MCSDK expansion package for EVALKIT-ROBOT-1

Introduction

The STM32 motor control software development kit (X-CUBE-MCSDK) offers a complete toolset for brushless driving applications starting from a wide board portfolio.

The MCSDK-PKG expansion packages for MCSDK further enrich this offer, adding the possibility to start a design from a predesigned application example or adding new hardware on the board library.

Customization is made simple by the motor control workbench (included in the MCSDK) and in a few steps the user can adapt the project to their needs.

1 MCSDK-PKG002 expansion package

The MCSDK-PKG002 is an MCSDK expansion package for the EVALKIT-ROBOT-1, allowing a simple customization of the control firmware through the motor control workbench included in the STM32 motor control software development kit (X-CUBE-MCSDK).

The EVALKIT-ROBOT-1 evaluation kit offers a ready-to-use brushless servomotor solution composed by an STSPIN32F0A control board and a maxon EC-i 40 brushless DC motor.

Figure 1. EVALKIT-ROBOT-1

2 Hardware and software requirements

The MCSDK-PKG002 requirements are the following:

- One EVALKIT-ROBOT-1
- A 36 V / 120 W DC power supply (1)
- An RS-485 2-wire serial port
- A communication software based on the MODBUS protocol
- X-CUBE-MCSDK
- STM32CubeMX
- One of the supported development toolchains:
  - IAR Embedded Workbench for Arm
  - Keil® microcontroller development kit
  - STM32CubeIDE
- An SWD programmer/debugger supporting the STM32F0 family, like the STLINK-V3 from STMicroelectronics.

1. *The operating range of the control electronics is between 12 V and 45 V. However, the system provides the best performance with a supply voltage of 36 V ± 20%.*

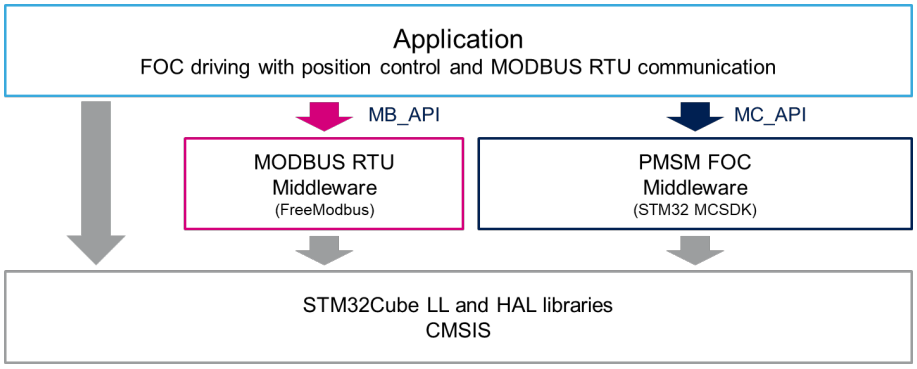
3 Getting started

This firmware example is based on the MCSDK implementing high performance PMSM FOC control with a positioning feature. The application firmware interfaces with the middleware via the specific API (MC_API).

The MODBUS RTU communication is based on the FreeModbus library and the application firmware interfaces with the middleware via the specific API (MB_API).

Peripheral management is performed by the STM32Cube low level diver (LL) and hardware abstraction layer (HAL) based on the standard CMSIS library.

Figure 2. Firmware architecture



3.1 Folders structure

This section provides an overview of the package content:

- **EVALKIT-ROBOT-1**: project folder
- **_htmsresc**: folder containing resources for readme and release note
- **readme.html**: brief quickstart guide
- **release-note.html**: release note of the package
- **manifest.json**: package manifest

The **EVALKIT-ROBOT-1** folder contains:

- **EVALKIT-ROBOT-1.ioc**: IOC file that can be used to generate customized code using the motor control workbench
- **Inc**: Application include files folder
- **Src**: Application source files folder
- **modbus**: MODBUS communication middleware folder
- **EWARM**: IAR Embedded Workbench project folder
- **MDK-ARM**: Keil-ARM uVision project folder
- **STM32CubeIDE**: STM32CubeIDE project folder

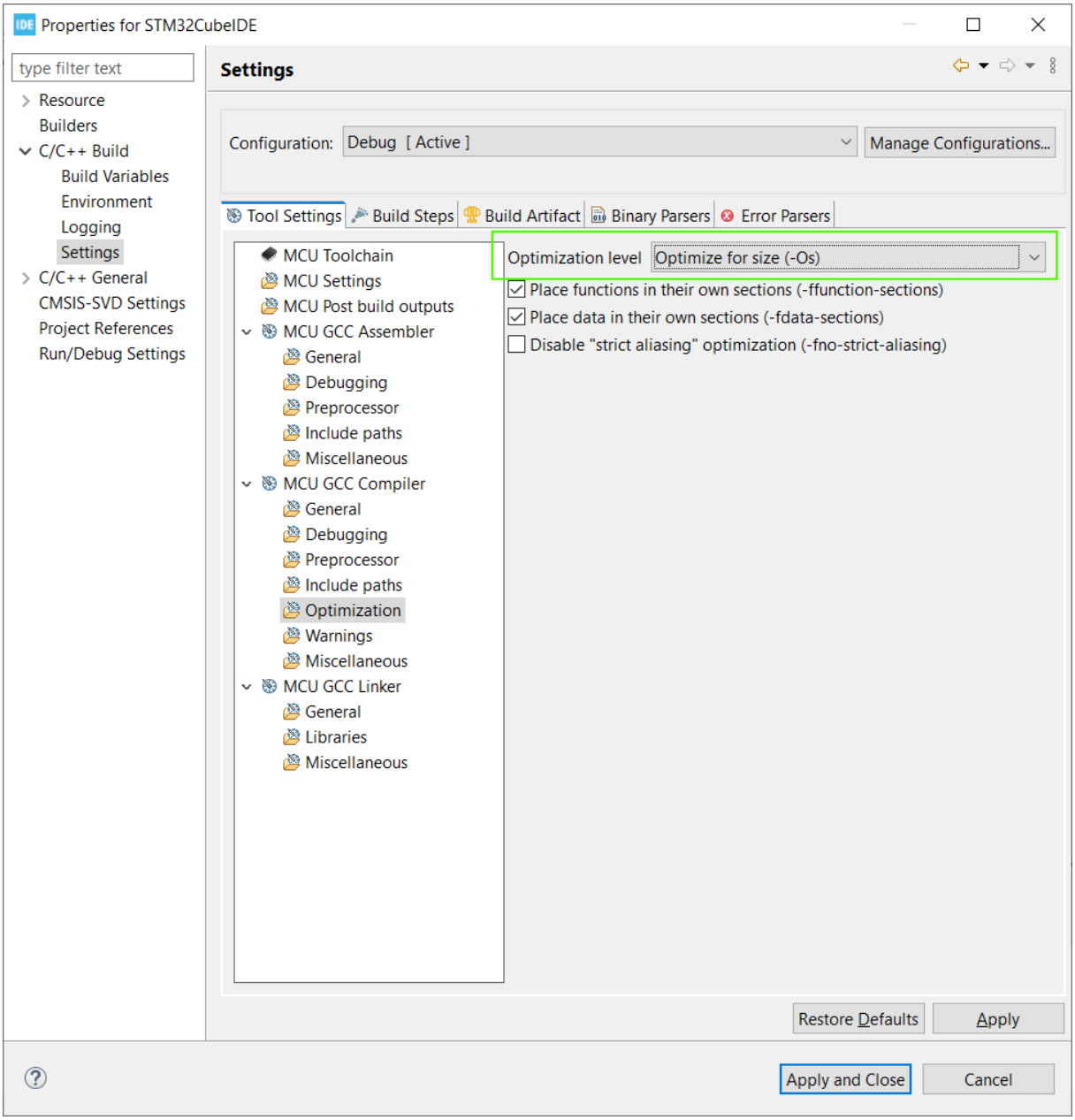
3.2 Code generation and EVALKIT-ROBOT-1 programming

Follow the step-by-step procedure to create and customize the project:

1. Extract the content of the example folder into your workspace.
2. Open the related **EVALKIT-ROBOT-1.ioc** file using the motor control workbench.
3. The project is already configured targeting the maxon EC-i 40 motor.
   However, it is possible to customize it according to the specific application (for instance: new motor parameters, control loop parameters, ...) as described in section 4.
   Consider that not all the parameters are editable. If the user just wants to regenerate the code, move to step 4.
4. Save the project **without changing the file location** and click on *"Generate the project"* selecting **LL libraries** and the preferred IDE from the supported ones.

5. Open the generated project with the previously selected IDE.
If STM32CubeIDE was selected, "Optimize for size" must be enabled in the MCU GCC compiler settings.

Figure 3. STM32CubeIDE - optimize for size setting



6. Build the project and program the board using the J12 connector.

Note: The board must be supplied through the J7 screw terminal before programming.

4 Code description

4.1 Main application code

The main application is composed by the following sections:

1. **Initialization**: peripherals are configured and data structures are initialized
2. **Encoder alignment**: mechanical alignment procedure is executed
3. **Main loop**: infinite loop monitoring command request, execution, and faults management
4. **Motor control loop (interrupt based)**: executes the PMSM FOC algorithm (part of MCSDK)
5. **MODBUS loop (interrupt based)**: manages the MODBUS RTU communication

4.2 MC_API

The motor control API (Src\mc_api.c) is part of the motor control SDK and provides the key functions controlling the motor.

Following are listed the functions used by the firmware example, for more details about the motor control SDK API refer to the respective user manual.

- **MC_StartMotor1()**: initiates the startup procedure for the motor including the encoder alignment
- **MC_StopMotor1()**: initiates the stop procedure for the motor
- **MC_ProgramPositionCommandMotor1()**: programs the execution of a position command indicating the target mechanical position (*fTargetPosition*) in radian and the expected execution time (*fDuration*) in seconds
- **MC_GetControlPositionStatusMotor1()**: returns the status of the positioning control state machine
- **MC_GetAlignmentStatusMotor1()**: returns the status of the encoder alignment (startup sequence)
- **MC_GetOccurredFaultsMotor1()**: returns a bitfield showing the new faults detected by the motor control algorithm
- **MC_AcknowledgeFaultMotor1()**: acknowledges a motor control fault occurred during motor control enabling a new command execution

4.3 MB_API

The MODBUS RTU API (modbus\mb_API.c/.h) manages the direct inputs, coils, input registers, and holding registers of the application.

- **SETMOVMENTCOMPLETED**: macro setting high the DONE direct input
- **SETMOVEMENTONGOING**: macro setting low the DONE direct input
- **SETZEROREACHED**: macro setting high the ALIGN_OK direct input
- **SETZEROONGOING**: macro setting low the ALIGN_OK direct input
- **CLEARFAULT**: macro setting low the FAULT direct input
- **SETFAULT**: macro setting high the FAULT direct input
- **GETNEWCOMMAND**: macro checking the MOVE coil
- **GETNEWALIGNMENT**: macro checking the ALIGN coil
- **GETFAULTACK**: macro checking the FAULT_ACK coil
- **CLEARCOMMAND**: macro forcing the MOVE coil low
- **CLEARALIGN**: macro forcing the ALIGN coil low
- **CLEARFAULTACK**: macro forcing the FAULT_ACK coil low
- **CLEARALL**: macro forcing all the coils low
- **setZeroReached**: sets high or low the ALIGN direct input
- **getTargetPos_Rad**: returns the target position stored in the holding registers (degrees) in radians
- **getMovementDuration_s**: returns the target movement duration stored in the holding registers (milliseconds) in seconds
- **updateStatusRegs**: update the status stored in the input registers
- **initPIDParamsRegs**: initialize the holding registers with current PID gains
- **updatePIDParams**: update PID gains according to the holding registers content

5 Customization

5.1 Customization allowed by motor control workbench

The motor control workbench allows to configure the parameters of the motor control algorithm as:

- Motor characteristics and nominal supply voltage
- Overvoltage and undervoltage protection thresholds
- Overcurrent protection threshold
- PWM frequency and deadtime
- Torque and flux current controls PID gains
- Position control PID gain

5.2 MODBUS section customization

The MODBUS section (that is, slave address, coils, direct inputs, holding registers, etc.) can be customized modifying the following code files:

- **mb_API.c/.h**: API interfacing the MODBUS with main application level
- **mbtask.c/.h**: implementation of the callbacks managing
  - Holding registers read/write
  - Input registers read
  - Coils read/write
  - Discrete inputs read

The parameters defining the MODBUS data model and slave address are as follows:

Table 1. MODBUS configuration parameters

Parameter	Location	Description
MB_SLAVE_ADDRESS	mbtask.h	Set the slave address of the device
REG_INPUT_START	mbtask.h	Input registers starting counter
REG_INPUT_NREGS	mbtask.h	Number of input registers (bytes)
REG_HOLDING_START	mbtask.h	Holding registers starting counter
REG_HOLDING_NREGS	mbtask.h	Number of holding registers (bytes)
REG_COILS_START	mbtask.h	Coils starting counter
REG_COILS_SIZE	mbtask.h	Number of coils (bits)
REG_DISC_START	mbtask.h	Discrete inputs starting counter
REG_DISC_SIZE	mbtask.h	Number of discrete inputs (bits)
MB_FUNC_OTHER_REP_SLAVEID_ENABLED	mbconfig.h	Enable/disable report slave ID function
MB_FUNC_READ_INPUT_ENABLED	mbconfig.h	Enable/disable read input register function
MB_FUNC_READ_HOLDING_ENABLED	mbconfig.h	Enable/disable read holding register function
MB_FUNC_WRITE_HOLDING_ENABLED	mbconfig.h	Enable/disable write holding register function

Parameter	Location	Description
MB_FUNC_WRITE_MULTIPLE_HOLDING_ENABLED	mbconfig.h	Enable/disable write multiple holding register function
MB_FUNC_READ_COILS_ENABLED	mbconfig.h	Enable/disable read coils function
MB_FUNC_WRITE_COIL_ENABLED	mbconfig.h	Enable/disable write coils function
MB_FUNC_WRITE_MULTIPLE_COILS_ENABLED	mbconfig.h	Enable/disable write multiple coils function
MB_FUNC_READ_DISCRETE_INPUTS_ENABLED	mbconfig.h	Enable/disable read discrete inputs function
MB_FUNC_READWRITE_HOLDING_ENABLED	mbconfig.h	Enable/disable read/write holding register function

Revision history

Table 2. **Document revision history**

Date	Revision	Changes
28-Aug-2024	1	Initial release.

Contents

List of tables

List of figures