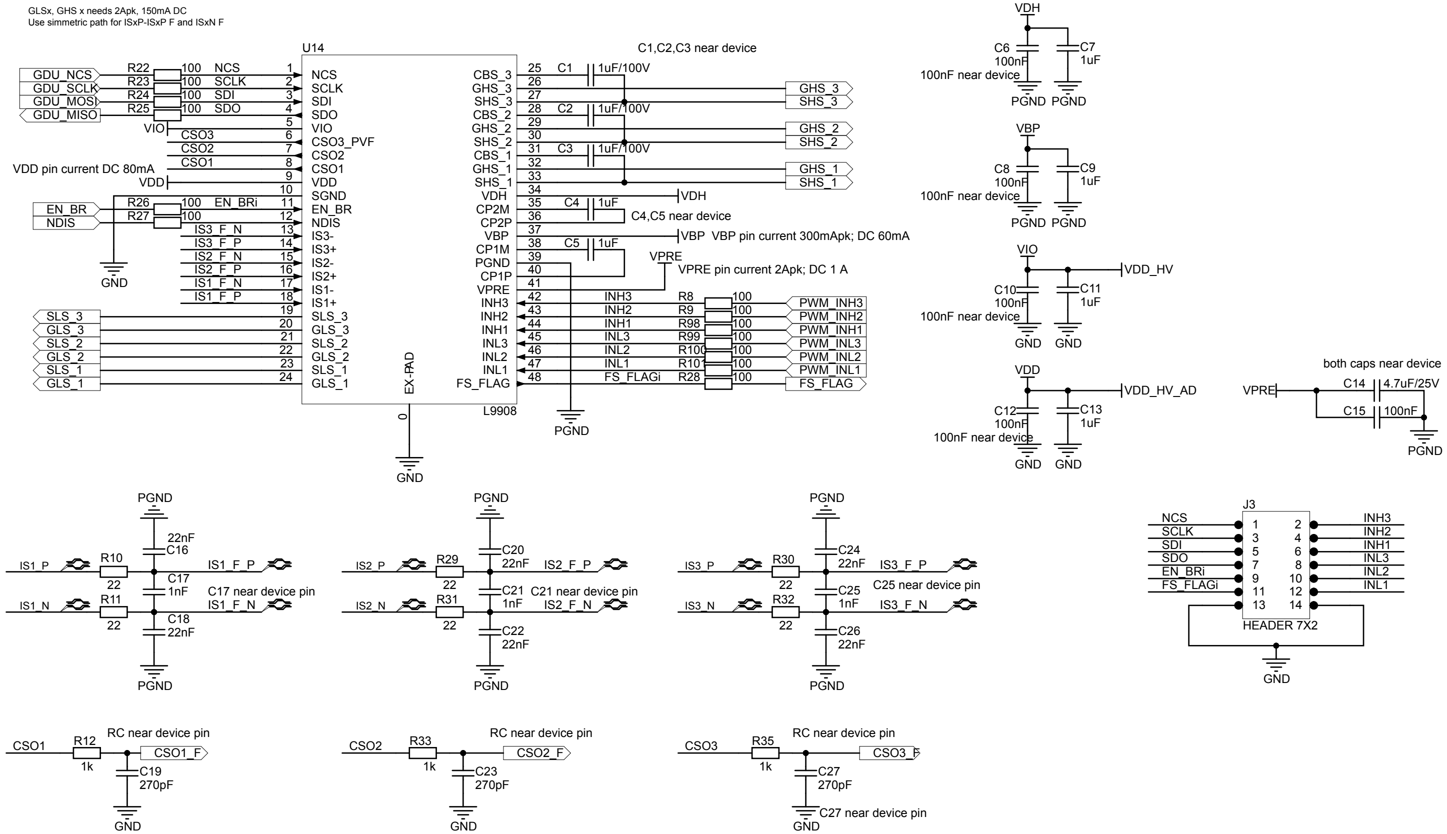


Figure 45. AEK-MOT-3P99081 circuit schematic (3 of 5)

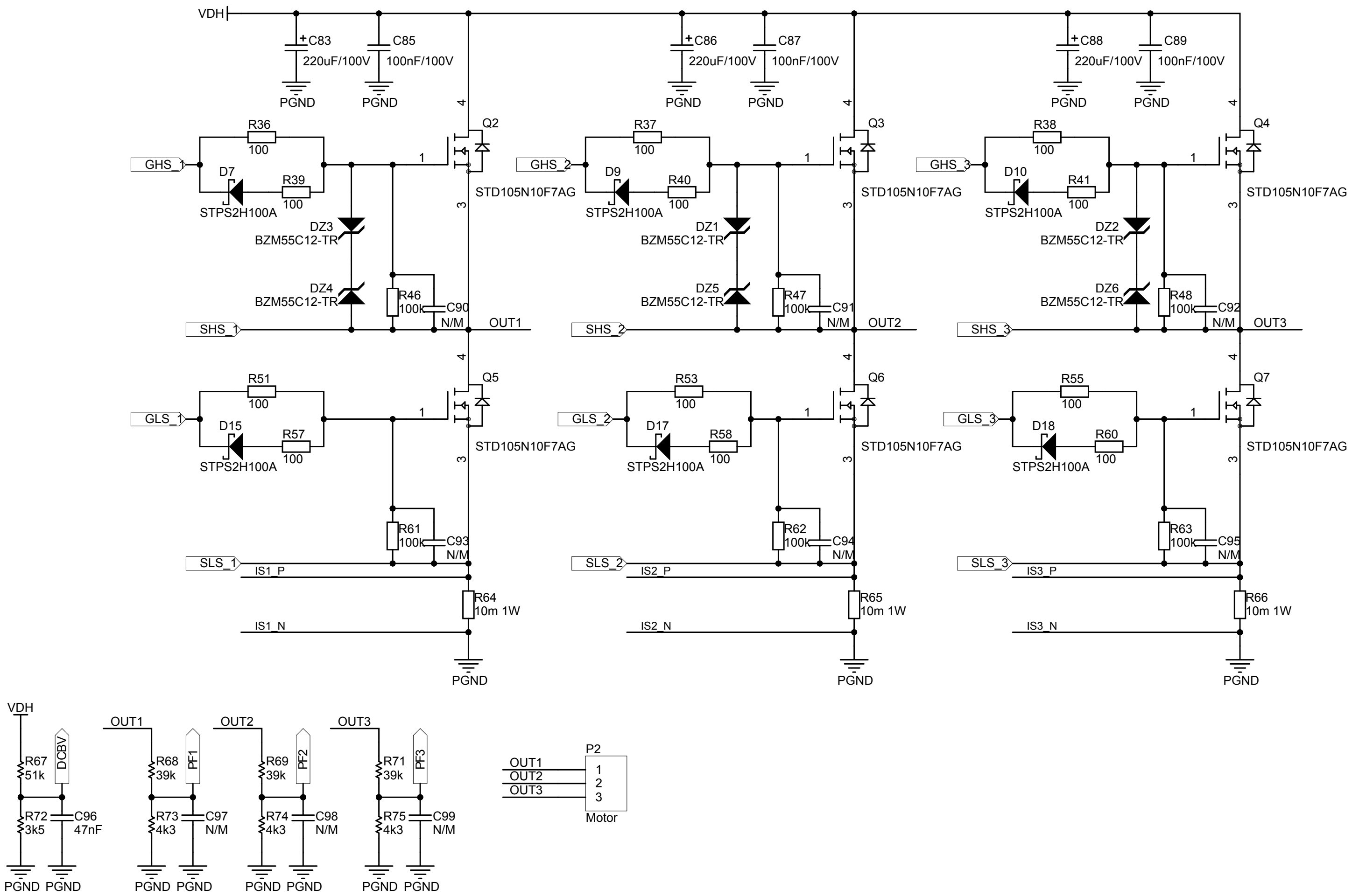


Figure 46. AEK-MOT-3P99081 circuit schematic (4 of 5)

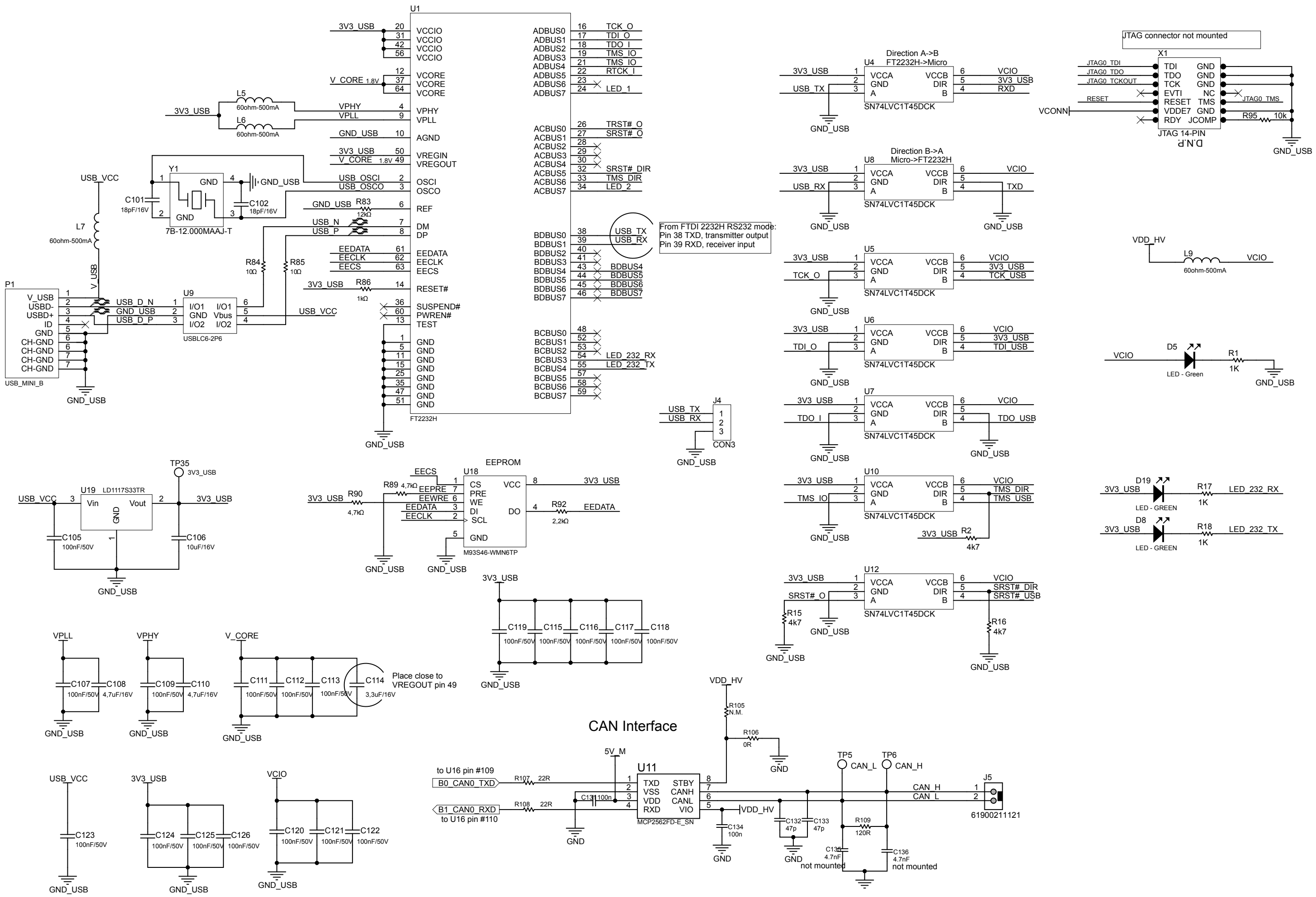
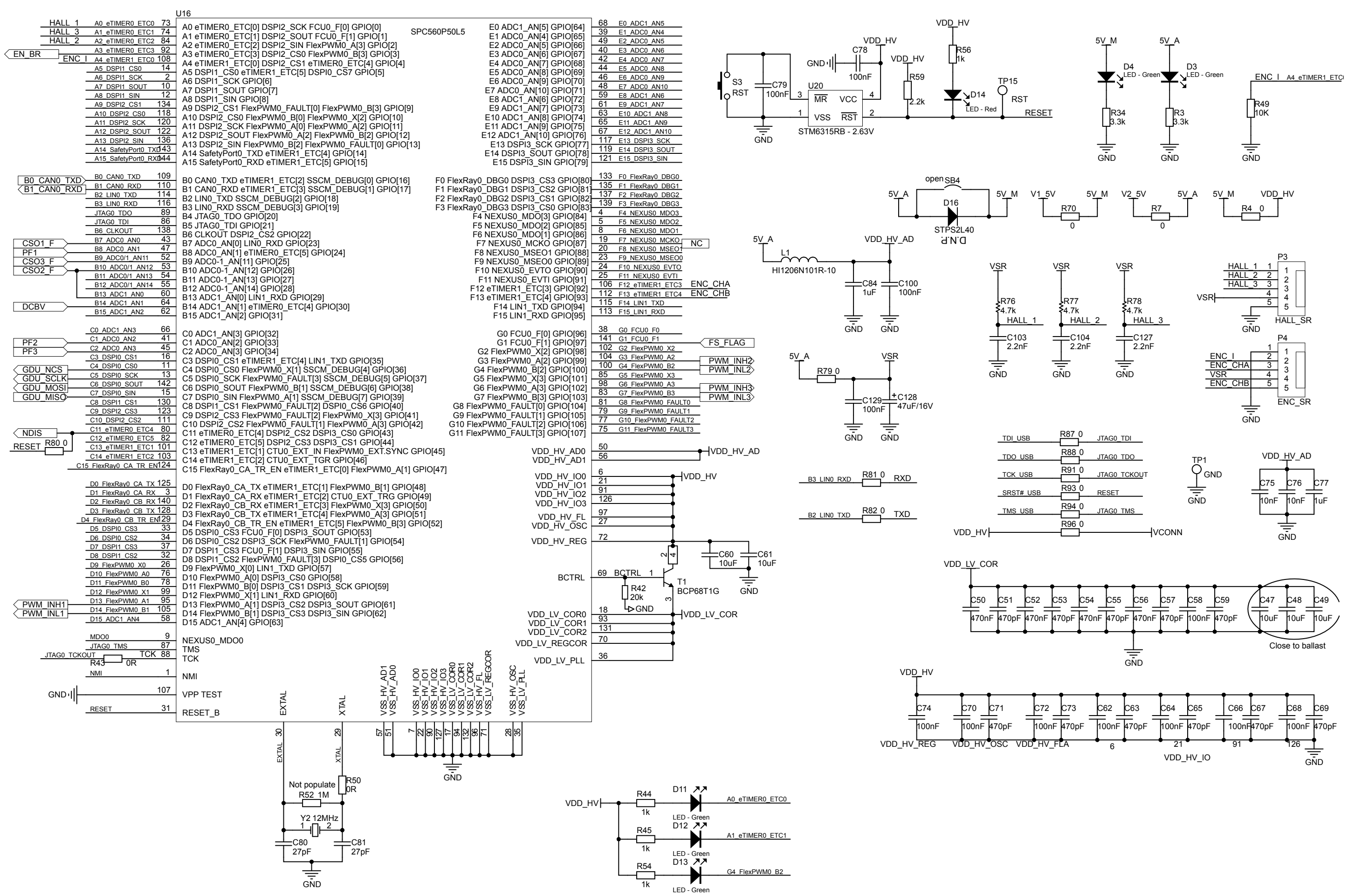


Figure 47. AEK-MOT-3P99081 circuit schematic (5 of 5)



12 Bill of materials

Table 6. AEK-MOT-3P99081 bill of materials

Item	Q.ty	Ref.	Part/Value	Description	Manufacturer	Order code
1	1	R105	D.N.P., R0603	SMD resistor	Any	
2	20	R4, R6, R7, R43, R50, R70, R79, R80, R81, R82, R87, R88, R91, R93, R94, R96, R102, R103, R106, SB4	0 Ohm, R0603, 1/10 W	SMD resistors	Vishay	CRCW06030000Z0EA
3	5	R104, SB1, SB5, SB6, SB7	0 Ohm, R1206, ¼ W	SMD resistors	Vishay	RCA12060000ZSEA
4	3	R64, R65, R66	0.010 Ohm, R2512, 1 W, ±1%	SMD resistors	Yageo	PE2512FKE070R01L
5	2	R84, R85	10 Ohm, R0603, 1/10 W, ±%	SMD resistors	Vishay	CRCW060310R0FKEB
6	8	R10, R11, R29, R30, R31, R32, R107, R108	22 Ohm, R0603, 1/3 W, ±5%	SMD resistors	Vishay	CRCW060322R0JNEAHP
7	26	R8, R9, R22, R23, R24, R25, R26, R27, R28, R36, R37, R38, R39, R40, R41, R51, R53, R55, R57, R58, R60, R97, R98, R99, R100, R101	100 Ohm, R0603, 1/10 W, ±%	SMD resistors	Vishay	CRCW0603100RFKTA
8	1	R109	120 Ohm, R0603, 1/10 W, ±5%	SMD resistors	Vishay	CRCW0603120RJNEA
9	11	R1, R12, R17, R18, R33, R35, R44, R45, R54, R56, R86	1 kOhm, R0603, 1/10 W, ±1	SMD resistors	Vishay	CRCW06031K00FKEC
10	1	R14	1.6 kOhm, R0603, 1/10 W, ±1%	SMD resistor	Vishay	CRCW06031K60FKEA
11	1	R20	2 kOhm, R0603, 1/10 W, ±5%	SMD resistor	Vishay	CRCW06032K00JNEA
12	2	R59, R92	2.2 kOhm, R0603, 1/10 W, ±%	SMD resistors	Vishay	CRCW06032K20FKEB

Item	Q.ty	Ref.	Part/Value	Description	Manufacturer	Order code
13	2	R3, R34	3.3 kOhm, R0603, 1/10W, +/-5%	SMD resistors	Vishay	CRCW06033K30JNEA
14	1	R72	3.48 kOhm, R0603, 1/10 W, ±%	SMD resistor	Vishay	MCT06030C3481FP500
15	1	R21	4.02 kOhm, R0603, 1/10W, ± 0.5%	SMD resistor	Yageo	RT0603DRE074K02L
16	3	R73, R74, R75	4.3 kOhm, R0603, 1/10 W, ±1%	SMD resistors	Vishay	MCWR06X4301FTL
17	8	R2, R15, R16, R76, R77, R78, R89, R90	4.7 kOhm, R0603, 1/10 W, ±5%	SMD resistors	Vishay	CRCW06034K70JNEAHP
18	2	R49, R95	10 kOhm, R0603, 1/10 W, ±5%	SMD resistors	Vishay	CRCW060310K0JNEAHP
19	1	R83	12 kOhm, R0603, 1/4 W, ±1%	SMD resistor	Vishay	RCS060312K0FKEA
20	1	R42	20 kOhm, R0603, 1/10 W, ±1%	SMD resistor	Vishay	CRCW060320K0FKEB
21	3	R68, R69, R71	39k Ohm, R0603, 1/4W, +/-1%	SMD resistors	Vishay	RCS060339K0FKEA
22	4	R5, R13, R19, R67	51 kOhm, R0603, 1/4 W, ±0.5%	SMD resistors	Panasonic	ERJ3RBD5102V
23	6	R46, R47, R48, R61, R62, R63	100 kOhm, R0603, 1/10 W, ±1%	SMD resistors	Vishay	CRCW0603100KFKEC
24	1	R52	1 MOhm, R0603, 1/10 W, ±5%	SMD resistor	Vishay	CRCW06031M00FKEAHP
25	9	C90, C91, C92, C93, C94, C95, C97, C98, C99	D.N.P, C0603	Ceramic capacitors	Any	
26	2	C101, C102	18 pF, C0603, 16 V, ±10%	Ceramic capacitors	TDK	CGA3E2C0G1H180J080A A
27	2	C80, C81	27 pF, C0603, 25 V, ±10%	Ceramic capacitors	TDK	CGA3E2C0G1H270J080A A
28	2	C132, C33	47 pF, C0603, 50 V, ±5%	Ceramic capacitors	TDK	C1608C0G1H470J080AA
29	3	C19, C23, C27	270 pF, C0603, 25 V, ±10%	Ceramic capacitors	TDK	CGA3E2C0G1H271J080A A
30	1	C39	330 pF, C0603, 25 V, ±10%	Ceramic capacitor	Wurth Elektronik	885012006060

Item	Q.ty	Ref.	Part/Value	Description	Manufacturer	Order code
31	1	C44	390 pF, C0603, 25 V, ±10%	Ceramic capacitor	TDK	CGA3E2C0G1H391J080AA
32	11	C51, C53, C55, C57, C59, C63, C65, C67, C69, C71, C73	470 pF, C0603, 25 V, ±10%	Ceramic capacitors	Wurth Elektronik	885012006061
33	3	C17, C21, C25	1 nF, C0603, 25 V, ±10%	Ceramic capacitors	Wurth Elektronik	885012006063
34	3	C103, C104, C127	2.2 nF, C0603, 25 V, ±10%	Ceramic capacitors	TDK	CGA3E2X7R1H222K080AA
35	2	C135, C136	4.7 nF, C0603, 50 V, ±10%	Ceramic capacitors	TDK	CGA3E2X7R1H472K080AA
36	2	C75, C76	10 nF, C0603, 25 V, ±10%	Ceramic capacitors	TDK	CGA3E2X7R2A103K080AA
37	6	C16, C18, C20, C22, C24, C26	22 nF, C0603, 25 V, ±10%	Ceramic capacitors	Wurth Elektronik	885012206091
38	1	C40	27 nF, C0603, 25 V, ±10%	Ceramic capacitor	Multicomp	MC0603B273K250CT
39	1	C96	47 nF, C0603, 25 V, ±10%	Ceramic capacitor	Wurth Elektronik	885012206093
40	1	C38	68 nF, C0603, 25 V, ±10%	Ceramic capacitor	Wurth Elektronik	885012206094
41	43	C6, C8, C10, C12, C15, C32, C36, C41, C43, C45, C58, C62, C64, C66, C68, C70, C72, C74, C78, C79, C82, C100, C105, C107, C109, C111, C112, C113, C115, C116, C117, C118, C119, C120, C121, C122, C123, C124, C125, C126, C129, C131, C134	100 nF, C0603, 50 V, ±10%	Ceramic capacitors	Wurth Elektronik	885012206095
42	4	C34, C85, C87, C89	100 nF, C0805, 100 V, ±10%	Ceramic capacitors	Wurth Elektronik	885012207128
43	4	C50, C52, C54, C56	470 nF, C0603, 25 V, ±10%	Ceramic capacitors	TDK	CGA3E3X5R1H474K080AB

Item	Q.ty	Ref.	Part/Value	Description	Manufacturer	Order code
44	10	C4, C5, C7, C9, C11, C13, C37, C77, C84, C130	1 µF, C0805, 25 V, ±10%	Ceramic capacitors	Wurth Elektronik	885012207103
45	4	C1, C2, C3, C30	1 µF, C0805, 100 V, ±10%	Ceramic capacitors	TDK	CGA4J3X7S2A105K125AB
46	3	C31, C42, C46	2.2 µF, C0805, 16 V, ±10%	Ceramic capacitors	Wurth Elektronik	885012207052
47	1	C114	3.3 µF, C0805, 16 V, ±10%	Ceramic capacitor	TDK	CGA4J3X7R1C335K125AD
48	2	C29, C35	3.3 µuF, C1206, 100 V, ±10%	Ceramic capacitors	TDK	CGA5L3X7S2A335K160AB
49	2	C108, C110	4.7 µF, C1206, 16 V, ±10%	Ceramic capacitors	TDK	CGA5L1X7R1E475K160AC
50	1	C14	4.7 µF, C1206, 25 V, ±10%	Ceramic capacitor	Wurth Elektronik	885012208068
51	7	C33, C47, C48, C49, C60, C61, C106	10 µF, C1206, 16 V, ±10%	Ceramic capacitors	Wurth Elektronik	885012208041
52	1	C28	10 µF, CAP. 8.3 X 6.2 - SMD, 100 V, ±20%	Electrolytic capacitor	Panasonic	EEE2AA100UP
53	1	C128	47 µF, CAP. 6.3 X 5.4 - SMD, 16 V, ±20%	Electrolytic capacitor	Wurth Elektronik	865230343004
54	3	C83, C86, C88	220 µF, CAP. 12.5 x 25 mm, 100 V, ±20%	Electrolytic capacitors	Wurth Elektronik	860040878017
55	4	L5, L6, L7, L9	60 Ohm, F0603, ±25%	Ferrite beads	Wurth Elektronik	74279267
56	1	L1	100 Ohm, F1206	Ferrite bead	TDK	MPZ2012S101ATD25
57	1	L4	39 µH, 744770139, 3.5 A, ±20%	Inductor	Wurth Elektronik	744770139
58	1	D14	0805 LED	Red LED	Wurth Elektronik	150080RS75000
59	8	D3, D4, D5, D8, D11, D12, D13, D19	0805 LED	Green LED	Wurth Elektronik	150080GS75000
60	1	D1	STPS5L60S, DO-214AB	60 V, 5 A low drop power Schottky rectifier	ST	STPS5L60S
61	1	D2	SMA6T56AY, DO-214AC	Automotive 600 W, 47.6 V TVS in SMA	ST	SMA6T56AY

Item	Q.ty	Ref.	Part/Value	Description	Manufacturer	Order code
62	1	D6	STPS2L60A, DO-214AC	60 V, 2 A low drop power Schottky rectifier	ST	STPS2L60A
63	6	D7, D9, D10, D15, D17, D18	STPS2H100A , DO-214AC	100 V, 2 A power Schottky rectifier	ST	STPS2H100A
64	1	D16	STPS2L40, DO-214AA	40 V, 2 A low drop power Schottky rectifier	ST	STPS2L40
65	6	DZ1, DZ2, DZ3, DZ4, DZ5, DZ6	BZM55C12-TR, MICRO-MELF	Zener diode	Vishay	BZM55C12-TR
66	1	Y1	7B-12.000MAAJ-T	Crystal oscillator	Wurth Elektronik	830055999
67	1	Y2	XTAL003210, HC49/4H SMX	Crystal oscillator	Wurth Elektronik	830003210
68	1	S3	TSB06339-050-4 (180, KTH55150KA	Tactile switch	ADIMPEX	TSB06339-050-4 (180
69	1	S1	MMS 228 T (SPDT)	Slide switch	KNITTER-SWITCH	MMS 228 T
70	8	TP1, TP2, TP3, TP4, TP5, TP6, TP15, TP35	TEST POINT	Test points	Any	Any
71	2	J2, J4	p. 2.54 mm, SIP3	Headers	Wurth Elektronik	61300311121
72	1	J3	p. 2.54 mm, HEADER 7X2	Header	Wurth Elektronik	61301421121
73	1	J5	p. 2.54 mm, 61900211121	Polarized header	Wurth Elektronik	61900211121
74	2	P3, P4	p. 2.54 mm, SIP5	Polarized headers	Wurth Elektronik	61900511121
75	1	X1	p. 2.54 mm, IDC14	Boxed header	Wurth Elektronik	61201421621
76	1	J1	p. 5.08 mm, CON2	Connector cart	Wurth Elektronik	691213510002
77	1	P2	p. 5.08 mm, CON3	Connector cart	Wurth Elektronik	691213510002
78	1	P1	MINI USB	Mini-USB	Wurth Elektronik	65100516121
79	6	Q2, Q3, Q4, Q5, Q6, Q7	STD105N10F7AG, DPAK	Automotive-grade N-channel 100 V, 6.8 mOhm typ., 80 A STripFET F7 power MOSFET in a DPAK package	ST	STD105N10F7AG

Item	Q.ty	Ref.	Part/Value	Description	Manufacturer	Order code
80	1	T1	BCP68T1G, SOT-223	Transistor	OnSemiconductor	BCP68T1G
81	1	U19	LD1117S33TR, SOT-223	Adjustable and fixed low drop positive voltage regulator	ST	LD1117S33TR
82	1	U13	L7987L, HTSSOP-16	61 V 2 A asynchronous step-down switching regulator with adjustable current limitation	ST	L7987L
83	1	U2	L4995RJ, PowerSSO-12	Automotive 5 V, 500 mA low drop voltage regulator	ST	L4995RJ
84	1	U3	L4995AJ, PowerSSO-12	Automotive 5 V, 500 mA low drop voltage regulator	ST	L4995AJ
85	1	U14	L9908, TQFP48	Automotive 3-phase motor gate driver unit	ST	L9908
86	1	U16	SPC560P50L5, LQFP144	32-bit power architecture MCU for automotive chassis and safety applications	ST	SPC560P50L5
87	1	U20	STM6315RBW13F, SOT-143	Open drain microprocessor reset	ST	STM6315RBW13F
88	1	U1	FT2232HL-REEL, QFP64	USB-to-UART	FTDI	FT2232HL-REEL
89	1	U9	USBLC6-2P6, SOT-666	ESD Protection for USB 2.0 High Speed	ST	USBLC6-2P6
90	1	U18	M93S46-WMN6TP, SO-8	1-Kbit MICROWIRE serial access EEPROM with block protection	ST	M93S46-WMN6TP
91	7	U4, U5, U6, U7, U8, U10, U12	SN74LVC1T45DCKT, SC-70	Transceivers	Texas Instruments	SN74LVC1T45DCKT
92	1	U11	MCP2562FD-E/SN, SO-8	CAN bus transceiver	Microchip	MCP2562FD-E/SN

13 Board versions

Table 7. AEK-MOT-3P99081 versions

Finished good	Schematic diagrams	Bill of materials
AEK$MOT-3P99081A [1]	AEK$MOT-3P99081A schematic diagrams	AEK$MOT-3P99081A bill of materials

1. *This code identifies the AEK-MOT-3P99081 evaluation board first version.*

14 Regulatory compliance

Formal Notice Required by the U.S. Federal Communications Commission

FCC NOTICE:

This kit is designed to allow:

(1) Product developers to evaluate electronic components, circuitry, or software associated with the kit to determine whether to incorporate such items in a finished product and

(2) Software developers to write software applications for use with the end product.

This kit is not a finished product and when assembled may not be resold or otherwise marketed unless all required FCC equipment authorizations are first obtained. Operation is subject to the condition that this product not cause harmful interference to licensed radio stations and that this product accept harmful interference. Unless the assembled kit is designed to operate under part 15, part 18 or part 95 of this chapter, the operator of the kit must operate under the authority of an FCC license holder or must secure an experimental authorization under part 5 of this chapter 3.1.2.

The evaluation kit has been designed to comply with part 15 of the FCC Technical Rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation.

Standard applied: FCC CFR 47 Part 15 Subpart B. Test method applied: ANSI C63.4 (2014).

Formal Product Notice Required by Industry Canada Innovation, Science and Economic Development

Canada compliance:

For evaluation purposes only. This kit generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to Industry Canada (IC) rules.

À des fins d'évaluation uniquement. Ce kit génère, utilise et peut émettre de l'énergie radiofréquence et n'a pas été testé pour sa conformité aux limites des appareils informatiques conformément aux règles d'Industrie Canada (IC).

This device has been tested with Innovation, Science and Economic Development RSS standards. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

Standard applied: ICES-003 Issue 7 (2020), Class B. Test method applied: ANSI C63.4 (2014).

Cet appareil a été testé pour les normes RSS d'Innovation, Science et Développement économique. L'utilisation est soumise aux deux conditions suivantes: (1) cet appareil ne doit pas causer d'interférences nuisibles, et (2) cet appareil doit accepter de recevoir tous les types d’interférence, y comprises les interférences susceptibles d'entraîner un fonctionnement indésirable.

Norme appliquée: NMB-003, 7e édition (2020), Classe B. Méthode d'essai appliquée: ANSI C63.4 (2014).

Formal product notice required by EU

This device is in conformity with the essential requirements of the Directive 2014/30/EU (EMC) and of the Directive 2015/863/EU (RoHS).

Standards applied: EN 61000-6-1:2019, EN 61000-6-3:2007 + A1:2011 + AC:2012, EN 55032:2015 + A1:2020, EN 55035:2017 + A11:2020, EN 61000-3-2:2019, EN 61000-3-3:2013 + A1:2019.

Appendix A CAN protocol in automotive applications

The car area network (CAN) protocol is widely used in automotive body and convenience applications where the main body control module (BCM) is connected with several electrical control units (ECUs).

Typical in-vehicle CAN messages belong to the following categories:

1. Messages triggered by specific events
2. Message cyclically sent for safety reasons

The ECU normally manages cyclic messages with a scheduler to ensure appropriate timing of transmission.
The SPC58ECxx_RLA_MainEcuForBLDCControl-L9908-Test Application demo software only implements basic communication with the IPC. For more information regarding CAN communication protocol and set-up in SPC5-STUDIO, refer to AN5416.

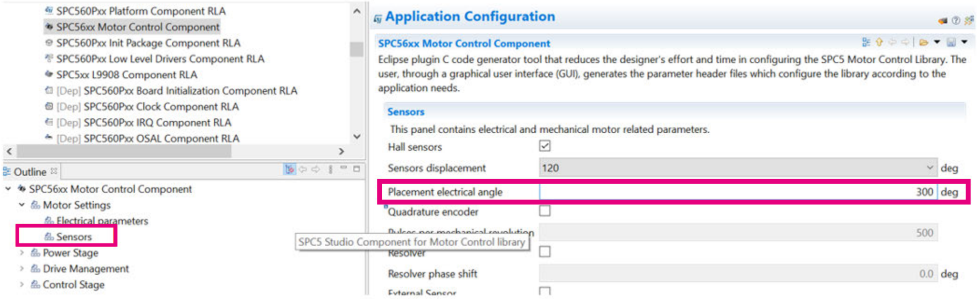
Appendix B Calculate electrical angle when using Hall sensors

The electrical angle is an important parameter that must be set when motor control algorithm inputs come from Hall sensors.

The Hall sensors provide absolute rotor position information, but, due to construction reasons, they are often not perfectly mounted. Even a small misalignment with the center of the stator winding of phase A might generate an error in the measurement of the angle that degrades the control.

This important parameter has to be added in the sensor section of the motor control library configuration.

Figure 48. Hall sensor configuration

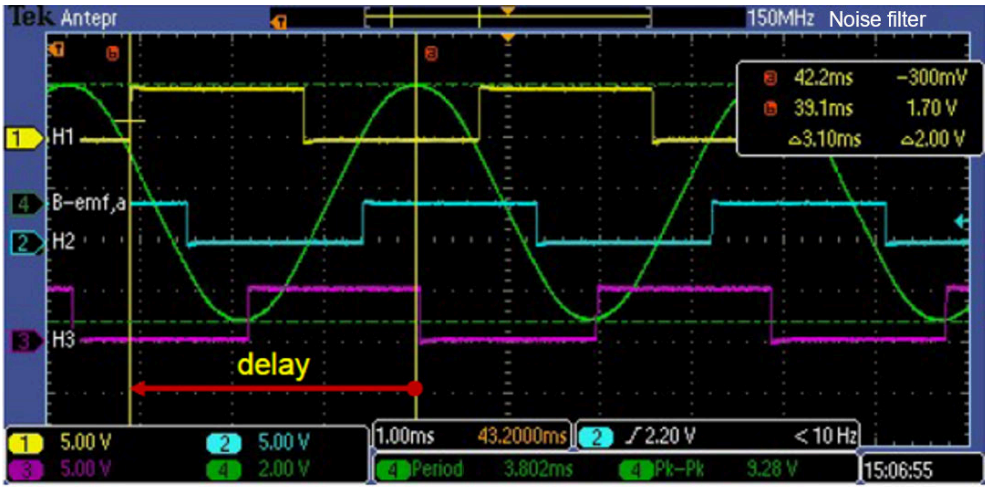


To calculate this value, follow the procedure below.

Step 1. Plot the following signals in the oscilloscope:

- Back Emf Phase A (B-emf, a in the image below)
- Hall Phase A (H1 in the image below)
- Hall Phase B (H2 in the image below)
- Hall Phase C (H3 in the image below)

Figure 49. Oscilloscope plot



Step 2. Measure:

- electrical period calculated between two rising edges of H1 = 3.78 ms in this case
- electrical period of Back Emf Phase A = 3.802 ms in this case

Step 3. Compute:

- delay by subtracting the rising edge of signal H1 time from the max point of B-emf on phase A 42.2 ms – 39.1 ms = 3.1 ms
- $$Phase\ Delta = \frac{Delay}{Electrical\ Period} \times 360\ \text{deg} = \frac{3.1ms}{3.78ms} \times 360\ \text{deg} = 295.24\ \text{deg}$$

Revision history

Table 8. **Document revision history**

Date	Revision	Changes
13-Sep-2021	1	Initial release.

Contents

List of tables

List of figures