UM1911
User manual

Getting started with the X-CUBE-SPN3 high power stepper motor driver software expansion for STM32Cube

Introduction

The X-CUBE-SPN3 is an expansion software package for STM32Cube. The software runs on the STM32 and includes driver recognition for the powerSTEP01 device. The expansion is built on STM32Cube software technology to ease portability across different STM32 microcontrollers. It is compatible with the NUCLEO-F401RE, NUCLEO-F030R8, NUCLEO-F334R8 and NUCLEO-L053R8 boards connected to one, two or three X-NUCLEO-IHM03A1 STM32 expansion boards.

The software comes with a sample implementation of the drivers to control one stepper motor.

The expansion is built on STM32Cube software technology to ease portability across different STM32 microcontrollers. Information regarding STM32Cube is available on www.st.com at http://www.st.com/stm32cube.

Contents

List of tables

List of figures

1 Acronyms and abbreviations

Table 1: Acronyms and abbreviations

Acronym	Description
API	Application programming interface
BSP	Board support package
CMSIS	Cortex® microcontroller software interface standard
HAL	Hardware abstraction layer
SPI	Serial port interface
IDE	Integrated development environment
LED	Light emitting diode

2 What is STM32Cube?

STMCube™ represents the STMicroelectronics initiative to make developers' lives easier by reducing development effort, time and cost. STM32Cube covers the STM32 portfolio.

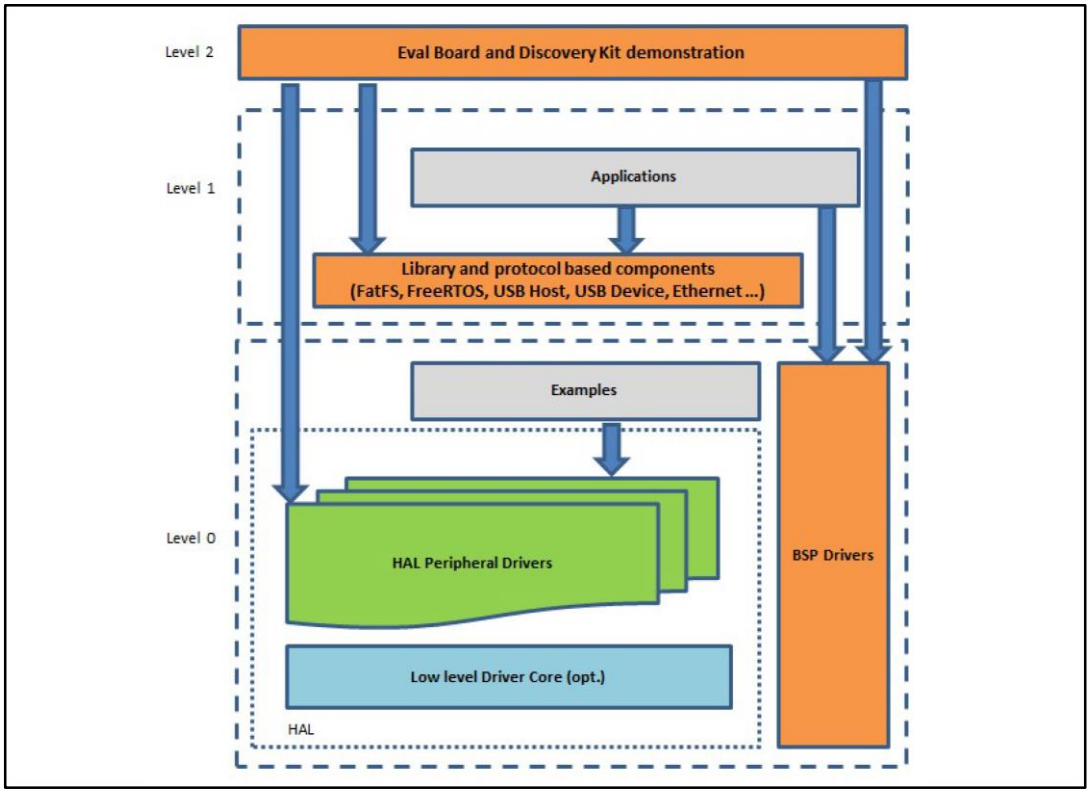
STM32Cube version 1.x includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform specific to each series (such as the STM32CubeF4 for the STM32F4 series), which includes:
  - the STM32Cube HAL embedded abstraction-layer software, ensuring maximized portability across the STM32 portfolio
  - a consistent set of middleware components such as RTOS, USB, TCP/IP and graphics
  - all embedded software utilities with a full set of examples

2.1 STM32Cube architecture

The STM32Cube firmware solution is built around three independent levels that can easily interact with one another, as described in the diagram below:

Figure 1: Firmware architecture



Level 0: This level is divided into three sub-layers:

- Board Support Package (BSP): this layer offers a set of APIs relative to the hardware components in the hardware boards (Audio codec, IO expander, Touchscreen, SRAM driver, LCD drivers. etc…) and composed of two parts:

  - Component: is the driver relative to the external device on the board and not related to the STM32, the component driver provides specific APIs to the external components of the BSP driver, and can be ported on any other board.
  - BSP driver: links the component driver to a specific board and provides a set of easy to use APIs. The API naming convention is BSP_FUNCT_Action(): e.g., BSP_LED_Init(), BSP_LED_On().

It is based on modular architecture allowing is to be easily ported on any hardware by just implementing the low level routines.

- Hardware Abstraction Layer (HAL): this layer provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides generic, multi-instance and function-oriented APIs to help offload user application development time by providing ready to use processes. For example, for the communication peripherals (I2S, UART, etc.) it provides APIs for peripheral initialization and configuration, data transfer management based on polling, interrupt or DMA processes, and communication error management. The HAL Drivers APIs are split in two categories: generic APIs providing common, generic functions to all the STM32 series and extension APIs which provide special, customized functions for a specific family or a specific part number.
- Basic peripheral usage examples: this layer houses the examples built around the STM32 peripherals using the HAL and BSP resources only.

Level 1: This level is divided into two sub-layers:

- Middleware components: set of libraries covering USB Host and Device Libraries, STemWin, FreeRTOS, FatFS, LwIP, and PolarSSL. Horizontal interaction among the components in this layer is performed directly by calling the feature APIs, while vertical interaction with low-level drivers is managed by specific callbacks and static macros implemented in the library system call interface. For example, FatFs implements the disk I/O driver to access a microSD drive or USB Mass Storage Class.
- Examples based on the middleware components: each middleware component comes with one or more examples (or applications) showing how to use it. Integration examples that use several middleware components are provided as well.

Level 2: This level is a single layer with a global, real-time and graphical demonstration based on the middleware service layer, the low level abstraction layer and basic peripheral usage applications for board-based functions.

3 X-CUBE-SPN3 software expansion for STM32Cube

3.1 Overview

The X-CUBE-SPN3 software package expands the functionality of STM32Cube.

The key features of the software package are:

- Complete middleware to build applications using the powerSTEP01 device
- Sample, single stepper motor control application for a X-NUCLEO-IHM03A1 board when plugged to NUCLEO-F401RE, NUCLEO-F030R8, NUCLEO-F334R8 or NUCLEO-L053R8 board
- Easy portability across different MCU families, thanks to STM32Cube
- Free, user-friendly license terms

The package includes:

- powerSTEP01 read/write registers to:
  - set or monitor the motor positions
  - set home position and mark another position
  - set minimum and maximum speed
  - monitor current speed
  - set acceleration and deceleration
  - set parameters for voltage mode driving
  - read the ADCIN voltage
  - set the overcurrent threshold
  - set the step mode and control method
  - enable/disable alarms
  - set current control parameters for gate driving
  - configure various operating options
  - read system status information
- Nucleo and expansion board configuration (GPIOs, PWMs, IRQs, etc.)
- Application commands to:
  - handle micro-stepping (up to 1/128)
  - handle the step-clock
  - perform various positioning, moves and stops
  - retrieve system status information
- FLAG and BUSY interrupt handling (alarm reporting)
- Daisy chain handling

When starting the powerSTEP01 driver, the user specifies the number of powerSTEP01 devices connected to the STM32 Nucleo board (i.e., the number of X-NUCLEO-IHM03A1 expansion boards).

Once set, the number of devices must not be changed.

Depending on the number of devices, the driver:

- sets up the required GPIOs to handle the motor direction and FLAG and BUSY interrupts
- starts the SPI driver to communicate with the powerSTEP01 devices

- releases the reset of each of the powerSTEP01 devices
- disables the power bridge and clears the status flags of the powerSTEP01 devices
- loads the registers of each powerSTEP01 device with the predefined values from “powerstep01_target_config.h”, in order to program speed profile boundaries, step mode, voltage or current mode, current decay settings, overcurrent protection, etc.

Once initialization is complete, the user can modify the powerSTEP01 registers as desired. Most of the functions of the driver take a device ID (from 0 to 254) as input parameter. It gives the user the possibility to specify which of the device configurations to modify.

The user can also write callback functions and attach them to:

- the flag interrupt handler depending on the actions to perform when an alarms is reported (read the flags, clear and read the flags, etc.)
- the busy interrupt handler which is called each time the busy pin position is changed
- the error handler which is called by the library when it reports an error.

Then, the user can request the movement of one or several motors (still using the principle of device ID). This request can be to:

- move for a given number of steps in a specified direction
- go to a specific position
- run until reception of a new instruction
- go at a constant speed until an external switch is closed triggering a soft stop
- go at minimum speed until an external switch is opened triggering a hard stop

The speed profile is completely handled by the powerstep01. The motor starts moving by using the programmed minimum speed (set in MIN_SPEED register). At each step, the speed is increased using the acceleration value (ACC register).

If the target position of a motion command is far enough, the motor performs a trapezoidal operation:

- accelerating phase using the device acceleration parameter
- steady phase where the motor turns at maximum speed
- decelerating phase using the device deceleration parameter
- stop at the targeted position

If the target position does not allow it to reach the max speed, the motor performs a triangular operation:

- accelerating phase using the device acceleration parameter
- decelerating phase using the device deceleration parameter
- stop at the targeted position

A motion command can be stopped at any moment:

- either by a soft stop or `softHiz`, which progressively decreases the speed using the deceleration parameter. Once the minimum speed is reached, the motor is stopped.
- or by a hard stop or `hardHiz` command which immediately stops the motor.

When the motor is stopped using the `softHiz` or `hardHiz` command, the power bridge is automatically disabled.

The driver has a `Powerstep01_WaitWhileActive()` command which locks program execution until the motor stops moving, to avoid sending a new command to a device before the completion of the previous one.

The library also offers the possibility to change the step mode (from full step until 1/128 micro-step mode) for a given device. When the step mode is changed, the current position (ABS_POSITION register) is automatically reset.

To limit the memory usage of the library, the maximum number of devices in the daisy chain is limited to 3 in powerstep01_target_config.h via the `MAX_NUMBER_OF_DEVICES` definition. This constant can be increased up to 255.

3.2 Architecture

This software is an STM32Cube expansion which fully complies with STM32Cube architecture, enabling the development of applications using stepper motor drivers.

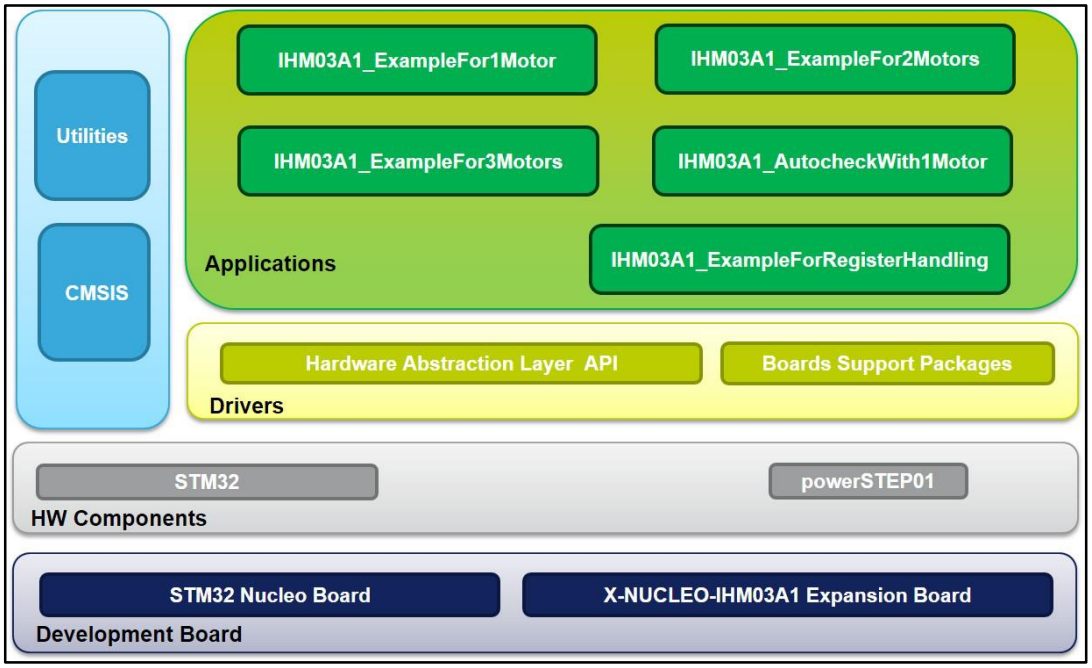
The software is based on the STM32CubeHAL hardware abstraction layer for the STM32 microcontroller. The package extends STM32Cube by providing a board support package (BSP) for the motor control expansion board and a BSP component driver for the powerSTEP01 device.

The software layers used by the application software to access and use the stepper motor driver expansion board are:

- STM32Cube HAL Layer: provides a generic multi-instance simple set of APIs (application programming interfaces) to interact with the upper layers (application, libraries and stacks). It is composed of generic and extension APIs. It is directly built around a generic architecture and allows the layers built upon it, such as the middleware layer, to implement their functions without necessitating specific hardware configurations for a given Microcontroller Unit (MCU). This structure improves library code reusability and guarantees easy portability across other devices.
- Board support package (BSP) Layer: the software provides support for the peripherals (apart from the MCU) on the STM32 Nucleo board in the board support package (BSP). It is a limited set of APIs which provides a programming interface for certain board specific peripherals such as the LED, the user button, etc. This interface also helps in identifying the specific board version. For motor control expansion boards, the motor control BSP provides the programming interface for various motor driver components. In the X-CUBE-SPN3 software, it is associated with the BSP component for the powerSTEP01 motor driver.

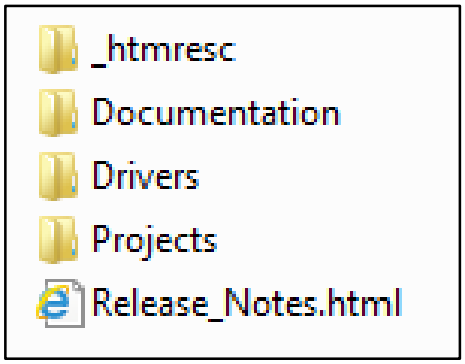
The following diagram outlines the software architecture of the package.

Figure 2: Software architecture



3.3 Folder structure

Figure 3: Folder structure



The software package code is located in two main folders:

- A **Drivers** folder with:
  - the required STM32Cube HAL files, located in the STM32L0xx_HAL_Driver, STM32F0xx_HAL_Driver, STM32F3xx_HAL_Driver or STM32F4xx_HAL_Driver subfolders. These files derive directly from the STM32Cube framework.
  - a CMSIS folder with the CMSIS (Cortex® microcontroller software interface standard) files from ARM. These files form the vendor-independent hardware abstraction layer for the Cortex-M processor series.
  - a BSP (board support package) folder with X-NUCLEO-IHM03A1 configuration files, the powerSTEP01 driver and the motor control API (see *BSP folder*).
- A **Project** folder with several use examples of the powerSTEP01 motor driver for different Nucleo platforms (see *Projects folder*).

3.3.1 BSP folder

3.3.1.1 STM32L0XX-Nucleo/STM32F0XX-Nucleo/ STM32F3XX-Nucleo/STM32F4XX-Nucleo BSPs

The BSPs provide an interface to configure and use the Nucleo peripherals with the expansion board X-NUCLEO-IHM03A1, for each compatible STM32 Nucleo board. In each respective STM32L0XX-Nucleo, STM32F0XX-Nucleo, STM32F3XX-Nucleo and STM32F4XX-Nucleo subdirectory, there are two ".c" and ".h" file pairs:

- **stm32XXxx_nucleo.c/h**: derive from the STM32Cube framework without modification and provide the functions to manipulate the specific STM32 Nucleo board user button and LEDs.
- **stm32XXxx_nucleo_ihm03a1.c/h**: for the configuration of the SPI, the PWMs, the GPIOs and the interrupt enabling/disabling required for X-NUCLEO-IHM03A1 expansion board operation.

3.3.1.2 Motor control BSP

This BSP provides a common interface to access the driver functions of motor drivers such as L6474, Powerstep01, etc., via the MotorControl/motorcontrol.c/h files. These files define all the functions to configure and control the motor driver. These functions are then mapped to the functions of the motor driver component which is used on the given expansion board via the motorDrv_t (defined in Components\Common\motor.h.) structure file, which defines a list of function pointers which are filled during its instantiation in the corresponding motor driver component.

For the X-CUBE-SPN3, the instance of the structure is called powerstep01Drv (see the BSP\Components\powerstep01\powerstep01.c file).

As the motor control BSP is common for all ST motor driver expansion boards, some functions may not be available on a given expansion board. In this case, during the instantiation of the motorDrv_t structure in the driver component, the unavailable functions are replaced with null pointers.

3.3.1.3 powerSTEP01 BSP component

The powerSTEP01 BSP component provides the driver functions of the powerSTEP01 motor driver in the folder stm32_cube\Drivers\BSP\Components\powerSTEP01.

This folder has the follwoing files:

- **powerstep01.c**: powerSTEP01 driver core functions
- **powerstep01.h**: powerSTEP01 driver function dedclarations and their associated definitions
- **powerstep01_target_config.h**: predefined values for the powerSTEP01 registers and for the motor device contexts

3.3.2 Projects folder

For each Nucleo platform, the following sample projects are available in stm32_cube\Projects\Multi\Examples\MotionControl\:

- IHM03A1_ExampleFor1Motor: examples of control functions with 1 motor
- IHM03A1_ExampleFor2Motors: examples of control functions with 2 motors
- IHM03A1_ExampleFor3Motors: examples of control functions with 3 motors
- IHM03A1_ExampleForRegisterHandling: examples of powerSTEP01 register handling with 1 device

- IHM03A1_AutocheckWith1Motor: example where the motor advances eight steps when the user button is pushed.

Each example has a folder dedicated to the targeted IDE:

- EWARM where are located the project files for IAR
- MDK-ARM where are located the project files for Keil
- SW4STM32 where are located the project files for OpenSTM32

Each example also has the following code files:

- inc\main.h: Main header file
- inc\ stm32xxxx_hal_conf.h: HAL configuration file
- inc\stm32xxxx_it.h: header for the interrupt handler
- src\main.c: main program (code of the example which is based on the motor control library for powerSTEP01)
- src\stm32xxxx_hal_msp.c: HAL initialization routines
- src\stm32xxxx_it.c: interrupt handler
- src\system_stm32xxxx.c: system initialization
- src\clock_xx.c: clock initialization

3.4 Required software

Communication between the powerSTEP01 and the MCU is mainly handled through the SPI interface. This requires the use of 4 GPIOs: CS, MOSI, MISO, SCK (SPI Clock). By default, the “stm32f4xx_nucleo_ihm03a1.h” of the STM32 Nucleo BSP uses the SPI1, but you can use SPI2 by declaring the `BSP_MOTOR_CONTROL_BOARD_USE_SPI2` preprocessor option.

You do not need to supply a step clock to a powerSTEP01 device as the speed profile generation and the positioning calculations are digitally controlled through SPI programming. Through SPI and daisy chaining, several expansion boards can be programmed with completely independent directions, speeds, and target positions.

Motion generation using an external step clock is still possible by configuring the STEP_MODE register; a GPIO is required for this task.

all expansion boards share the same GPIO for the step clock generation.

For flag interrupt handling, the X-CUBE-SPN3 software uses only one external interrupt for all devices, as all flag pins are connected together. The same applies for the busy interrupt.

The direction is also handled through SPI programming.

Finally, one common GPIO is used for all the powerSTEP01 device resets.

Table 2: Required resources for the X-CUBE-SPN3 software

Resources F4xx, F3xx and F0xx	Resources L0xx	Digital pin	Features	Device
Ext Line 10 GPIO PA10		2	Flag interrupt	All
Ext Line 5 GPIO PB5		5	Busy interrupt	All
GPIO PA9		8	Reset	All
Timer 3 Ch2 GPIO PC7	Timer 3 Ch2 GPIO PC7	9	Step clock	All
CS GPIO PB6		10	SPI chip select	All
MOSI GPIO PA7 for SPI1 GPIO PB15 for SPI2		11	SPI master OUT Slave IN	All
MOSI GPIO PA6 for SPI1 GPIO PB14 for SPI2		12	SPI master IN Slave OUT	0
SCK GPIO PA5 for SPI1 GPIO PB14 for SPI2		13	SPI serial clock	0

3.5 APIs

The API of the X-CUBE-SPN3 software is defined in the motor control BSP. Its functions have the `BSP_MotorControl_` prefix.

not all the functions of this module are available for the powerSTEP01 and consequently for the X-NUCLEO-IHM03A1 expansion board.

Detailed technical information regarding the APIs available to the user can be found in a compiled HTML file located in the package Documentation folder, with full function and parameter descriptions.

3.6 Sample application description

Several sample applications using the X-NUCLEO-IHM03A1 expansion board with either the NUCLEO-F401RE, NUCLEO-F030R8, NUCLEO-F334R8 or NUCLEO-L053R8 board are provided in the Projects directory. Ready-to-build projects are available for multiple IDEs (see *Projects folder*).

4 System setup guide

4.1 Hardware description

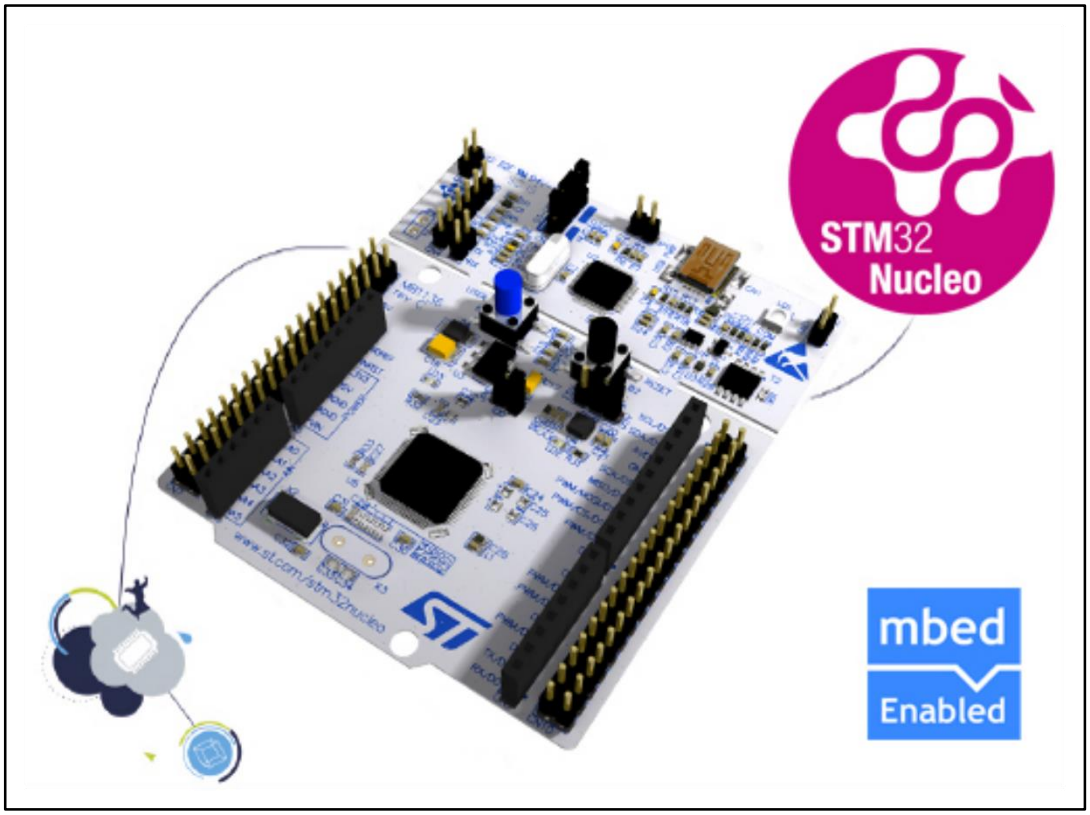
This section describes the hardware components needed to develop a stepper motor driver-based application.

4.1.1 STM32 Nucleo platform

The STM32 Nucleo boards provide an affordable and flexible way for users to try out new ideas and build prototypes with any STM32 microcontroller lines. The Arduino™ connectivity support and ST morpho headers make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide range of specialized expansion boards to choose from. The STM32 Nucleo board does not require any separate probe as it integrates the ST-LINK/V2-1 debugger/programmer. The STM32 Nucleo board comes with the comprehensive STM32 software HAL library together with various packaged software examples.

Information regarding the STM32 Nucleo board is available on www.st.com at http://www.st.com/stm32nucleo

Figure 4: STM32 Nucleo board

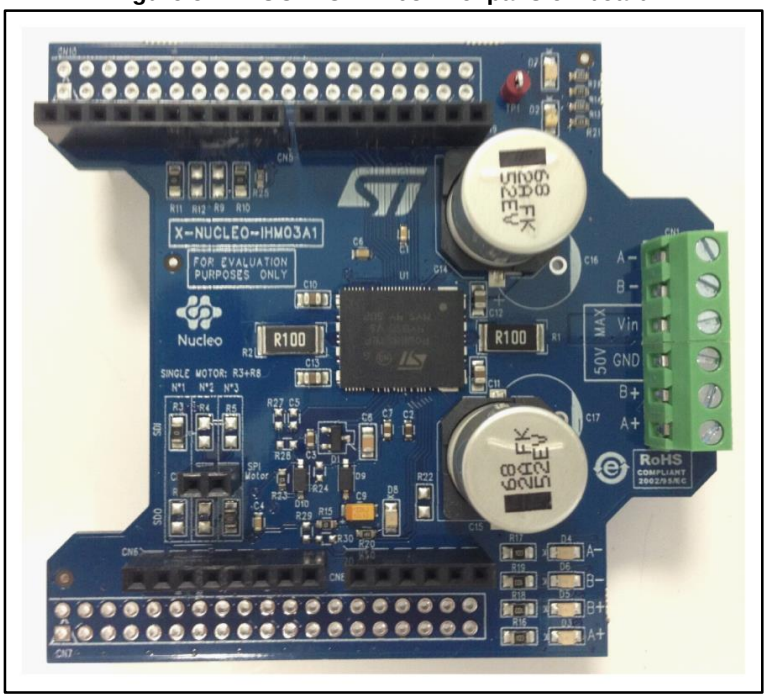


4.1.2 X-NUCLEO-IHM03A1 expansion board

The X-NUCLEO-IHM03A1 is a power microstepper motor driver expansion board based on the powerSTEP01 device. It provides an affordable and easy-to-use solution for driving a stepper motor in your STM32 Nucleo project.

The X-NUCLEO-IHM03A1 board is compatible with the Arduino UNO R3 connector, and supports the addition of other boards which can be stacked to drive up to three stepper motors with a single STM32 Nucleo board.

Figure 5: X-NUCLEO-IHM03A1 expansion board



Information regarding the X-NUCLEO-IHM03A1 expansion board is available at: http://www.st.com/x-nucleo

4.1.3 Further hardware components

To complete the hardware setup, you will need:

- 1 to 3 stepper motors
- an external 10.5 - 50 V DC power supply with 2 electrical cables for the X-NUCLEO-IHM03A1 board
- a USB cable type A to mini-B to connect the STM32 Nucleo to a PC

4.2 Software description

The following software components are needed in order to set up a suitable development environment for creating applications based on the motor driver expansion board:

- X-CUBE-SPN3, the STM32 expansion for powerSTEP01 motor driver application development. The X-CUBE-SPN3 firmware and related documentation are available on www.st.com.
- A development tool-chain and compiler, chosen from one of the following environments:
  - Keil RealView Microcontroller Development Kit (MDK-ARM) toolchain V5.12
  - IAR Embedded Workbench for ARM (EWARM) toolchain V7.20
  - OpenSTM32 System Workbench for STM32 (SW4STM32)

4.3 Hardware and software setup

This section describes the hardware and software setup procedure to run the sample applications and develop new applications based on the motor driver expansion board.

4.3.1 Common setup to drive 1, 2 or 3 motors

The STM32 Nucleo must be configured with the following jumper positions:

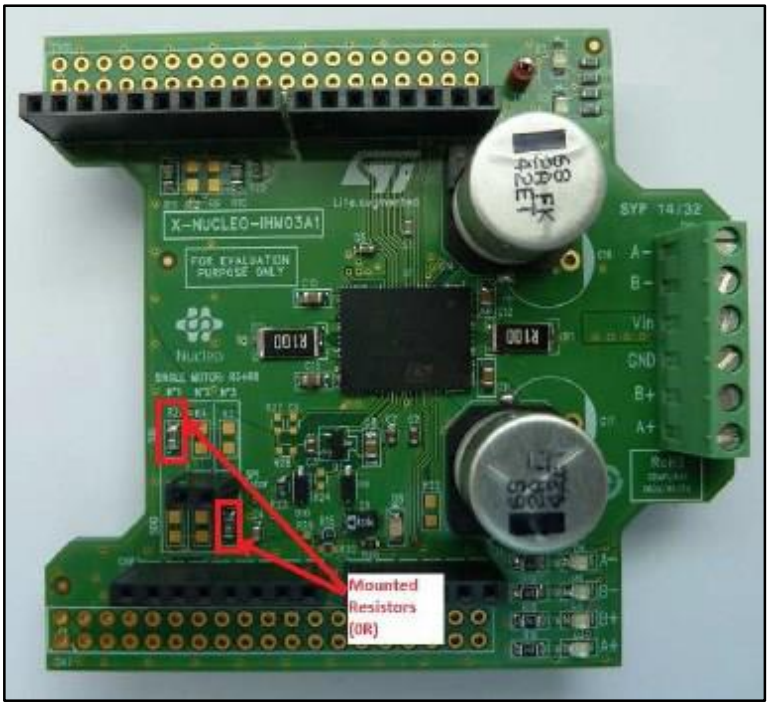
- JP1 off
- JP5 (PWR) on UV5 side
- JP6 (IDD) on

4.3.2 Setup to drive 1 motor

The X-NUCLEO-IHM03A1 expansion board must have:

- Resistors (0R) R3 and R8 mounted
- Resistors R4, R5, R6, and R7 not mounted

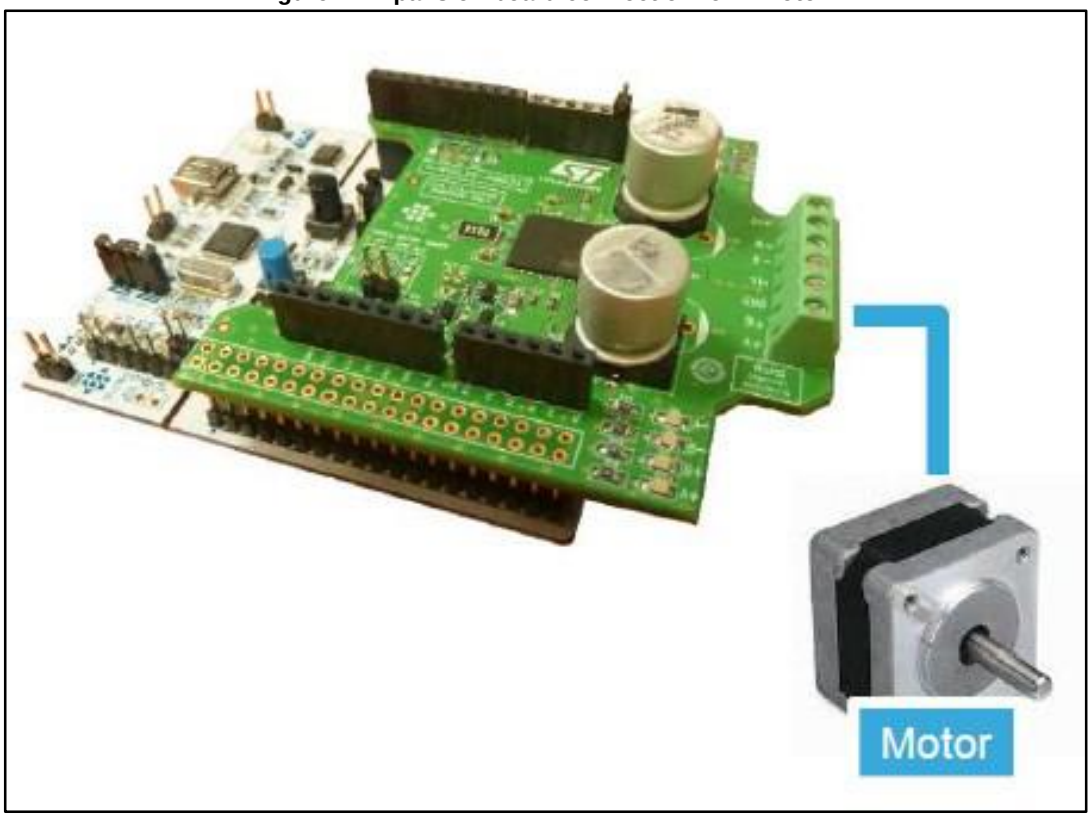
Figure 6: X-NUCLEO-IHM03A1 configuration for 1 motor



Once the boards are properly configured:

- Plug the X-NUCLEO-IHM03A1 expansion board on top of the STM32 Nucleo via the Arduino UNO connectors
- Connect the STM32 Nucleo board to a PC with the USB cable through USB connector CN1 to power the board
- Power the X-NUCLEO-IHM03A1 expansion board on by connecting its Vin and Gnd connectors to the DC power supply. The DC supply must be set to deliver the required voltage for the stepper motor.
- Connect the stepper motor to the X-NUCLEO-IHM03A1 bridge connectors A± and B±

Figure 7: Expansion board connection for 1 motor



Once the system is set up:

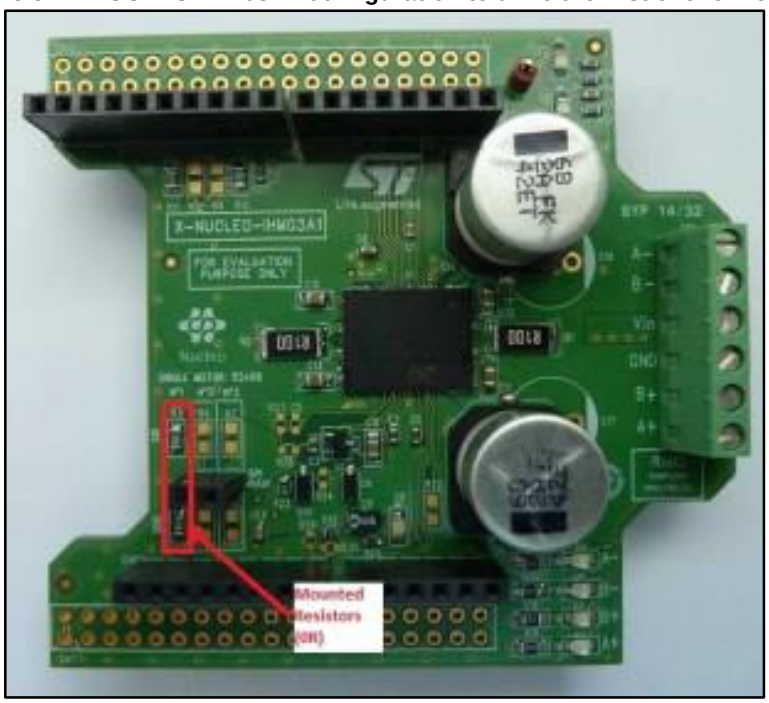
- Open your preferred toolchain (MDK-ARM from Keil, EWARM from IAR, or SW4STM32 from OpenSTM32)
- Depending on the STM32 Nucleo board used, open the software project from:
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor1Motor\YourToolChainName\STM32F401RE-Nucleo for Nucleo STM32F401
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor1Motor\YourToolChainName\STM32F030R8-Nucleo for Nucleo STM32F030
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor1Motor\YourToolChainName\STM32F334R8-Nucleo for Nucleo STM32F334
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor1Motor\YourToolChainName\STM32L053R8-Nucleo for Nucleo STM32L053
- To adapt the powerSTEP01 driver to your stepper motor, open stm32_cube\Drivers\BSP\Components\powerstep01\powerstep01_target_config.h and modify the parameters ending in `_DEVICE_0`.
- Rebuild all the files and load your image into target memory
- Run the sample application. The motor automatically starts (see main.c for the demo sequence details).

4.3.3 Setup to drive 2 motors

For the first motor, The X-NUCLEO-IHM03A1 expansion board must have:

- Resistors (0R) R3 and R6 mounted.
- Resistors R4, R5, R7 and R8 not mounted

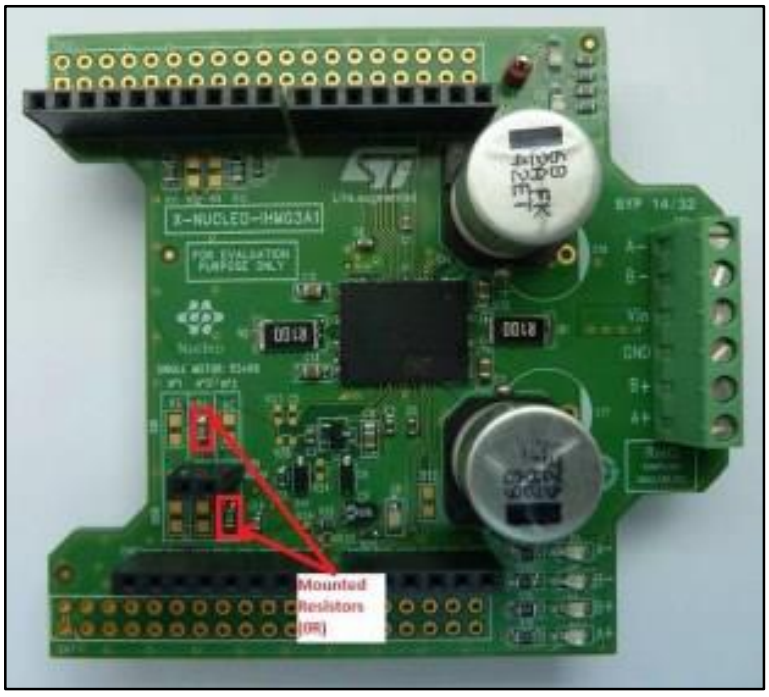
Figure 8: X-NUCLEO-IHM03A1 configuration to drive the first of two motors



For the second motor, the X-NUCLEO-IHM03A1 expansion board for second motor must have:

- Resistors (0R) R4 and R8 mounted
- Resistors R3, R5, R6 and R7 not mounted

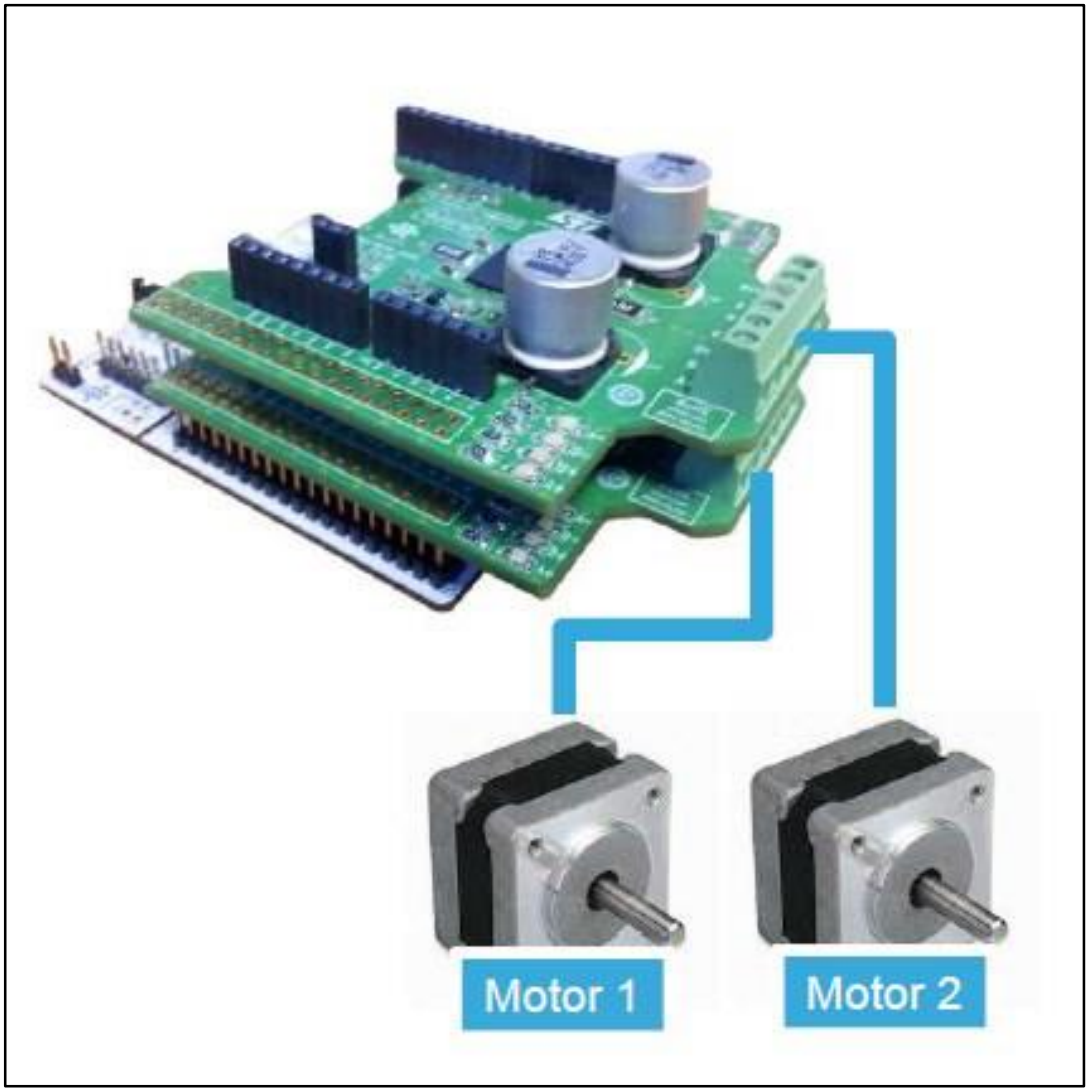
Figure 9: X-NUCLEO-IHM03A1 configuration to drive the second of two motors



Once the boards are properly configured:

- Plug the X-NUCLEO-IHM03A1 for the first motor on top of the STM32 Nucleo via the Arduino UNO connectors
- Plug the X-NUCLEO-IHM03A1 for the second motor on top of the one for the first motor
- Connect the STM32 Nucleo board to a PC with the USB cable through USB connector CN1 to power the board
- Power on the X-NUCLEO-IHM03A1 expansion boards by connecting their Vin and Gnd connectors to the DC power supply. The DC supply must be set to deliver the required voltage for the stepper motors.
- Connect each stepper motor to the bridge connectors A± and B± of their corresponding X-NUCLEO-IHM03A1 board

Figure 10: Expansion board connection for 2 motors



Once the system is set up:

- Open your preferred toolchain (MDK-ARM from Keil, EWARM from IAR, or SW4STM32 from OpenSTM32)
- Depending on the STM32 Nucleo board used, open the software project from:
  - stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor2Motors\ YourToolChainName\STM32F401RE-Nucleo for Nucleo STM32F401

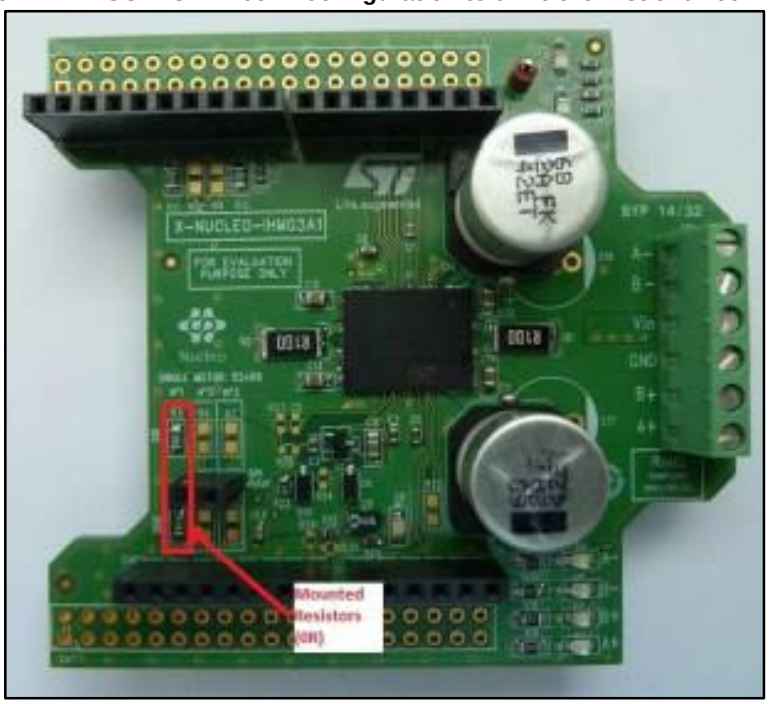
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor2Motors\ YourToolChainName\STM32F030R8-Nucleo for Nucleo STM32F030
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor1Motor\Y
  - ourToolChainName\STM32F334R8-Nucleo for Nucleo STM32F334
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor2Motors\ YourToolChainName\STM32L053R8-Nucleo for Nucleo STM32L053
- To adapt the powerSTEP01 driver to your stepper motor, open stm32_cube\Drivers\BSP\Components\powerstep01\powerstep01_target_config.h and modify the parameters ending in `_DEVICE_0` for the first motor, and `_DEVICE_1` for the second motor
- Rebuild all files and load your image into target memory
- Run the sample application. The motors start automatically (see main.c for the demo sequence details)

4.3.4 Setup to drive 3 motors

For the first motor, The X-NUCLEO-IHM03A1 expansion board must have:

- Resistors (0R) R3 and R6 mounted.
- Resistors R4, R5, R7 and R8 not mounted

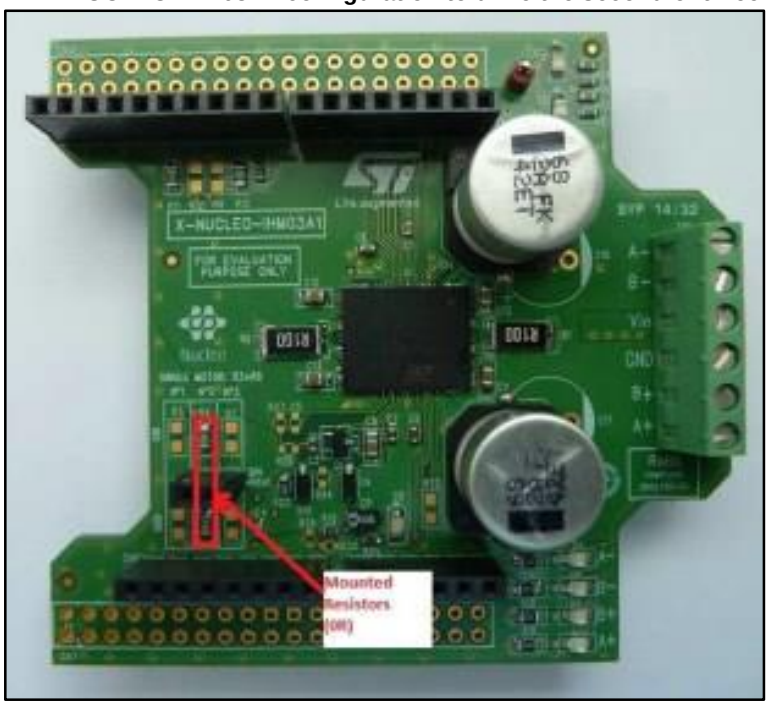
Figure 11: X-NUCLEO-IHM03A1 configuration to drive the first of three motors



For the second motor, the X-NUCLEO-IHM03A1 expansion board must have:

- Resistors (0R) R4 and R7 mounted
- Resistors R3, R5, R6 and R8 not mounted

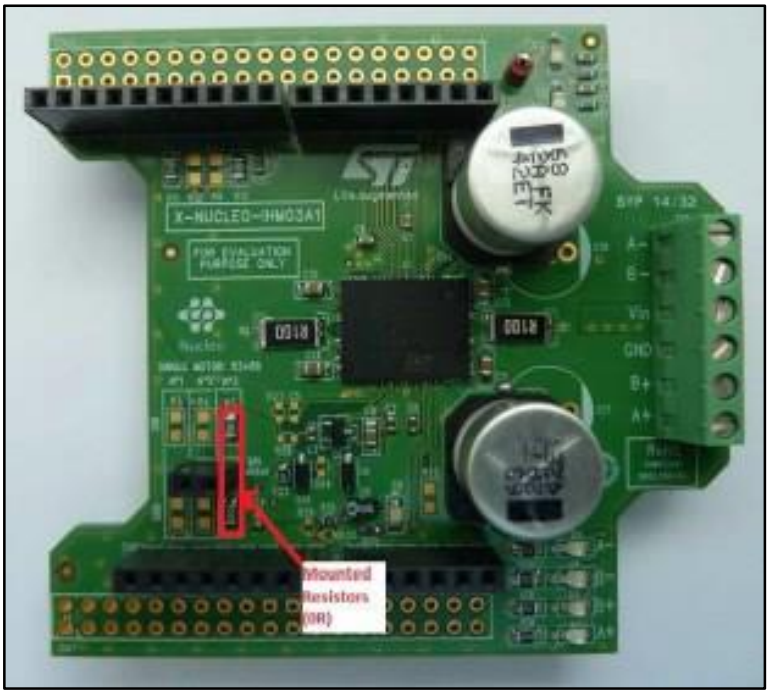
Figure 12: X-NUCLEO-IHM03A1 configuration to drive the second of three motors



For the third motor, the X-NUCLEO-IHM03A1 expansion board must have:

- Resistors (0R) R5 and R8 mounted
- Resistors R3, R4, R6 and R7 not mounted

Figure 13: X-NUCLEO-IHM03A1 configuration to drive the third of three motors

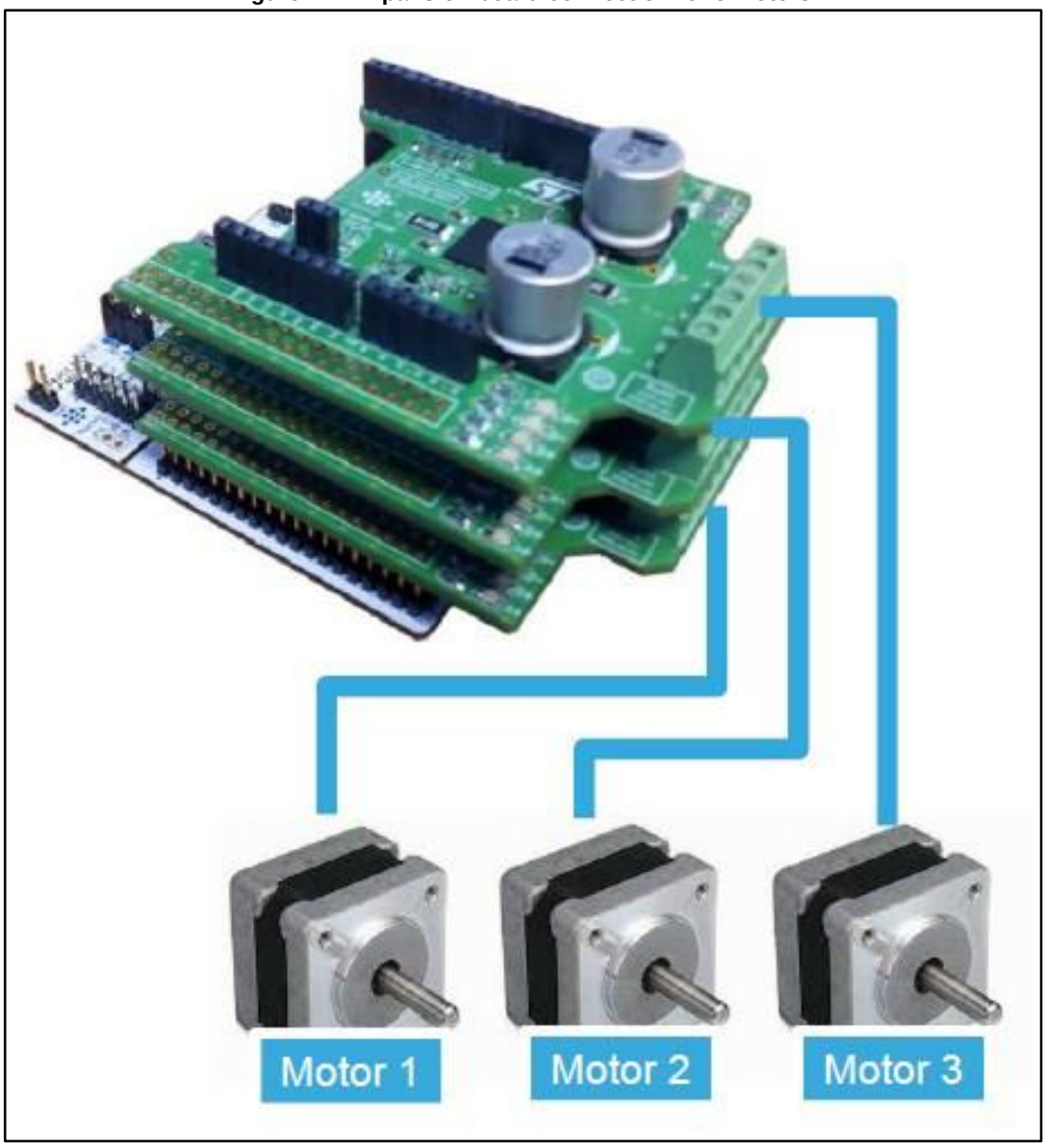


Once the boards are properly configured:

- Plug the X-NUCLEO-IHM03A1 for first motor on top of the STM32 Nucleo by using the Arduino UNO connectors

- Plug the X-NUCLEO-IHM03A1 for the second motor on top of the one for first motor
- Plug the X-NUCLEO-IHM03A1 for the third motor on top of the second one
- Connect the STM32 Nucleo board to a PC with the USB cable through USB connector CN1 to power the board
- Power on the X-NUCLEO-IHM03A1 expansion boards by connecting their connectors Vin and Gnd to the DC power supply. The DC supply must be set to deliver the required voltage for the stepper motors.
- Connect each stepper motor to the bridge connectors A± and B± of their dedicated X-NUCLEO-IHM03A1 board.

Figure 14: Expansion board connection for 3 motors



Once the system is set up:

- Open your preferred toolchain (MDK-ARM from Keil, EWARM from IAR, or SW4STM32 from OpenSTM32)
- Depending on the STM32 Nucleo board used, open the software project from:

  - stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor3Motors\ YourToolChainName\STM32F401RE-Nucleo for Nucleo STM32F401
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor3Motors\ YourToolChainName\STM32F030R8-Nucleo for Nucleo STM32F030
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor1Motor\Y ourToolChainName\STM32F334R8-Nucleo for Nucleo STM32F334
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor3Motors\ YourToolChainName\STM32L053R8-Nucleo for Nucleo STM32L053
- To adapt the powerSTEP01 driver to your stepper motor, open stm32_cube\Drivers\BSP\Components\powerstep01\powerstep01_target_config.h and modify the parameters ending in `_DEVICE_0` for the first motor, `_DEVICE_1` for the second motor and `_DEVICE_2` for the third motor
- Rebuild all files and load your image into target memory
- Run the example. The motors start automatically (see main.c for demo sequence details).

5 Revision history

Table 3: Document revision history

Date	Version	Changes
22-Jul-2016	1	Initial release.
01-Mar-2016	2	Throughout document: - text and formatting changes - removed all specific API descriptions (now available in a compiled HTML file in the package Documentation folder). - added STM32F3xx compatibility information - replaced references to "TrueStudio" with "System Workbench for STM32 (SW4STM32)"

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.