UM1927
User manual

Getting started with the X-CUBE-SPN5 bipolar stepper motor driver software expansion for STM32Cube

Introduction

The X-CUBE-SPN5 expansion package for STM32Cube gives you full control of stepper motor operation. With this software running on the STM32 microcontroller of an NUCLEO-F401RE, NUCLEO-F030R8, NUCLEO-F334R8 or NUCLEO-L053R8 board, you can build and test your applications with ST's fully integrated L6208 stepper motor driver on an X-NUCLEO-IHM05A1 board.

The STM32Cube platform allows flexible solution design and integration in a persistent environment. Build your own ideas from scratch or begin experimenting immediately with the included sample setup to control a single bipolar stepper motor.

Information regarding STM32Cube is available on www.st.com at http://www.st.com/stm32cube.

Contents

List of tables

List of figures

1 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.

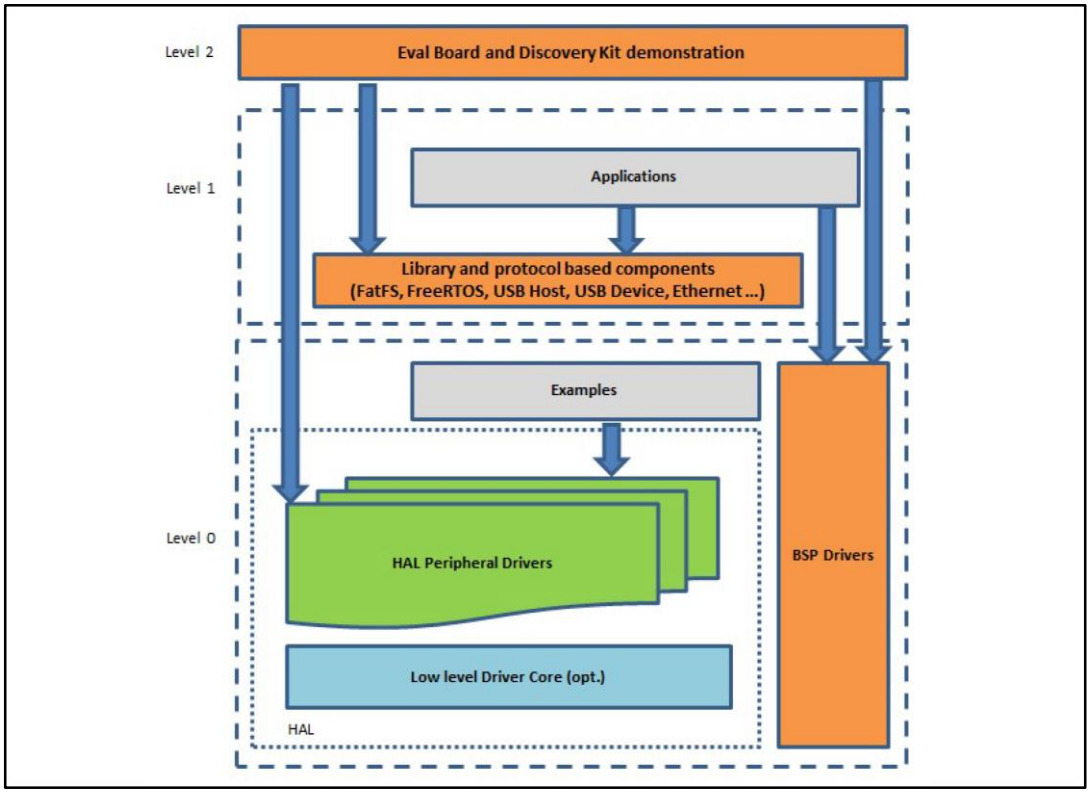
STM32Cube version 1.x includes:

- 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

1.1 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.

2 X-CUBE-SPN5 software expansion for STM32Cube

2.1 Overview

This 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 STM32 expansion board based on the L6208.

It allows complete management of the L6208 through a comprehensive set of APIs, and gives you access to the following features:

•

The software implements pseudo registers and motion commands:

- By configuring PWMs to generate step clocks and voltage references for micro-stepping
- By managing acceleration, deceleration, min and max speed, positions at speed profile boundaries, mark position, micro-stepping samples, micro-stepping mode, direction, current decay mode, velocity or position mode, motion state, etc. in the form of device parameters

The software implements a tick timer. At the end of each tick timer pulse, a callback is executed to call the tick handler which controls motor motion via:

- The step clock signal falling and rising edges
- In full step or half step mode, a rising edge command is generated each time the position computation equals a new full or half step (relevant bit toggling is checked)
- In micro-stepping mode, a rising edge command is generated each time a quarter period of the reference sine-wave voltages has elapsed (relevant bit toggling is checked)
- The micro-stepping sample generation by setting the duty cycles on the PWMs for the two bridges
- The motor direction by a GPIO level
- The motor position which is varied by the speed (steps per tick)
- The speed which is varied by acceleration or deceleration (steps per tick^2)
- the motion state in positioning or velocity modes (e.g., the motor is stopped when it reaches target position).

The speed is a linear function of the step clock frequency.

The software is currently able to divide a single step into up to 16 micro-steps; this is obtained through a lookup table.

To use the L6208 driver library, you must first call the initialization function to:

- Set up the required GPIOs to handle the bridge enable pin EN, the CONTROL pin for current decay mode selection, the half/full pin for stepping or micro-stepping mode selection, the CW/CCW pin for motor direction selection, the CLOCK pin for step clock management, the RESET pin, the Flag interrupt which reports overcurrent detection or thermal protection, and the PWMs for VREFA and VREFB pin reference voltage generation
- Set up the tick timer
- Load the driver parameters with predefined values from "l6208_target_config.h" (acceleration, deceleration, minimum and maximum speed, step mode, decay mode, stop mode, PWM frequency, etc.)

Following initialization, you can modify the driver parameters by calling specific functions. The user can also write callback functions and attach them to:

- The Flag interrupt handler depending on the actions to be performed when an overcurrent or thermal alarm is signaled
- The Error handler called by the library when it reports an error

You can then proceed to invoke motor control commands:

- `BSP_MotorControl_Move` to move for a given number of steps in a specific direction
- `BSP_MotorControl_GoTo`, `BSP_MotorControl_GoHome`, and `BSP_MotorControl_GoMark` to take the shortest direction to a specific position
- `BSP_MotorControl_CmdGoToDir` to move in a set direction to a specific position
- `BSP_MotorControl_Run` to run until a new instruction is received

The speed profile is completely handled by the microcontroller. The motor starts moving at the set minimum speed (`BSP_MotorControl_SetMinSpeed`). At each step, the motor is accelerated or decelerated according to relevant settings (`BSP_MotorControl_SetAcceleration` and `BSP_MotorControl_SetDeceleration`).

The velocity versus time function is trapezoidal if the target position is far enough for the motor to reach and maintain maximum speed:

- Acceleration phase governed by `BSP_MotorControl_SetAcceleration`
- Steady phase where the motor turns at maximum speed (`BSP_MotorControl_SetMaxSpeed`).
- Deceleration phase governed by (`BSP_MotorControl_SetDeceleration`)
- Stop at target position

The velocity versus time function is triangular if the target position is not far enough to reach maximum speed::

- Acceleration phase governed by `BSP_MotorControl_SetAcceleration`
- Deceleration phase governed by (`BSP_MotorControl_SetDeceleration`)
- Stop at target position

A motion command can be stopped at any moment with a progressive 'soft stop' (`BSP_MotorControl_SoftStop`) involving deceleration, or an immediate 'hard stop' (BSP_MotorControl_HardStop). When the motor is stopped, the power bridge is automatically disabled if the stop mode parameter (`BSP_MotorControl_SetStopMode`) is set to `HIZ_MODE`.

Direction, speed, acceleration and deceleration can be changed either when the motor is stopped or when motion is requested via the `BSP_MotorControl_Run` function.

To ensure new commands do not interrupt current instructions, the `BSP_MotorControl_WaitWhileActive` command locks program execution until the motor stops moving.

The library also lets you change the step mode from full step to 1/16th step mode with `BSP_MotorControl_SelectStepMode`). When the step mode is changed, the device and the current position and speed are automatically reset.

For more information regarding APIs, see *Section 2.5: "APIs"*.

2.2 Architecture

This software is an expansion for STM32Cube, and as such it fully complies with the architecture of STM32Cube and expands it in order to enable development of applications using stepper motor drivers. Please see the previous chapter for an introduction to the STM32Cube architecture.

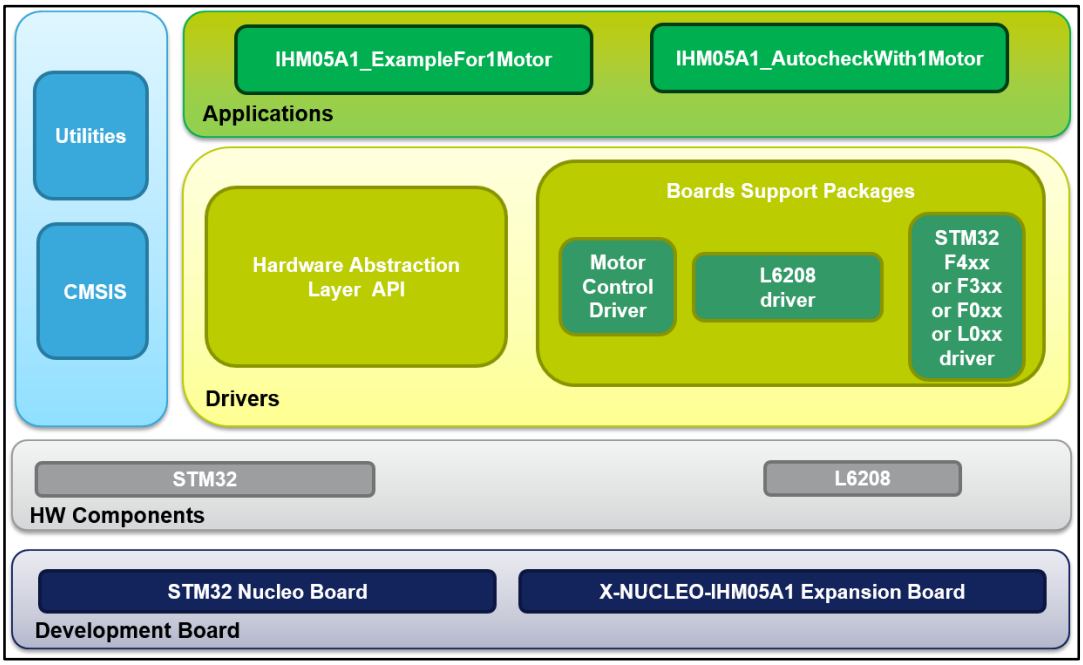
The software is based on the STM32CubeHAL, the hardware abstraction layer for the STM32 microcontroller. The package extends STM32Cube by providing a board support package (BSP) for the motor control expansion board and a BSP component driver for the L6208 motor driver.

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

- STM32Cube HAL layer: The HAL driver 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 that are built upon, such as the middleware layer, to implement their functionalities without dependency on the specific hardware configuration for a given microcontroller unit (MCU). This structure improves library code reusability and guarantees easy portability to other devices.
- Board support package (BSP) layer: The software package needs to support the peripherals on the STM32 Nucleo board apart from the MCU. This software is included in the board support package (BSP). This is a limited set of APIs which provides a programming interface for certain board specific peripherals, e.g. the LED, user button, etc. The interface also helps in identifying the specific board version. If using motor control expansion boards, the motor control BSP provides the programming interface for various motor driver components. In the X-CUBE-SPN5 software, it is associated with the BSP component for the L6208 motor driver.

The following diagram outlines the software architecture of the package:

Figure 2: Overall software architecture



2.3 Folders structure

The software code is located in two main folders:

- A **Drivers** folder with:
  - **STM32Cube HAL files**: in the STM32L0xx_HAL_Driver, STM32F0xx_HAL_Driver, STM32F3xx_HAL_Driver and STM32F4xx_HAL_Driver subfolders. These files represent the STM32Cube framework subset required to run the sample motor driver applications.
  - **CMSIS folder**: containing the CMSIS (Cortex® microcontroller software interface standard) files from ARM. These files form a vendor-independent hardware abstraction layer for the Cortex-M processor series. This folder is also unchanged from the STM32Cube framework.
  - **BSP (board support package) folder**: with the code required for the X-NUCLEO-IHM05A1 configuration and the L6208 driver and the motor control API. see *Section 2.3.1: "BSP folder"*.
- A **Project** folder with several use examples of the L6208 motor driver for different Nucleo platforms.

2.3.1 BSP folder

The following board support packages are included in the X-CUBE-SPN5 software.

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

These BSPs provide an interface to configure and use the Nucleo peripherals with the X-NUCLEO-IHM05A1 expansion board. In each compatible STM32 Nucleo board subfolder, there are two .c/.h file pairs:

- **stm32XXxx_nucleo.c/h**: these files come from the STM32Cube framework without modification and provide the functions to handle the user button and the LEDs of the corresponding Nucleo board.
- **stm32XXxx_nucleo_ihm05a1.c/h**: these files are dedicated to the configuration of the PWMs, the GPIOs and interrupt enabling/disabling for the proper operation of the X_NUCLEO_IHM05A1 expansion board.

2.3.1.2 Motor control BSP

This BSP provides a common interface to access the driver functions of various motor drivers such as the L6474, L6206 and L6208, via the MotorControl/motorcontrol.c/h file pair. These files define all the motor driver configuration and control functions, which are then mapped to the functions of the motor driver component used on the given expansion board in the `motorDrv_t` (defined in Components\Common\motor.h.) structure. This structure defines a list of function pointers which are filled during its instantiation in the corresponding motor driver component.

For the X-CUBE-SPN5, the structure is called `l6208Drv` (refer to the BSP\Components\l6208\l6208.c file).

As the motor control BSP is common for all motor driver expansion boards, not all of its functions are available for a given expansion board. During the instantiation of the `motorDrv_t` structure in the driver component, unavailable functions are replaced by a null pointer.

2.3.1.3 L6208 BSP component

The L6208 BSP component provides the driver functions of the L6208 motor driver in the stm32_cube\Drivers\BSP\Components\L6208 folder. This folder has the following files:

- l6208.c: core functions of the L6208 driver
- l6208.h: declaration of the L6208 driver functions and their associated definitions
- l6208_target_config.h: predefined values for the L6208 parameters and for the motor device context

2.3.2 Projects folder

For each Nucleo platform, one sample project is available in stm32_cube\Projects\Multi\Examples\MotionControl\:

- IHM05A1_ExampleFor1Motor: example of control functions in a single motor configuration

Each example has a target IDE folder:

- EWARM with the project files for IAR
- MDK-ARM with the project files for Keil
- SW4STM32 with 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 L6208)
- 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

2.4 Software required resources

Microcontroller management of the L6208 device and communication between the two is accomplished through GPIOs. This requires the use of six GPIOs for the EN, CONTROL, HALF/FULL, CLOCK, RESET, CW/CCW pins and two PWMs for the VREFA and VREFB pins.

To handle the overcurrent and the overtemperature alarms, the X-CUBE-SPN5 software uses an external interrupt, configured on the GPIO used for the EN pin after it has been used to either enable or disable the power bridges.

Table 1: Required resources for the X-CUBE-SPN5 software

Resources F4xx	Resources F3xx	Resources F0xx	Resources L0xx	Pin	Features
		Port A GPIO 10 External line 10		D2	EN and Flag interrupt (OCD, OVT)
Port B GPIO 3 Timer2 Ch2	Port B GPIO 3 Timer2 Ch2		Port B GPIO 3 Timer2 Ch2	D3	VREFA
		Port B GPIO 5		D4	HALF/FULL
		Port B GPIO 4		D5	CONTROL

Resources F4xx	Resources F3xx	Resources F0xx	Resources L0xx	Pin	Features
		Port B GPIO 10		D6	CLOCK
		Port A GPIO 8		D7	CW/CCW (DIR)
		Port A GPIO 9		D8	RESET
Port C GPIO 7 Timer3 Ch2	Port C GPIO 7 Timer3 Ch2	Port C GPIO 7 Timer3 Ch2	Port C GPIO 7 Timer22 Ch2	D9	VREFB
		Port A GPIO 7 Timer14 Ch2		D11	VREFA

2.5 APIs

The X-CUBE-SPN5 API is defined in the Motor Control BSP. Its functions are prefixed with `BSP_MotorControl_`.

not all the functions of this module are available on the L6208 device and hence on the expansion board X-NUCLEO-IHM05A1.

Detailed technical information regarding the APIs available to the user can be found in a compiled HTML file in the package Documentation folder, where all the functions and parameters are fully described.

2.6 Sample application description

A sample application using the X-NUCLEO-IHM05A1 STM32 expansion board with a compatible STM32 Nucleo board is provided in the "Projects" directory. Ready-to-build projects are available for multiple IDEs (see *Section 2.3.2: "Projects folder"*).

3 System setup guide

3.1 Hardware description

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

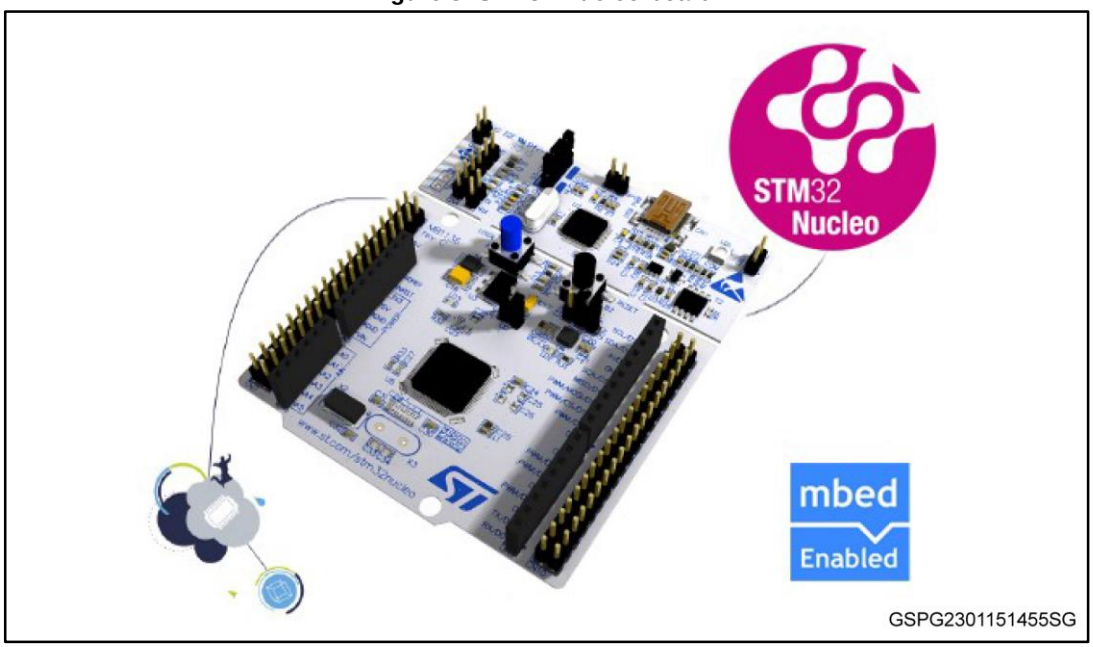
The following subsections describe the individual components.

3.1.1 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 of the 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 choice of specialized expansion boards. The STM32 Nucleo board does not require any separate probes as it integrates the ST-LINK/V2-1 debugger/programmer. The STM32 Nucleo board comes with the STM32 comprehensive software HAL library together with various packaged software examples.

Information about the STM32 Nucleo boards is available on *www.st.com* at *www.st.com/stm32nucleo*

Figure 3: STM32 Nucleo board

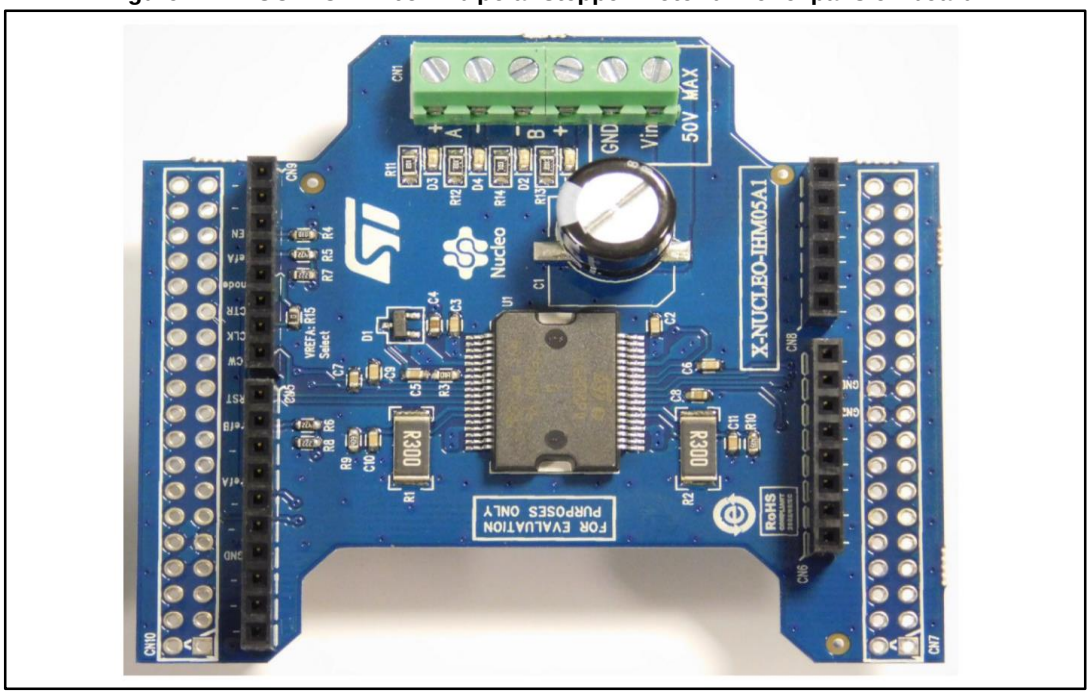


3.1.2 X-NUCLEO-IHM05A1 bipolar stepper motor driver expansion board

The X-NUCLEO-IHM05A1 is a bipolar stepper motor driver expansion board based on the L6208. It provides an affordable and easy-to-use solution for driving a stepper motor in your STM32 Nucleo project.

The X-NUCLEO-IHM05A1 is compatible with the Arduino UNO R3 connector, and supports the addition of other boards.

Figure 4: X-NUCLEO-IHM05A1 bipolar stepper motor driver expansion board



Information about the X-NUCLEO-IHM05A1 expansion board is available on *www.st.com* at *http://www.st.com/x-nucleo*.

3.1.3 Miscellaneous hardware components

To complete the hardware setup, you will need:

- 1 bipolar stepper motor.
- an external DC power supply with 2 electrical cables for the X-NUCLEO-IHM05A1 board.
- a USB cable type A to mini-B to connect the STM32 Nucleo to a PC.

3.2 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-SPN5: an expansion for STM32Cube dedicated to L6208 motor driver application development. The X-CUBE-SPN5 firmware and related documentation is available on *www.st.com*.
- A development toolchain and compiler. Three toolchains are supported:
  - 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)

3.3 Hardware and software setup

This section describes the hardware and software setup procedure to execute the examples provided, and to develop new applications based on the motor driver expansion board.

3.3.1 Setup to drive 1 motor

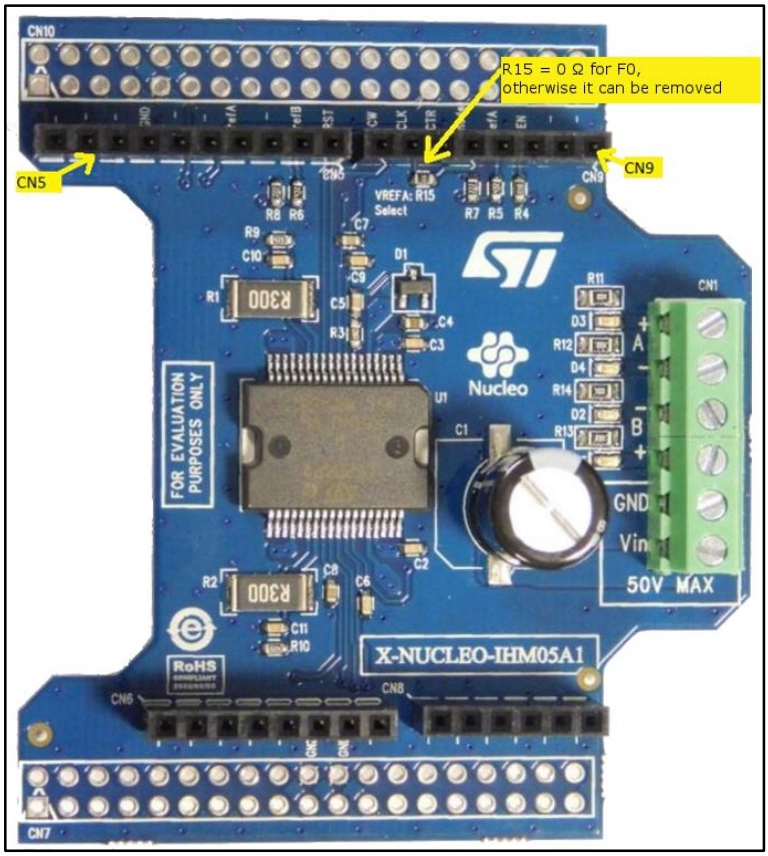
The STM32 Nucleo must be configured with the following jumper positions:

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

X-NUCLEO-IHM05A1 expansion board configuration:

- When using an STM32 Nucleo based on the STM32F334: none
- When using an STM32 Nucleo based on the STM32F401: none
- When using an STM32 Nucleo based on the STM32L053: none
- When using an STM32 Nucleo based on the STM32F030: pin 4 of the CN5 connector must be connected to pin 4 of the CN9 connector, as shown below.

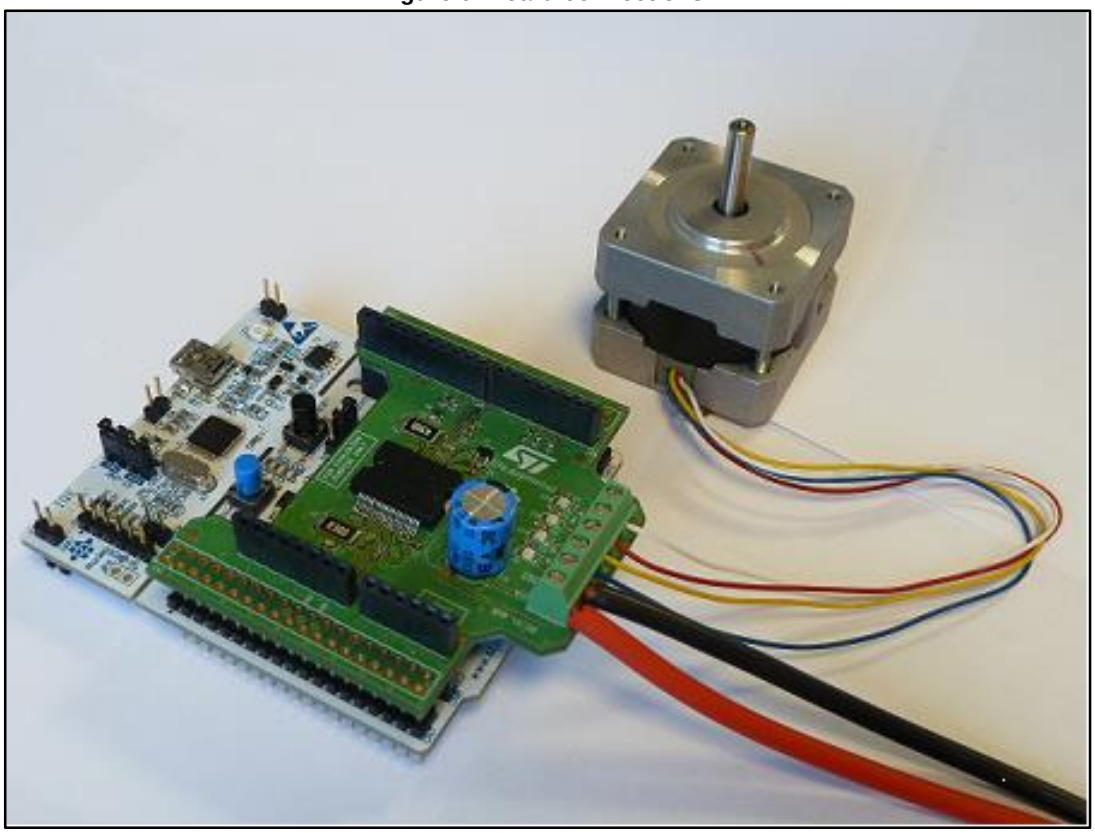
Figure 5: X-NUCLEO-IHM05A1 stepper motor driver configuration for STM32 Nucleo based on STM32F030



Once the board is properly configured:

- Plug the X-NUCLEO-IHM05A1 expansion board on top of the STM32 Nucleo using the Arduino UNO R3 connectors.
- 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-IHM05A1 expansion board by connecting its connectors Vin and GND to the DC power supply. The DC supply must be set to deliver the required voltage to the stepper motor.
- Connect the stepper motor to the X-NUCLEO-IHM05A1 bridge connectors A+/- and B+/-

Figure 6: Board connections



Once system setup is complete:

- 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\IHM05A1_ExampleFor1Motor\YourToolChainName\STM32L053R8-Nucleo for Nucleo STM32L053
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM05A1_ExampleFor1Motor\YourToolChainName\STM32F401RE-Nucleo for Nucleo STM32F401
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM05A1_ExampleFor1Motor\YourToolChainName\STM32F030R8-Nucleo for Nucleo STM32F030
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM05A1_ExampleFor1Motor\YourToolChainName\STM32F334R8-Nucleo for Nucleo STM32F334
- In order to adapt the default parameters used by the L6208 to your stepper motor, use either:
  - the `BSP_MotorControl_Init` function with the NULL pointer and modify stm32_cube\Drivers\BSP\Components\l6208\l6208_target_config.h parameters accordingly
  - the `BSP_MotorControl_Init` function with the address of the `initDevicesParameters` structure with values modified accordingly
- Rebuild all files and load your image into target memory.
- Run the sample application; your motor will start automatically (find program sequence details in main.c)

4 Acronyms and abbreviations

Table 2: Acronyms and abbreviations

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

5 Revision history

Table 3: Document revision history

Date	Revision	Changes
07-Aug-2015	1	Initial release.
10-Mar-2016	2	Text changes throughout document Updated *Figure 2: "Overall software architecture"* Removed API details (now available in a compiled HTML file inside the package “Documentation” folder). Added STM32F3xx compatibility information. Replaced 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.