UM1848
User manual

Getting started with the software package for L6474 stepper motor driver X-CUBE-SPN1 expansion for STM32Cube

Introduction

X-CUBE-SPN1 is a software package based on STM32Cube for the X-NUCLEO-IHM01A1 expansion board. It is compatible with the NUCLEO-F401RE, the NUCLEO-F030R8, the NUCLEO-L053R8 and NUCLEO-F334R8 boards equipped with one or more (up to 3) X-NUCLEO-IHM01A1 boards.

The software is based on the STM32Cube technology to facilitate portability across different STM32 MCU families. 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?

2.1 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.

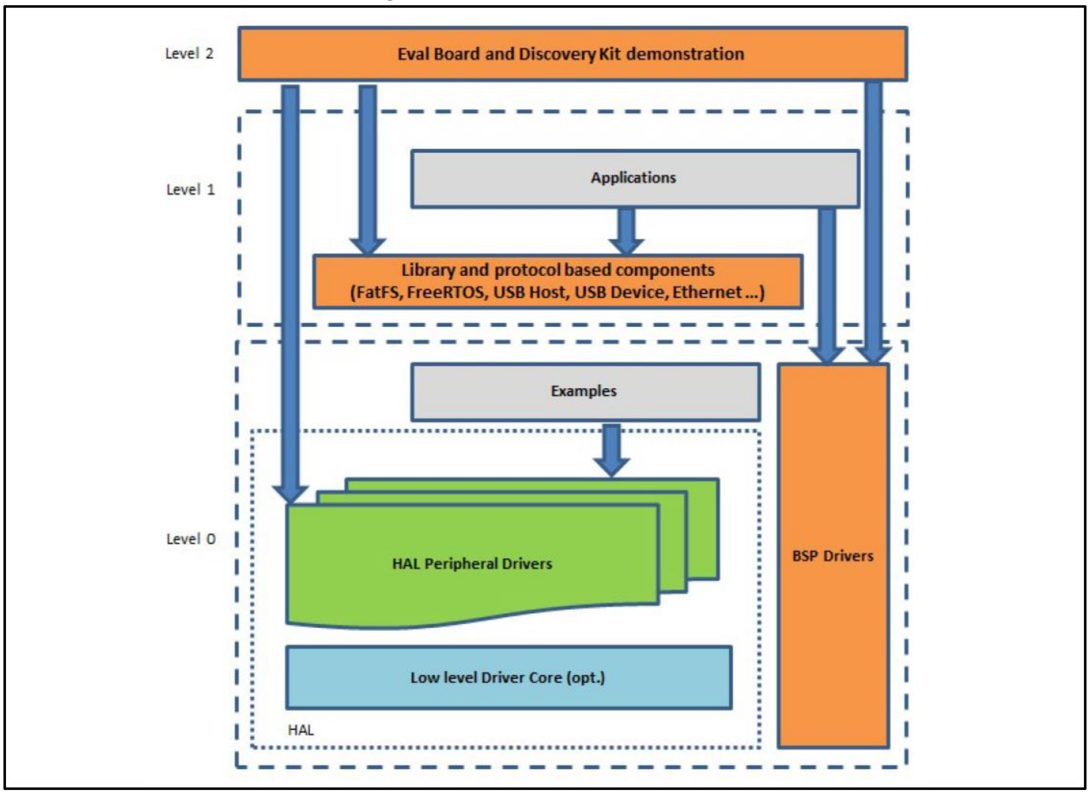
STM32Cube version 1.x includes:

- 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

2.2 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 2
Eval Board and Discovery Kit demonstration
Level 1
Applications
Library and protocol based components
(FatFS, FreeRTOS, USB Host, USB Device, Ethernet...)
Examples
Level 0
HAL Peripheral Drivers
BSP Drivers
Low level Driver Core (opt.)
HAL

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-SPN1 software expansion for STM32Cube

3.1 Overview

The X-CUBE-SPN1 software package allows complete management of the L6474 fully integrated microstepping motor driver by providing complete APIs. It offers the following features:

- L6474 registers read, write
- Nucleo and expansion board configuration (GPIOs, PWMs, IRQs, etc.)
- Speed profile configuration
- Motion commands
- FLAG interrupts handling (alarms reporting)
- Microstepping handling
- Daisy chaining handling

When initializing the L6474 driver, the user specifies the number of L6474 devices which are connected to the STM32 Nucleo Board (i.e. the number of X-NUCLEO-IHM01A1 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 directions and the FLAG interrupt
- initializes the PWMs to act as step clock generators
- initializes the speed profile (acceleration, deceleration, min and max speed) from the parameters in `l6474_target_config.h`
- starts the SPI driver to communicate with the L6474 devices
- releases the reset of each of the L6474 devices
- disables the power bridge and clears the status flags of the L6474 devices
- loads the registers of each L6474 device with the predefined values in `l6474_target_config.h`

Once initialization is complete, the user can modify the L6474 registers and speed profile configurations as desired. Most of the functions of the driver take a device ID (from 0 to 2) as an input parameter so users can specify which device configuration to modify.

The user can also write a callback function and attach it to the Flag interrupt handler, depending on the actions to be performed when an alarm is reported (read the flags, clear and read the flags, etc.).

The user can then request the movement of one or more motors (via device IDs). This request can be:

- to move for a given number of steps in a specified direction
- to go to a specific position
- to run until it receives a new instruction

On reception of this request, the driver enables the PWM which is used as the step clock of the corresponding L6474.

For each pulse period, the motor performs one step and an ISR (interrupt service routine) is triggered on the microcontroller side. This ISR is used by the firmware to count the number of performed steps and to update the speed. Indeed, the motor starts moving by using the minimum speed parameter. At each step, the speed is increased using the acceleration parameter.

If the target position is far enough, the motor will perform a trapezoidal move:

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

Otherwise, if the target position does not allow it to reach maximum speed, the motor will perform a triangular move:

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

A move command can be interrupted at any time by either:

- a soft stop or `softHiz` which progressively decreases the speed using the deceleration parameter. Once the minimum speed is reached, the motor is stopped.
- 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.

To inhibit the sending of a new command to a device before the completion of the previous one, the driver offers the `BSP_MotorControl_WaitWhileActive()` command which locks program execution until the motor stops moving. The driver also allows changing the step mode (from full step to 1/16 microstep mode) for each device. When the step mode is changed, the current position (`ABS_POSITION` register) is automatically reset, but the user must update the speed profile (maximum and minimum speed, acceleration deceleration).

3.2 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 based on the L6474 component. Refer to the previous section for an introduction to the STM32Cube architecture.

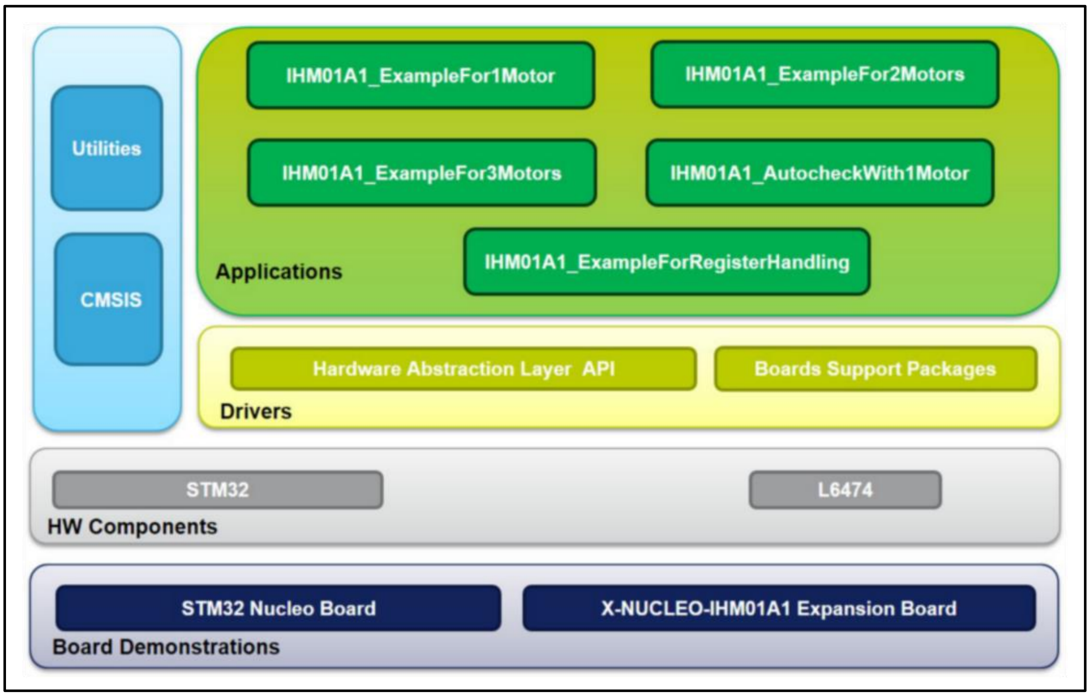
The software is based on the STM32CubeHAL hardware abstraction layer for the STM32 microcontroller. The package extends STM32Cube by providing a board support package (BSP) for the motor control expansion board and a BSP component driver for the L6474 motor driver.

The software layers used by the application software to access and use the 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 and is directly built around a generic architecture so that layers built on top of it, such as the middleware layer, can implement their functions without dependending on specific hardware configurations for a given microcontroller unit (MCU). This structure improves reusability of the library code and guarantees easy portability on other devices.
- **Board support package (BSP) layer**: provides support for all the peripherals on the STM32 Nucleo board, apart from the MCU. It is a limited set of APIs which provides a programming interface for certain board specific peripherals, e.g. the LED, the user button, etc. This interface can also help identify 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-SPN1 software, it is associated with the BSP component for the L6474 motor driver.

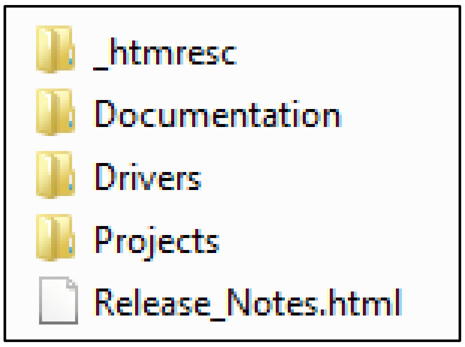
The figure below outlines the software architecture of this package:

Figure 2: X-CUBE-SPN1 software architecture


Utilities
CMSIS
IHM01A1_ExampleFor1Motor
IHM01A1_ExampleFor2Motors
IHM01A1_ExampleFor3Motors
IHM01A1_AutocheckWith1Motor
Applications
IHM01A1_ExampleForRegisterHandling
Hardware Abstraction Layer API
Boards Support Packages
Drivers
STM32
L6474
HW Components
STM32 Nucleo Board
X-NUCLEO-IHM01A1 Expansion Board
Board Demonstrations

3.3 Folder structure

Figure 3: X-CUBE-SPN1 folder structure



The code of the software package is located in two main folders:

- A “Drivers” folder, with:
  - The STM32Cube HAL files to run the motor driver examples are located in the STM32L0xx_HAL_Driver, STM32F0xx_HAL_Driver, STM32F4xx_HAL_Driver or STM32F3xx_HAL_Driver sub-folders. These files are not specific to the X-CUBE-SPN1 software, but derive directly from the STM32Cube framework.
  - A CMSIS folder with the CMSIS (Cortex® Microcontroller Software Interface Standard) vendor-independent files from ARM.
  - The hardware abstraction layer for the Cortex-M processor series.
  - A BSP (board support package) folder with files containing the codes for X-NUCLEO-IHM01A1 configuration, the L6474 driver and the motor control API.
- A “Project” folder, with several usage examples of the L6474 motor driver for different Nucleo platforms.

3.3.1 BSP folder

The X-CUBE-SPN1 software uses the BSPs described in this section.

3.3.1.1 STM32L0XX-Nucleo, STM32F0XX-Nucleo, STM32F4XX-Nucleo or STM32F3XX-Nucleo BSP

Depending on the STM32 Nucleo used, these BSPs provide an interface to configure and use the

Nucleo peripherals with the expansion board X-NUCLEO-IHM01A1. The relevant subfolder contains the following '.c' and '.h' file pairs:

- stm32XXxx_nucleo.c/h: these files are directly from the STM32Cube framework and provide the functions to handle the user button and the LEDs of the corresponding STM32 Nucleo board.
- stm32XXxx_nucleo_ihm01a1.c/h: these files are dedicated to the configuration of the SPI, the PWMs, the GPIOs and interrupt enabling/disabling required for the X-NUCLEO-IHM01A1 expansion board functions.

3.3.1.2 Motor control BSP

This BSP provides a common interface to access the driver functions to configure and control various motor drivers such as L6474 and Powerstep01, via the MotorControl/motorcontrol.c/h files.

The functions are then mapped to the functions of the motor driver component used on a given expansion board via the motorDrv_t (defined in Components\Common\motor.h.) structure file. This structure defines a list of function pointers filled during instantiation in the corresponding motor driver component. For X-CUBE-SPN1, the instance of the structure is called 'l6474Drv' (see BSP\Components\l6474\l6474.c file).

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

3.3.1.3 L6474 BSP component

The L6474 BSP component provides the driver functions of the L6474 fully integrated microstepping motor driver in the stm32_cube\Drivers\BSP\Components\l6474 folder.

This folder has 3 files:

- l6474.c: core functions of the L6474 driver
- l6474.h: declaration of the l6474 driver functions and their associated definitions
- l6474_target_config.h: predefines values for the l6474 registers and for the motor devices context (speed profile)

3.3.2 Projects folder

Five sample projects are available in the stm32_cube\Projects\Multi\Examples\MotionControl\ folder for each Nucleo platform:

- IHM01A1_ExampleFor1Motor: examples of control functions in configuration with 1 motor
- IHM01A1_ExampleFor2Motors: examples of control functions in configuration with 2 motors
- IHM01A1_ExampleFor3Motors: examples of control functions in configuration with 3 motors

- IHM01A1_ExampleForRegisterHandling: examples of L6474 registers handling in configuration with 1 device
- IHM01A1_AutocheckWith1Motor: example where the motor is moved 8 steps when the user button is pushed

Each example has a dedicated folder for the target IDE:

- EWARM with project files for IAR
- MDK-ARM with project files for Keil
- SW4STM32 with 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 L6474)
- 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 Software required resources

Communication between the L6474 and the MCU is mainly through the SPI interface. This requires the use of the CS, MOSI, MISO and SCK (SPI clock) GPIOs. By default, the STM32 Nucleo BSP “stm32f4xx_nucleo_ihm01a1.h” file uses SPI1, but you can use SPI2, provided you declare the `BSP_MOTOR_CONTROL_BOARD_USE_SPI2` preprocessor option.

The step clock for each L6474 device is generated by a PWM. The frequency of this PWM can be updated at each pulse depending on the speed profile of the device. This is why the library requires one timer and its corresponding ISR by device.

For flag interrupt handling, the X-CUBE-SPN1 software uses only one external interrupt for all devices, as all flag pins are connected together.

One GPIO per device is also required for the handling of the direction. And finally, one common GPIO is used for the reset of the L6474 devices.

Table 2: Required resources for the X-CUBE-SPN1 software

Resources F4xx	Resources F0xx	Resources L0xx	Resources F3xx	Digital pin	Features	Device
	Ext Line 10 GPIO PA10		Ext Line 10 GPIO PA10	2	Flag interrupt	All
	GPIO PA9		GPIO PA9	8	L6474 reset	All
	CS GPIO PB6		CS GPIO PB6	10	SPI chip select	All
	MOSI GPIO PA7 for SPI1 GPIO PB15 for SPI2		MOSI GPIO PA7 for SPI1 GPIO PB15 for SPI2	11	SPI master out slave in	All

Resources F4xx	**Resources F0xx**	**Resources L0xx**	**Resources F3xx**	**Digital pin**	**Features**	**Device**
	MISO GPIO PA6 for SPI1 GPIO PB14 for SPI2		MISO GPIO PA6 for SPI1 GPIO PB14 for SPI2	12	SPI master in slave out	All
	SCK GPIO PA5 for SPI1 GPIO PB13 for SPI2		SCK GPIO PA5 for SPI1 GPIO PB13 for SPI2	13	SPI serial clock	All
Timer 3 Ch2 GPIO PC7	Timer 3 Ch2 GPIO PC7	Timer 22 Ch2 GPIO PC7	Timer 3 Ch2 GPIO PC7	9	PWM 1	0
	GPIO PA8		GPIO PA8	7	Direction 1	0
Timer 2 Ch2 GPIO PB3	Timer 14 Ch1 GPIO PB3	Timer 2 Ch2 GPIO PB3	Timer 14 Ch1 GPIO PB3	3	PWM 2	1
	GPIO PB5		GPIO PB5	4	Direction 2	1
Timer 4 Ch3 GPIO PB10	Timer 15 Ch1 GPIO PB10	Timer 21 Ch1 GPIO PB10	Timer 16 Ch1 GPIO PB10	6	PWM 3	2
	GPIO PB4		GPIO PB4	5	Direction 3	2

Of course, if device 2 and/or 1 are unplugged, the corresponding resources are free.

Due to pinning constraints on the STM32F401 MCU (only Timer 2 is available on GPIO PB10) and to have an independent frequency for each PWM, PWM3 is not handled by hardware. Indeed, GPIO PB10 is toggled by software from the ISR of Timer 4.

Due to pinning constraints on the STM32F334 MCU (only Timer 16 is available on GPIO PB10) and to have an independent frequency for each PWM, PWM3 is not handled by hardware. Indeed, GPIO PB10 is toggled by software from the ISR of Timer 16.

Due to pinning constraints on the STM32F030 MCU (no Timer is available on GPIOs PB3 and PB10) and to have an independent frequency for each PWM, PWM2 and PWM3 are not handled by hardware. Indeed, GPIO PB3 is toggled by software from the ISR of Timer 14 and the GPIO PB10 is toggled by software from the ISR of Timer 15.

Due to pinning constraints on the STM32L053 MCU (only Timer 2 is available on GPIO PB10) and to have an independent frequency for each PWM, PWM3 is not handled by hardware. Indeed, GPIO PB10 is toggled by software from the ISR of Timer 21.

3.5 APIs

The API of the X-CUBE-SPN1 software is defined in the motor control BSP. Its functions are prefixed by `BSP_MotorControl_`. Incidentally, not all the functions of this module are available for the L6474 device and hence for the expansion board X-NUCLEO-IHM01A1.

Technical descriptions of the API functions and parameters available to the user can be found in a compiled HTML file located inside the package “Documentation” folder.

These functions are divided in two groups:

- Motor control: high level functions for simple motor movement handling.
- L6474 control: low level functions for L6474 device configuration.

4 Getting started

4.1 Hardware description

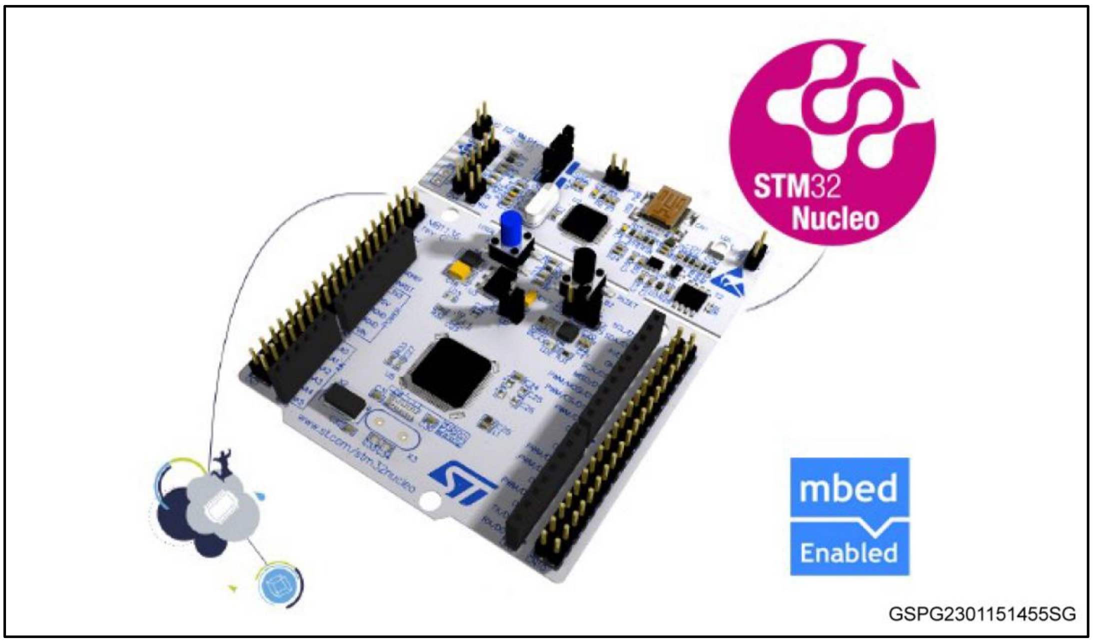
This section describes the hardware components which are required to execute the XCUBE-SPN1 software and to successfully drive one or several stepper motors. The following sub-sections describe the individual components.

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 STM32 Nucleo boards is available on www.st.com at http://www.st.com/stm32nucleo

Figure 4: STM32 Nucleo board

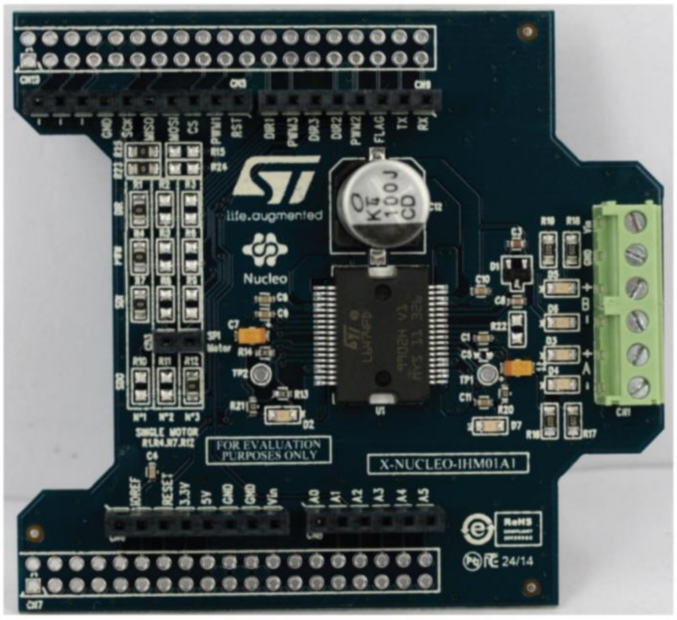


4.1.2 X-NUCLEO-IHM01A1 stepper motor driver expansion board

The X-NUCLEO-IHM01A1 is a stepper motor driver expansion board based on the L6474. It provides an affordable and easy-to-use solution for driving a stepper motor in your STM32 Nucleo project.

The X-NUCLEO-IHM01A1 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-IHM01A1 stepper motor driver expansion board



Information about the X-NUCLEO-IHM01A1 expansion board is available on www.st.com at *http://www.st.com/x-nucleo*.

4.1.3 Miscellaneous HW components

To complete the HW setup, you will need:

- 1 to 3 stepper motors
- an external DC power supply with 2 electric cables for the X-NUCLEO-IHM01A1 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:

- The X-CUBE-SPN1 expansion for STM32Cube dedicated to L6474 motor driver application development. The X-CUBE-SPN1 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)

4.3 Hardware and software setup

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

4.3.1 Setup to drive one motor

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

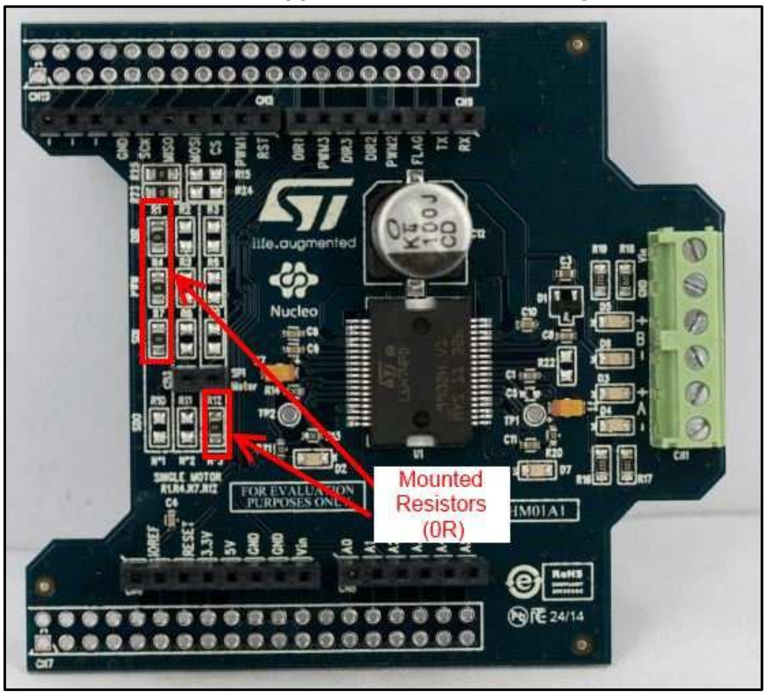
- JP1 off

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

The X-NUCLEO-IHM01A1 expansion board must have:

- Resistors R1, R4, R7 and R12 mounted (0R)
- Resistors R2, R3, R5, R6, R8, R9, R10 and R11 not mounted

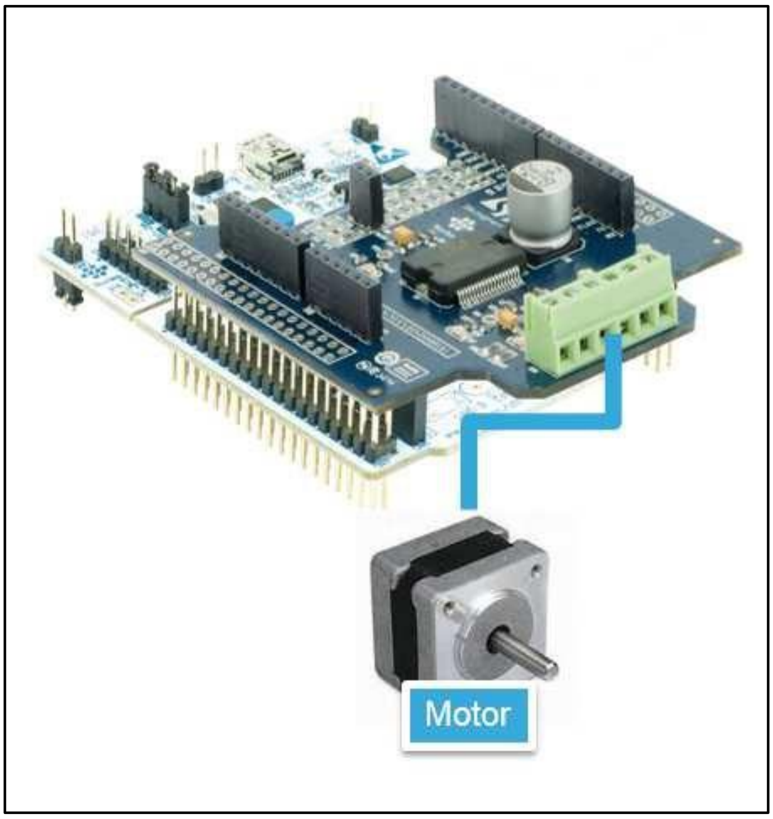
Figure 6: X-NUCLEO-IHM01A1 stepper motor driver in configuration to drive 1 motor



Once the boards are properly configured:

- Plug the X-NUCLEO-IHM01A1 expansion board onto the STM32 Nucleo by 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-IHM01A1 expansion board by connecting the Vin and GND connectors 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-IHM01A1 bridge connectors A± and B±.

Figure 7: Configuration for 1 motor



Once the system setup is ready:

- 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: In order to adapt the default parameters which are used by the L6474 to your stepper motor characteristics, open the file: stm32_cube\Drivers\BSP\Components\l6474\ l6474_target_config.h. and modify the parameters ending in "_DEVICE_0".
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM01A1_ExampleFor1Motor\YourToolChainName\STM32F401RE-Nucleo for Nucleo STM32F401
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM01A1_ExampleFor1Motor\YourToolChainName\STM32F030R8-Nucleo for Nucleo STM32F030
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM01A1_ExampleFor1Motor\YourToolChainName\STM32F334R8-Nucleo for Nucleo STM32F334
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM01A1_ExampleFor1Motor\YourToolChainName\STM32L053R8-Nucleo for Nucleo STM32L053
- Rebuild all files and load your image into target memory
- Run the example; the motor starts automatically (see main.c for the detailed demo sequence).

4.3.2 Setup to drive 2 motors

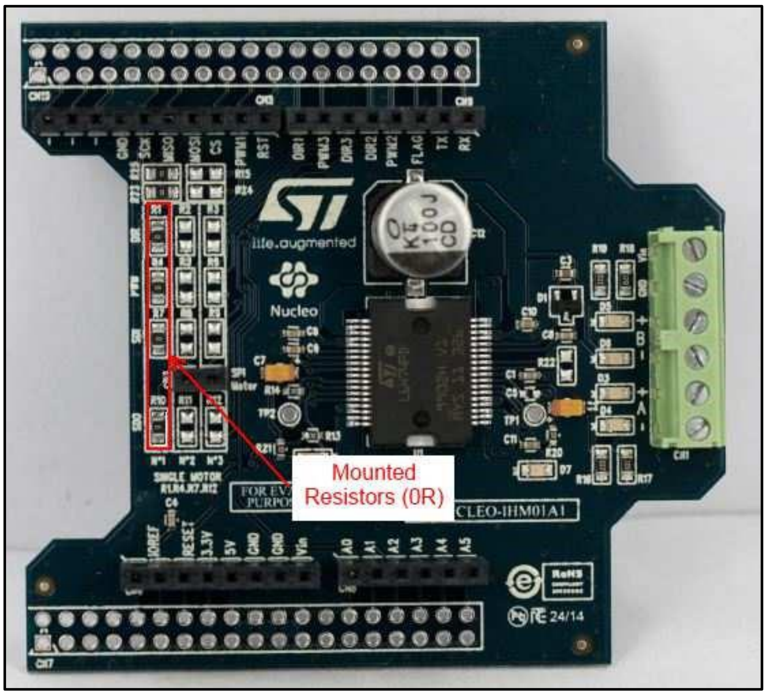
The STM32 Nucleo has to be configured with the following jumper positions:

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

The X-NUCLEO-IHM01A1 expansion board must have, for the first motor:

- Resistors R1, R4, R7 and 10 mounted (0R)
- Resistors R2, R3, R5, R6,R8, R9, R11 and R12 not mounted

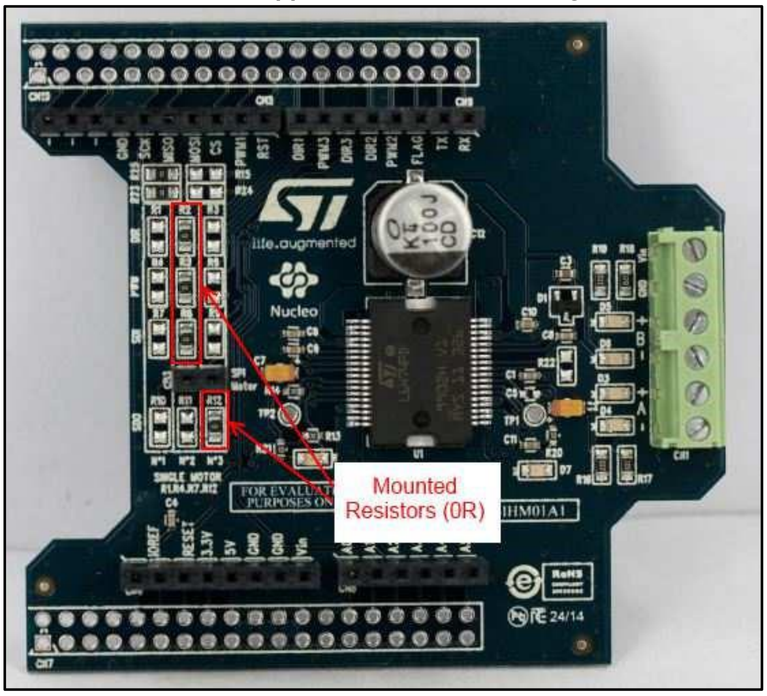
Figure 8: X-NUCLEO-IHM01A1 stepper motor driver in configuration to drive motor ½



The X-NUCLEO-IHM01A1 expansion board must have, for the second motor:

- Resistors R2, R5, R8 and R12 mounted (0R)
- Resistors R1, R3, R4, R6, R7, R9, R10 and R11 not mounted

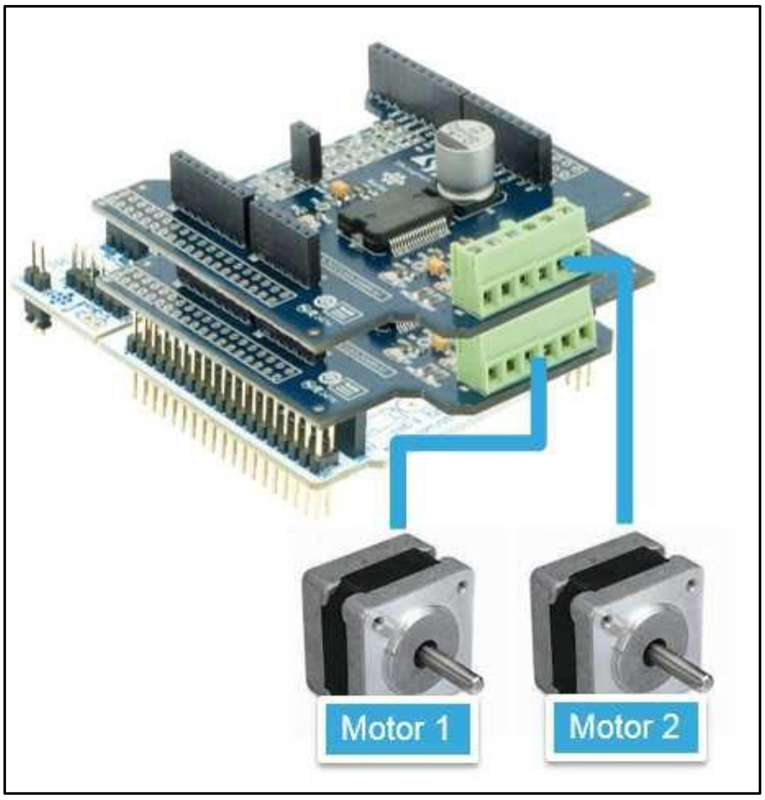
Figure 9: X-NUCLEO-IHM01A1 stepper motor driver in configuration to drive motor 2/2



Once the boards are properly configured:

- Plug the X-NUCLEO-IHM01A1, for first motor, onto the STM32 Nucleo by using the Arduino UNO R3 connectors
- Plug the X-NUCLEO-IHM01A1 for the second motor onto 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-IHM01A1 expansion boards by connecting the Vin and Gnd connectors to the DC power supply. The DC supply must be set to deliver the required voltage to the stepper motors.
- Connect each stepper motor to the bridge connectors A± and B± of their dedicated X-NUCLEO-IHM01A board

Figure 10: Configuration for 2 motors



Once the system setup is ready:

- 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\IHM01A1_ExampleFor2Motors\YourToolChainName\STM32F401RE-Nucleo for Nucleo STM32F401
  – \stm32_cube\Projects\Multi\Examples\MotionControl\IHM01A1_ExampleFor2Motors\YourToolChainName\STM32F030R8-Nucleo for Nucleo STM32F030
  – \stm32_cube\Projects\Multi\Examples\MotionControl\IHM01A1_ExampleFor2Motors\YourToolChainName\STM32F334R8-Nucleo for Nucleo STM32F334
  – \stm32_cube\Projects\Multi\Examples\MotionControl\IHM01A1_ExampleFor2Motors\YourToolChainName\STM32L053R8-Nucleo for Nucleo STM32L053
- In order to adapt the default parameters which are used by the L6474s to your stepper motor characteristics, open the file stm32_cube\Drivers\BSP\Components\l6474\l6474_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 example; the motor automatically starts (see main.c for the detailed demo sequence).

4.3.3 Setup to drive 3 motors

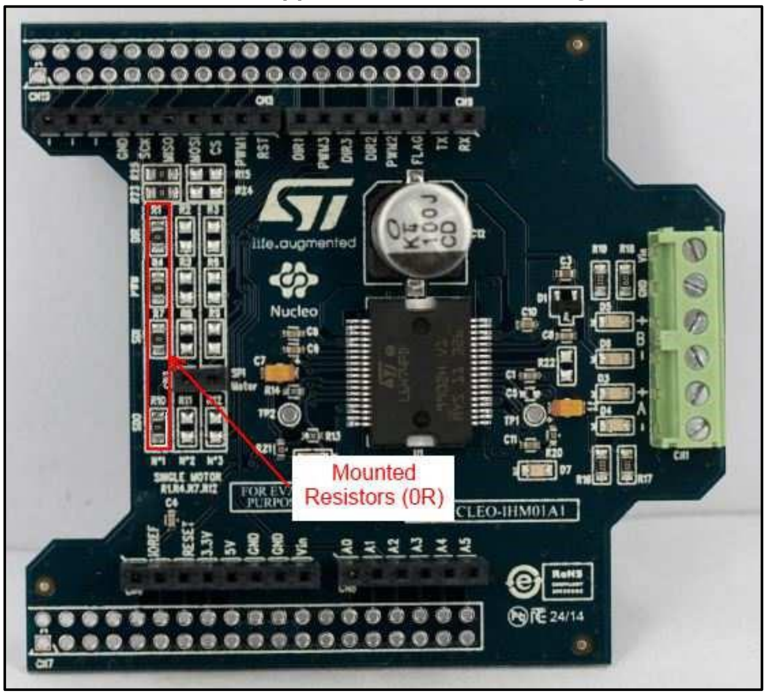
The STM32 Nucleo must be configured with the following jumper positions:

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

The X-NUCLEO-IHM01A1 expansion board must have, for first motor:

- Resistors R1, R4, R7 and 10 mounted (0R)
- Resistors R2, R3, R5, R6,R8, R9, R11 and R12 not mounted

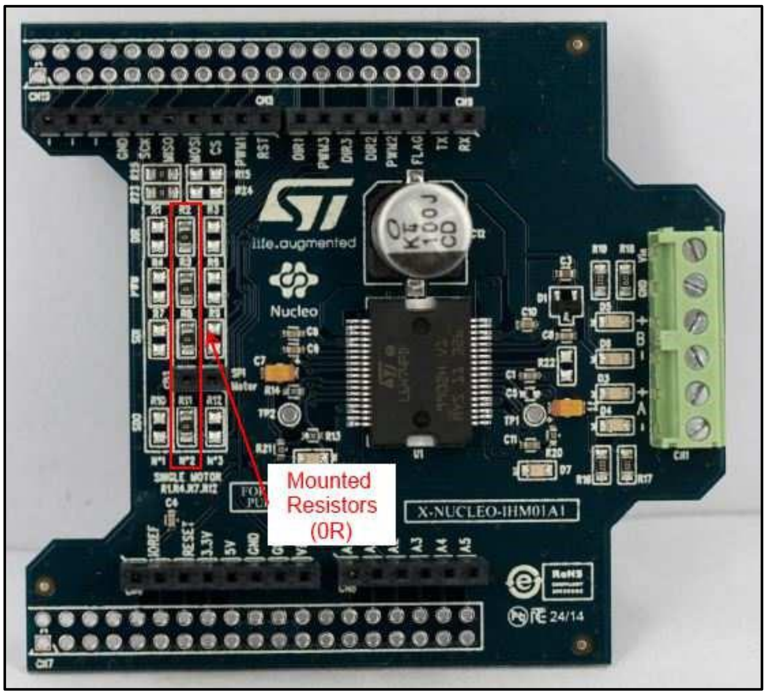
Figure 11: X-NUCLEO-IHM01A1 stepper motor driver in configuration to drive motor 1/3



The X-NUCLEO-IHM01A1 expansion board must have, for the second motor:

- Resistors R2, R5, R8 and R11 mounted (0R)
- Resistors R1, R3, R4, R6,R7, R9, R10 and R12 not mounted

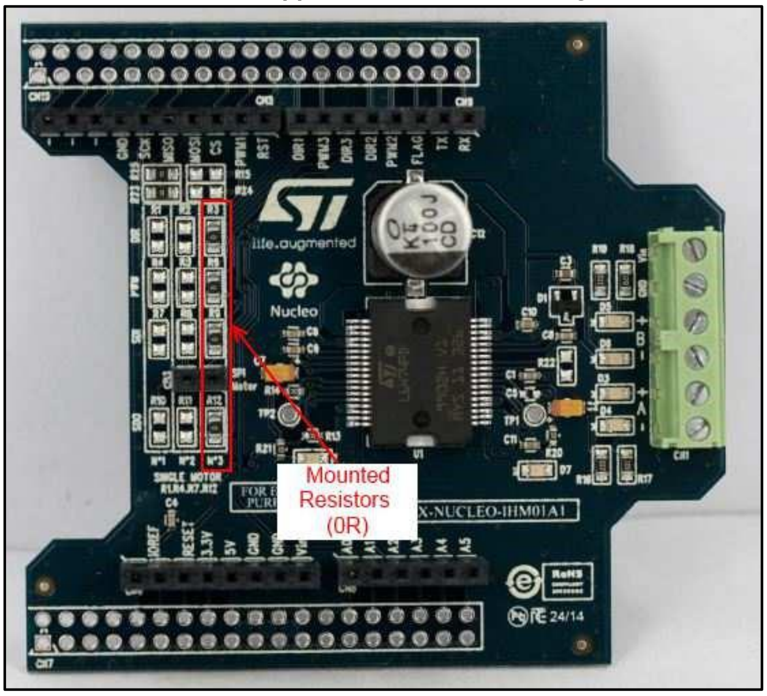
Figure 12: X-NUCLEO-IHM01A1 stepper motor driver in configuration to drive motor 2/3



The X-NUCLEO-IHM01A1 expansion board must have, for the third motor:

- Resistors R3, R6, R9 and R12 mounted (0R)
- Resistors R1, R2, R4, R5, R7, R8, R10 and R11 not mounted

Figure 13: X-NUCLEO-IHM01A1 stepper motor driver in configuration to drive motor 3/3

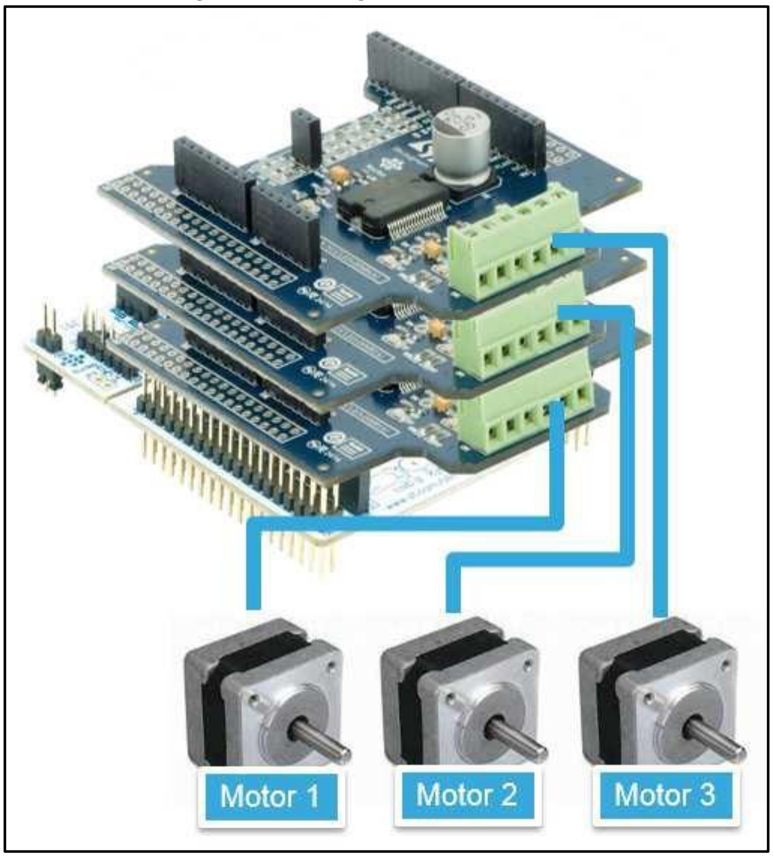


Once the boards are properly configured:

- Plug the X-NUCLEO-IHM01A1 for the first motor onto the STM32 Nucleo by using the Arduino UNO R3 connectors

- Plug the X-NUCLEO-IHM01A1 for the second motor onto the one for the first motor
- Plug the X-NUCLEO-IHM01A1 for the third motor onto 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-IHM01A1 expansion boards by connecting the Vin and Gnd connectors to the DC power supply. The DC supply must be set to deliver the required voltage to the stepper motors.
- Connect each stepper motor to the bridge connectors A± and B± of their dedicated X-NUCLEO-IHM01A1 board

Figure 14: Configuration for 3 motors



Once the system setup is ready:

- 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\IHM01A1_ExampleFor3Motors\YourToolChainName\STM32F401RE-Nucleo for Nucleo STM32F401
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM01A1_ExampleFor3MotorsYourToolChainName\STM32F030R8-Nucleo for Nucleo STM32F030
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM01A1_ExampleFor3Motors\YourToolChainName\STM32F334R8-Nucleo for Nucleo STM32F334
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM01A1_ExampleFor3Motors\YourToolChainName\STM32L053R8-Nucleo for Nucleo STM32L053
- In order to adapt the default parameters used by the L6474s to your stepper motor characteristics, open the file: stm32_cube\Drivers\BSP\Components\l6474\l6474_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 motor starts automatically (see main.c for the detailed demo sequence).

5 Revision history

Table 3: Document revision history

Date	Version	Changes
12-Mar-2015	1	Initial release.
19-Feb-2016	2	Text changes throughout document Added STM32F3xx support information Replaced references to TrueStudio by System Workbench with STM32 (SW4STM32) Removed section "Motor control functions" Removed section "L6474 Control functions"