UM1911
User manual

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

Introduction

This user manual presents how to use the X-CUBE-SPN3 expansion software within the STM32Cube software environment. Combined with the use of one or several X-NUCLEO-IHM03A1 boards, this software allows an STM32 Nucleo board to control one or more stepper motors. The X-NUCLEO-IHM03A1 high power stepper motor driver expansion board is designed around STMicroelectronics' powerSTEP01, a system-in-package micro-stepping controller with power MOSFETs.

This document does not cover the powerSTEP01 device operation. For this please refer to the powerSTEP01 datasheet "DS9836: System-in-package integrating microstepping controller and 10 A power MOSFETs", which is available on www.st.com

The software is based on STM32Cube technology and expands STM32Cube based packages.

Contents

1 What is STM32Cube?

1.1 STM32Cube overview

The STM32Cube™ initiative was originated by STMicroelectronics to ease developers' life by reducing development effort, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube version 1.x includes:

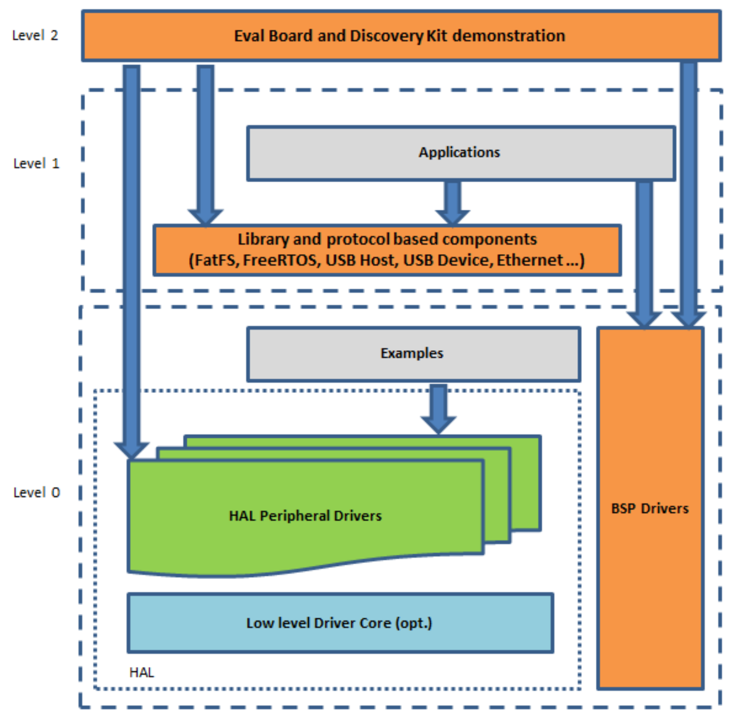
- The STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards
- A comprehensive embedded software platform, delivered per series (such as STM32CubeF4 for the STM32F4 series)
- The STM32Cube HAL, an STM32 abstraction layer embedded software, which ensures maximized portability across the STM32 portfolio
- A consistent set of middleware components such as RTOS, USB, TCP/IP, graphics
- All embedded software utilities with a full set of examples

Information about the STM32Cube is available at: http://www.st.com/stm32cube.

1.2 STM32Cube architecture

The STM32Cube firmware solution is built on three independent levels that can easily interact with each other, as shown 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 is composed of two parts:
  - Component: the driver relative to the external device on the board and not related to the STM32. The component driver provides specific APIs to the BSP driver external components and could be portable to any other board.
  - BSP driver: permits linking of the component driver to a specific board and provides a set of user-friendly APIs. The API naming rule is BSP_FUNCT_Action(): ex. BSP_LED_Init(),BSP_LED_On()

It's based on modular architecture allowing easy porting to any hardware simply by 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 functionality-oriented APIs which permit offloading of the user's application implementation by providing a ready-to-use process. For example, for the communication peripherals (I^2S, UART, etc.) it provides APIs that allow initialization and configuration of the peripheral, management of data transfer based on polling, interrupt or DMA processes, and management of communication errors that may arise during communication. The HAL driver APIs are split into two categories: generic APIs which provide common and generic functions to all the STM32 series, and extension APIs which provides customized functions for a specific family or a specific part number.
- Basic peripheral usage examples: this layer encloses the examples build over the STM32 peripheral using only the HAL and BSP resources.

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 between the components of this layer is done directly by calling the feature APIs while the vertical interaction with the low level drivers is done through specific callbacks and static macros implemented in the library system call interface. For example, the FatFs implements the disk I/O driver to access the microSD drive or the USB mass storage class.
- Examples based on the middleware components: each middleware component comes with one or more examples (also called Applications) showing how to use it. Integration examples that use several middleware components are provided as well.

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

2 X-CUBE-SPN3 software expansion for STM32Cube

2.1 Overview

X-CUBE-SPN3 is a software package that expands the functionality provided by STM32Cube. The key features of the package are:

- powerSTEP01 read/write registers
- Nucleo and expansion board configuration (GPIOs, PWMs, IRQs, etc.)
- Application commands
- FLAG and BUSY interrupt handling (alarm reporting)
- Daisy chain handling

The registers allow the user to easily:

- 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

The application commands allow the user to easily:

- handle micro-stepping (up to 1/128)
- handle step-clock
- perform various positioning, moves and stops
- get system status

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 the initialization is done, 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 will perform a trapezoidal move:

- 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

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

- 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.

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

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 without problem.

2.2 Architecture

This software is an expansion for STM32Cube, as such it fully complies with the architecture of STM32Cube and it 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.

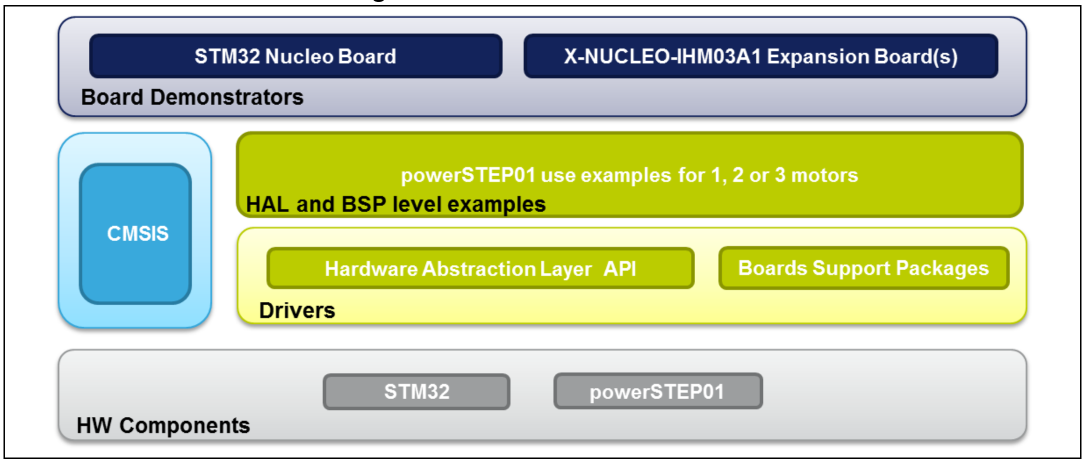
The software is based on the STM32CubeHAL, the hardware abstraction layer for the STM32 microcontroller. The package extends STM32Cube by providing a board support package (BSP) for the motor control expansion board and a BSP component driver for powerSTEP01 motor driver.

The software layers used by the application software to access and use the stepper motor drivers 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 dependencies on the specific hardware configuration for a given Microcontroller Unit (MCU). This structure improves the library code reusability and guarantees an easy portability on 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, the user button etc. This interface also helps in identifying the specific board version. In the case of the 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


STM32 Nucleo Board
X-NUCLEO-IHM03A1 Expansion Board(s)
Board Demonstrators
CMSIS
powerSTEP01 use examples for 1, 2 or 3 motors
HAL and BSP level examples
Hardware Abstraction Layer API
Boards Support Packages
Drivers
STM32
powerSTEP01
HW Components

2.3 Folder structure

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

- A “Drivers” folder which provides:
  - the STM32Cube HAL required files. These files are located in the subfolders STM32L0xx_HAL_Driver, STM32F0xx_HAL_Driver or STM32F4xx_HAL_Driver. These files are not described here as they are no specific to the X-CUBE-SPN3 software but come directly from the STM32Cube framework. Note that only the HAL files which are required to run the motor driver examples are present.
  - a CMSIS folder which contains the CMSIS (Cortex® microcontroller software interface standard) files from ARM. These files are a vendor-independent hardware abstraction layer for the Cortex-M processor series. This folder comes also unchanged from the STM32Cube framework.
  - a BSP (board support package) folder whose files contain the codes required to the X-NUCLEO-IHM03A1 configuration, the powerSTEP01 driver and the motor control API. The content of this folder is detailed further.
- A “Project” which contains several use examples of the powerSTEP01 motor driver for different Nucleo platforms.

2.3.1 BSP folder

In case of the X-CUBE-SPN3 software, three different BSP are used:

STM32L0XX-Nucleo/STM32F0XX-Nucleo/ STM32F4XX-Nucleo BSPs

Depending of the used Nucleo, these BSPs provides an interface to configure and use the Nucleo peripherals with the expansion board X-NUCLEO-IHM03A1. Under each subfolder STM32L0XX-Nucleo/STM32F0XX-Nucleo/ STM32F4XX-Nucleo, there are 2 pairs of .c/.h files:

- **stm32XXxx_nucleo.c/h**: these files come from the STM32Cube framework without modification and provide the functions to handle the user button, the LEDs of the corresponding Nucleo board.
- **stm32XXxx_nucleo_ihm03a1.c/h**: these files are dedicated to the configuration of the SPI, the PWMs, the GPIOs, the interrupt enabling/disabling which are required for the working of the X_NUCLEO_IHM03A1 expansion board.

Motor control BSP

This BSP provides a common interface to access the driver functions of various motor driver like L6474, Powerstep01, etc. This is done via a few c/h files: MotorControl/motorcontrol.c/h. These files define all the functions which can be used 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 structure file: motorDrv_t (defined in Components\Common\motor.h.). This structure defines a list of function pointers which are filled during its instantiation in the corresponding motor driver component. In the case of the X-CUBE-SPN3, the instance of the structure is called powerstep01Drv (see file: BSP\Components\powerstep01\powerstep01.c).

As the motor control BSP is common for all motor driver expansion board, all its functions are not 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 null pointer.

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 3 files:

- **powerstep01.c**: core functions of the powerSTEP01 driver
- **powerstep01.h**: declaration of the powerSTEP01 driver functions and their associated definitions
- **powerstep01_target_config.h**: predefines values for the powerSTEP01 registers and for the motor devices context

2.3.2 Projects folder

For each Nucleo platform, 5 examples projects are available from the folder stm32_cube\Projects\Multi\Examples\MotionControl\:

- IHM03A1_ExampleFor1Motor: examples of control functions in configuration with 1 motor
- IHM03A1_ExampleFor2Motors: examples of control functions in configuration with 2 motors
- IHM03A1_ExampleFor3Motors: examples of control functions in configuration with 3 motors
- IHM03A1_ExampleForRegisterHandling: examples of powerSTEP01 registers handling in configuration with 1 device
- IHM03A1_AutocheckWith1Motor: example where the motor is moved of 8 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
- TrueSTUDIO where are located the project files for Atollic True Studio

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

2.4 Software required resources

The communication between the powerSTEP01 and the MCU is mainly done through the SPI interface. This requires the use of 4 GPIOs: CS, MOSI, MISO, SCK (SPI Clock). By default, the file “stm32f4xx_nucleo_ihm03a1.h” of the Nucleo BSP uses the SPI1. But it is possible to use the SPI2 provided to declare the preprocessor option: BSP_MOTOR_CONTROL_BOARD_USE_SPI2.

It is not needed to supply a step clock to a powerSTEP01 device as the speed profile generation and the positioning calculations are fully digitally controlled through SPI programming. Using SPI and daisy chaining, several expansion boards can be programmed with completely independent direction, speed, and position targets.

The motion generation using an external step clock is still possible as an option by configuring the STEP_MODE register: a GPIO is required for this task. Note that all expansion boards share the same GPIO for the step clock generation.

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

The same rule applies for the busy interrupt.

The handling of the direction is also done by SPI programming.

And finally, one common GPIO is used for the reset of all the powerSTEP01 devices.

Table 1. Required resources for the X-CUBE-SPN3 software

Resources F4xx	Resources 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	Timer 3 Ch2 GPIO PC7	9	Step clock	All
	CS GPIO PB6		10	SPI chip select	All

Table 1. Required resources for the X-CUBE-SPN3 software

Resources F4xx	Resources F0xx	Resources L0xx	Digital pin	Features	Device
	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

2.5 APIs

The API of the X-CUBE-SPN3 software is defined in the motor control BSP. Its functions are prefixed by BSP_MotorControl_. Not all of the functions of this module are available for the powerSTEP01 and thus for the expansion board X-NUCLEO-IHM03A1. Here we will describe only the functions which are available for X-CUBE-SPN3. These functions are either high level functions which simply allow the handling of the motor movement or low level functions which are dedicated to the powerSTEP01 chip configuration.

2.5.1 BSP_MotorControl_AttachBusyInterrupt()

```
void BSP_MotorControl_AttachBusyInterrupt(void(*)(void) callback)
```

Description

Attaches a user callback to the Busy interrupt Handler. The call back will then be called each time the library detects a BUSY signal falling edge.

Parameters

`[in] callback`	Name of the callback to attach to the Busy interrupt Handler

Return values

None

Example

```
#include "main.h"
static void MyBusyInterruptHandler(void);
int main(void)
{
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Attach the function MyFlagInterruptHandler (defined below) to the flag
interrupt */
  BSP_MotorControl_AttachBusyInterrupt(MyBusyInterruptHandler);
…
}
/**
  * @brief  This function is the User handler for the busy interrupt
  * @param  None
  * @retval None
  */
void MyBusyInterruptHandler(void)
{
  if (BSP_MotorControl_CheckBusyHw())
  {
  /* Busy pin is low, so at list one Powerstep01 chip is busy */
  /* To be customized (for example Switch on a LED) */
  }
  else
  {
  /* To be customized (for example Switch off a LED) */
  }
}
```

2.5.2 BSP_MotorControl_AttachErrorHandler()

`void BSP_MotorControl_AttachErrorHandler( void(*)(uint16_t) callback)`

Description

Attaches a user callback to the error Handler. The call back will then be called each time the library detects an error.

Parameters

`[in] callback`	Name of the callback to attach to the error handler

Return values

None

Example

```
#include "main.h"
static void Error_Handler(uint16_t error);
int main(void)
{
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 2 devices */
  /* The powerSTEP01 registers are set with the predefined values */
  /* from file powerstep01_target_config.h*/
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01,2);
/* Attach the function Error_Handler (defined below) to the error Handler*/
```

```
  BSP_MotorControl_AttachErrorHandler(MyErrorHandler);
…
}
/**
  * @brief  This function is executed in case of error occurrence.
  * @param[in] error Number of the error
  * @retval None
  */
static void MyErrorHandler(uint16_t error)
{
  /* Backup error number */
  gLastError = error;
/* Infinite loop */
  while(1)
  {
  }
}
```

2.5.3 BSP_MotorControl_AttachFlagInterrupt()

`void BSP_MotorControl_AttachFlagInterrupt(void(*)(void) callback)`

Description

Attaches a user callback to the flag Interrupt. The call back will be then called each time the status flag pin is pulled down due to the occurrence of a programmed alarms (OCD, thermal pre-warning or shutdown, UVLO, wrong command, non-performable command).

Parameters

`[in] callback`	Name of the callback to attach to the flag interrupt

Return values

None

Example

```
#include "main.h"
static void MyFlagInterruptHandler(void);
int main(void)
{
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Attach the function MyFlagInterruptHandler (defined below) to the flag
interrupt */
  BSP_MotorControl_AttachFlagInterrupt(MyFlagInterruptHandler);
…
}
/**
  * @brief  This function is the User handler for the flag interrupt
  * @param  None
  * @retval None
  */
void MyFlagInterruptHandler(void)
{
  /* Get the value of the status register via the powerSTEP01 command
GET_STATUS: this will clear the flags */
  uint16_t statusRegister = BSP_MotorControl_CmdGetStatus(0);
  }
```

2.5.4 BSP_MotorControl_BusyInterruptHandler()

`void BSP_MotorControl_ BusyInterruptHandler(void )`

Description

Handlers of the busy interrupt which calls the user callback (if defined).

This function is supposed to be called from the external line callback which is associated to the busy interrupt.

Parameters

None

Return values

None

Example

```
#include "main.h"
static void MyBusyInterruptHandler(void);
int main(void)
{
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Attach the function MyBusyInterruptHandler (defined below) to the busy
interrupt */
  BSP_MotorControl_AttachBusyInterrupt(MyBusyInterruptHandler);
…
}
/**
  * @brief  This function is the User handler for the busy interrupt
  * @param  None
  * @retval None
  */
void MyBusyInterruptHandler(void)
{
  if (BSP_MotorControl_CheckBusyHw())
  {
    /* Busy pin low, so at list one Powerstep01 chip is busy */
    /* To be customized (for example Switch on a LED) */
  }
  else
  {
    /* To be customized (for example Switch off a LED) */
  }
}
```

Then in file: stm32f4xx_hal_msp.c or stm32f0xx_hal_msp.c or stm32l0xx_hal_msp.c:

```
/**
  * @brief External Line Callback
  * @param[in] GPIO_Pin pin number
  * @retval None
  */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if (GPIO_Pin == BSP_MOTOR_CONTROL_BOARD_BUSY_PIN)
  {
    BSP_MotorControl_BusyInterruptHandler();
  }
}
```

2.5.5 BSP_MotorControl_CheckBusyHw()

```
uint8_t BSP_MotorControl_ CheckBusyHw ( void )
```

Description

Handlers of the busy interrupt which calls the user callback (if defined).

This function is supposed to be called from the external line callback which is associated to the busy interrupt.

Parameters

None

Return values

One if at least one device is busy, otherwise zero

Example

```
#include "main.h"
int main(void)
{
  Uint8_t busy;
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 3 devices */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 3);
/* Check if at least one of the three devices is busy */
  busy = BSP_MotorControl_CheckBusyHw();
}
```

2.5.6 BSP_MotorControl_CheckStatusHw()

`uint8_t BSP_MotorControl_ CheckStatusHw ( void )`

Description

Checks if at least one device has an alarm flag set by reading flag pin position. The flag pin is shared between all devices.

Parameters

None

Return values

One if at least one device has an alarm flag set, otherwise zero

Example

```
#include "main.h"
int main(void)
{
  Uint8_t status;
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 3 devices */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 3);
/* Check if at least one of the three devices has an alarm flag set */
  status = BSP_MotorControl_CheckStatusHw();
}
```

2.5.7 BSP_MotorControl_CmdGetParam()

`uint32_t BSP_MotorControl_CmdGetParam ( uint8_t deviceId, uint32_t param )`

Description

Issues the GetParam command to the specified powerSTEP01 device.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)
`[in] param`	Register address of the powerSTEP01 from powerstep01_Registers_t enum:

```
typedef enum {
          POWERSTEP01_ABS_POS      = ((uint8_t)0x01),
          POWERSTEP01_EL_POS       = ((uint8_t)0x02),
          POWERSTEP01_MARK         = ((uint8_t)0x03),
POWERSTEP01_SPEED        = ((uint8_t)0x04),
          POWERSTEP01_ACC          = ((uint8_t)0x05),
          POWERSTEP01_DEC          = ((uint8_t)0x06),
          POWERSTEP01_MAX_SPEED    = ((uint8_t)0x07),
          POWERSTEP01_MIN_SPEED    = ((uint8_t)0x08),
          POWERSTEP01_FS_SPD       = ((uint8_t)0x15),
          POWERSTEP01_KVAL_HOLD    = ((uint8_t)0x09),
          POWERSTEP01_KVAL_RUN     = ((uint8_t)0x0A),
          POWERSTEP01_KVAL_ACC     = ((uint8_t)0x0B),
          POWERSTEP01_KVAL_DEC     = ((uint8_t)0x0C),
          POWERSTEP01_INT_SPD      = ((uint8_t)0x0D),
          POWERSTEP01_ST_SLP       = ((uint8_t)0x0E),
          POWERSTEP01_FN_SLP_ACC   = ((uint8_t)0x0F),
          POWERSTEP01_FN_SLP_DEC   = ((uint8_t)0x10),
          POWERSTEP01_K_THERM      = ((uint8_t)0x11),
          POWERSTEP01_ADC_OUT      = ((uint8_t)0x12),
          POWERSTEP01_OCD_TH       = ((uint8_t)0x13),
          POWERSTEP01_STALL_TH     = ((uint8_t)0x14),
          POWERSTEP01_STEP_MODE    = ((uint8_t)0x16),
          POWERSTEP01_ALARM_EN     = ((uint8_t)0x17),
          POWERSTEP01_GATECFG1     = ((uint8_t)0x18),
          POWERSTEP01_GATECFG2     = ((uint8_t)0x19),
          POWERSTEP01_CONFIG       = ((uint8_t)0x1A),
          POWERSTEP01_STATUS       = ((uint8_t)0x1B)
} powerstep01_Registers_t;
```

Return values

Register value

Example

```
#include "main.h"
int main(void)
{
   uint32_t regValue;
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Read ACC register of device 0*/
  regValue = BSP_MotorControl_CmdGetParam(0, POWERSTEP01_ACC);
}
```

2.5.8 BSP_MotorControl_CmdGetStatus()

`uint16_t BSP_MotorControl_CmdGetStatus ( uint8_t deviceId)`

Description

Issues the GetStatus command to the powerSTEP01 specified device.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)

Return values

Status	value of the STATUS register

Note: Once the GetStatus command is performed, the flags of the status register are reset. This is not the case when the status register is read with the GetParam command (via the functions BSP_MotorControl_ReadStatusRegister or BSP_MotorControl_CmdGetParam).

Example

```
#include "main.h"
void MyFlagInterruptHandler(void);
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Attach the function MyFlagInterruptHandler (defined below) to the flag
interrupt */
  BSP_MotorControl_AttachFlagInterrupt(MyFlagInterruptHandler);
}
void MyFlagInterruptHandler(void)
{
  /* Get the value of the status register via the powerSTEP01 command
GET_STATUS */
  uint16_t statusRegister = BSP_MotorControl_CmdGetStatus(0);
}
```

2.5.9 BSP_MotorControl_CmdGoToDir()

`void BSP_MotorControl_CmdGoToDir ( uint8_t deviceId,`

```
             motorDir_t dir,
             int32_t targetPosition
           )
```

Description

Requests the motor to move to the specified position in the specified direction.

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

– Accelerating phase using the device acceleration
– Steady phase where the motor turns at max speed
– Decelerating phase using the device deceleration

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

– Accelerating phase using the device acceleration
– Decelerating phase using the device deceleration

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)
`[in] dir`	movement direction
`[in] targetPosition`	absolute position in steps

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to go to position -200 in the FORWARD direction */
  BSP_MotorControl_CmdGoToDir(0, FORWARD, -200);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
}
```

2.5.10 BSP_MotorControl_CmdGoUntil()

```
void BSP_MotorControl_ CmdGoUntil ( uint8_t deviceId,
             motorAction_t action,
             motorDir_t dir,
             uint32_t speed
           )
```

Description

Issues powerstep01 GoUntil command: it produces a motion in the specified direction at the specified speed until an input falling edge occurs on the SW pin of the specified device, triggering execution of the action and then an automatic soft stop of the specified device.

In case of ACTION_RESET, the ABS_POS register is reset. In case of ACTION_COPY, the ABS_POS register value is copied into the MARK register.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)
`[in] action`	ACTION_RESET or ACTION_COPY
`[in] dir`	FORWARD or BACKWARD
`[in] speed`	speed in steps/s

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to move at 200 steps/s in BACKWARD direction until an
input falling edge occurs on the SW pin of the device 0 */
  BSP_MotorControl_CmdGoUntil(0, ACTION_COPY, BACKWARD, 200);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
}
```

2.5.11 BSP_MotorControl_CmdHardHiZ()

```
void BSP_MotorControl_CmdHardHiZ ( uint8_t deviceId)
```

Description

Immediately stops the motor and disables the power bridge.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)

Return values

None

Example

```
#include "main.h"
int main(void)
```

```
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to run in FORWARD direction at 200 steps/s */
  BSP_MotorControl_CmdRun(0, FORWARD, 200);
/* Request device 0 to stop immediately and disable the power bridges */
  BSP_MotorControl_CmdHardHiZ(0);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
}
```

2.5.12 BSP_MotorControl_CmdNop()

```
void BSP_MotorControl_CmdNop ( uint8_t deviceId)
```

Description

Issues the Nop command to the powerSTEP01 specified device.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Send Nop command to powerSTEP01 device 0*/
  BSP_MotorControl_CmdNop(0);
}
```

2.5.13 BSP_MotorControl_CmdReleaseSw()

```
void BSP_MotorControl_ CmdReleaseSw ( uint8_t deviceId,
             motorAction_t action,
             motorDir_t dir
          )
```

Description

Issues powerstep01 ReleaseSW command: it produces a motion in the specified direction at the minimum speed until SW rising edge occurs on the SW pin of the specified device, triggering execution of the action and then an automatic hard stop of the specified device.

In case of ACTION_RESET, the ABS_POS register is reset. In case of ACTION_COPY, the ABS_POS register value is copied into the MARK register.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)
`[in] action`	ACTION_RESET or ACTION_COPY
`[in] dir`	FORWARD or BACKWARD

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to move at minimum speed in BACKWARD direction until a
rising edge occurs on the SW pin of the device 0 */
  BSP_MotorControl_CmdReleaseSw(0, ACTION_COPY, BACKWARD, 200);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
}
```

2.5.14 BSP_MotorControl_CmdResetDevice()

```
void BSP_MotorControl_CmdResetDevice ( uint8_t deviceId)
```

Description

Issues ResetDevice command: resets the device to power-up conditions.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)

Return values

None

Example

```
#include "main.h"
```

```
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to run in FORWARD direction at 200 steps/s */
  BSP_MotorControl_CmdRun(0, FORWARD, 200);
/* Request device 0 to make an hard stop */
  BSP_MotorControl_HardStop(0);
/* Resets the powerSTEP01 device to power-up conditions*/
  BSP_MotorControl_CmdResetDevice(0);
}
```

2.5.15 BSP_MotorControl_CmdResetPos()

```
void BSP_MotorControl_CmdResetPos ( uint8_t deviceId)
```

Description

Issues ResetPos command: resets the ABS_POS register to zero. The zero position is also defined as the HOME position.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to move of 200 steps in BACKWARD direction */
  BSP_MotorControl_Move(0, BACKWARD, 200);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
/* Resets the ABS_POS register to zero */
  BSP_MotorControl_CmdResetPos(0);
}
```

2.5.16 BSP_MotorControl_CmdRun()

```
void BSP_MotorControl_ CmdRun ( uint8_t deviceId,
             motorDir_t dir,
             uint32_t speed
          )
```

Description

Issues powerstep01 Run command: it produces a motion in the specified direction, accelerating from the min speed up to the specified speed by using the device acceleration.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)
`[in] dir`	FORWARD or BACKWARD
`[in] speed`	speed in steps/s

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to move at 200 steps/s in BACKWARD direction */
  BSP_MotorControl_CmdRun(0, BACKWARD, 200);
}
```

2.5.17 BSP_MotorControl_CmdSetParam()

```
void BSP_MotorControl_CmdSetParam ( uint8_t deviceId,
                uint32_t param,
                uint32_t value
             )
```

Description

Issues the SetParam command to the powerSTEP01 specified device.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)
`[in] param`	Register address of the powerSTEP01 from powerstep01_Registers_t enum:

```
typedef enum {
          POWERSTEP01_ABS_POS      = ((uint8_t)0x01),
          POWERSTEP01_EL_POS       = ((uint8_t)0x02),
          POWERSTEP01_MARK         = ((uint8_t)0x03),
          POWERSTEP01_SPEED        = ((uint8_t)0x04),
          POWERSTEP01_ACC          = ((uint8_t)0x05),
          POWERSTEP01_DEC          = ((uint8_t)0x06),
          POWERSTEP01_MAX_SPEED    = ((uint8_t)0x07),
          POWERSTEP01_MIN_SPEED    = ((uint8_t)0x08),
          POWERSTEP01_FS_SPD       = ((uint8_t)0x15),
          POWERSTEP01_KVAL_HOLD    = ((uint8_t)0x09),
          POWERSTEP01_KVAL_RUN     = ((uint8_t)0x0A),
          POWERSTEP01_KVAL_ACC     = ((uint8_t)0x0B),
          POWERSTEP01_KVAL_DEC     = ((uint8_t)0x0C),
          POWERSTEP01_INT_SPD      = ((uint8_t)0x0D),
          POWERSTEP01_ST_SLP       = ((uint8_t)0x0E),
          POWERSTEP01_FN_SLP_ACC   = ((uint8_t)0x0F),
          POWERSTEP01_FN_SLP_DEC   = ((uint8_t)0x10),
          POWERSTEP01_K_THERM      = ((uint8_t)0x11),
          POWERSTEP01_ADC_OUT      = ((uint8_t)0x12),
          POWERSTEP01_OCD_TH       = ((uint8_t)0x13),
          POWERSTEP01_STALL_TH     = ((uint8_t)0x14),
          POWERSTEP01_STEP_MODE    = ((uint8_t)0x16),
          POWERSTEP01_ALARM_EN     = ((uint8_t)0x17),
          POWERSTEP01_GATECFG1     = ((uint8_t)0x18),
          POWERSTEP01_GATECFG2     = ((uint8_t)0x19),
          POWERSTEP01_CONFIG       = ((uint8_t)0x1A),
          POWERSTEP01_STATUS       = ((uint8_t)0x1B)
} powerstep01_Registers_t;
```

`[in] value`	Value to set in the register

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Set ALARM_EN register of device 0 to predefined value of device 1 */
```

```
 BSP_MotorControl_CmdSetParam(0, POWERSTEP01_ALARM_EN,
                          POWERSTEP01_CONF_PARAM_ALARM_EN_DEVICE_1);
}
```

2.5.18 BSP_MotorControl_CmdSoftHiZ()

`void BSP_MotorControl_CmdSoftHiZ ( uint8_t deviceId)`

Description

Issues SoftHiZ command: it causes the motor to decelerate with the device programmed deceleration value until the MIN_SPEED value is reached and then forces the bridges into high impedance state (no holding torque is present).

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to run in FORWARD direction at 200 steps/s */
  BSP_MotorControl_CmdRun(0, FORWARD, 200);
/* Request device 0 to stop smoothly and disable the power bridges */
  BSP_MotorControl_CmdSoftHiZ(0);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
}
```

2.5.19 BSP_MotorControl_CmdSoftStop()

`void BSP_MotorControl_CmdSoftStop ( uint8_t deviceId)`

Description

Issues SoftStop command: it causes the motor to decelerate with the device programmed deceleration value until the MIN_SPEED value is reached and then lets the bridges enable.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to run in FORWARD direction at 200 steps/s */
  BSP_MotorControl_CmdRun(0, FORWARD, 200);
/* Request device 0 to stop smoothly and keep the power bridges enabled */
  BSP_MotorControl_CmdSoftStop(0);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
}
```

2.5.20 BSP_MotorControl_CmdStepClock()

```
void BSP_MotorControl_ CmdStepClock ( uint8_t deviceId,
             motorDir_t dir
          )
```

Description

Issues powerstep01 StepClock command: The StepClock command switches the device to step-clock mode and imposes the forward or backward direction.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)
`[in] dir`	FORWARD or BACKWARD

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to enter step clock mode and impose backward direction
*/
  BSP_MotorControl_CmdStepClock(0, BACKWARD);
}
```

2.5.21 BSP_MotorControl_ErrorHandler()

`void BSP_MotorControl_ErrorHandler(uint16_t error)`

Description

Allows an external call of the motor control error callback, provided it has been previously attached. Otherwise, the function will loop forever.

Parameters

`[in] error`	Number of the raised error

Return values

None

Example

```
#include "main.h"
static void Error_Handler(uint16_t error);
int main(void)
{
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 2 devices */
  /* The powerSTEP01 registers are set with the predefined values */
  /* from file powerstep01_target_config.h*/
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01,2);
/* Attach the function Error_Handler (defined below) to the error Handler*/
  BSP_MotorControl_AttachErrorHandler(Error_Handler);
…
/* Call Error_Handler with error number 100 */
 BSP_MotorControl_ErrorHandler(100);
}
/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
static void Error_Handler(uint16_t error)
{
  /* Backup error number */
  gLastError = error;
/* Infinite loop */
  while(1)
  {
  }
}
```

2.5.22 BSP_MotorControl_FetchAndClearAllStatus()

`void BSP_MotorControl_FetchAndClearAllStatus( void )`

Description

Fetch and clear status flags of all devices by issuing a GET_STATUS command simultaneously to all devices. Then, the fetched status of each device can be retrieved by using the BSP_MotorControl_GetFetchedStatus function, provided there are no other calls to functions which use the SPI in between.

Parameters

None

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 3 devices */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 3);
/* Fetch status of the 3 devices */
  BSP_MotorControl_FetchAndClearAllStatus ();
}
```

2.5.23 BSP_MotorControl_FlagInterruptHandler()

`void BSP_MotorControl_ FlagInterruptHandler(void )`

Description

Handler of the flag interrupt which calls the user callback (if defined).

This function is supposed to be called from the external line callback which it is associated to the Flag interrupt.

Parameters

None

Return values

None

Example

```
#include "main.h"
static void MyFlagInterruptHandler(void);
int main(void)
```

```
{
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Attach the function MyFlagInterruptHandler (defined below) to the flag
interrupt */
  BSP_MotorControl_AttachFlagInterrupt(MyFlagInterruptHandler);
…
}
/**
  * @brief  This function is the User handler for the flag interrupt
  * @param  None
  * @retval None
  */
void MyFlagInterruptHandler(void)
{
  /* Get the value of the status register via the powerSTEP01 command
GET_STATUS: this will clear the flags */
  uint16_t statusRegister = BSP_MotorControl_CmdGetStatus(0);
  }
Then in file: stm32f4xx_hal_msp.c or stm32f0xx_hal_msp.c or
stm32l0xx_hal_msp.c:
/**
  * @brief External Line Callback
  * @param[in] GPIO_Pin pin number
  * @retval None
  */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if (GPIO_Pin == BSP_MOTOR_CONTROL_BOARD_FLAG_PIN)
  {
    BSP_MotorControl_FlagInterruptHandler();
  }
 }
```

2.5.24 BSP_MotorControl_GetBoardId()

`uint16_t BSP_MotorControl_GetBoardId( void )`

Description

Returns the motor driver board ID

Parameters

None

Return values

`MotorControlBoardId`	BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01 for X-CUBE-SPN3

Example

```
#include "main.h"
int main(void)
{
  uint16_t boardId;
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 3 devices */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01,3);
    …
  /* Get the motor driver board Id */
  boardId = BSP_MotorControl_GetBoardId();
}
```

2.5.25 BSP_MotorControl_GetFetchedStatus()

`Uint16_t BSP_MotorControl_GetFetchedStatus ( uint8_t deviceId)`

Description

Gets the value of the STATUS register which was fetched by using BSP_MotorControl_FetchAndClearAllStatus. The fetched values are available as long as there are no other calls to functions which use the SPI.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)

Return values

STATUS register value

Example

```
#include "main.h"
int main(void)
{
  uint16_t status;
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 3 devices */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 3);
/* Request device 1 to run in FORWARD direction at 200 steps/s */
  BSP_MotorControl_CmdRun(1, FORWARD, 200);
/* Fetch status of the 3 devices */
```

```
  BSP_MotorControl_FetchAndClearAllStatus();
/* Get the status of device 0*/
  status = BSP_MotorControl_GetFetchedStatus(0);
/* Get the status of device 1*/
  status = BSP_MotorControl_GetFetchedStatus(1);
/* Get the status of device 2*/
  status = BSP_MotorControl_GetFetchedStatus(2);
}
```

2.5.26 BSP_MotorControl_GetFwVersion()

`uint8_t BSP_MotorControl_GetFwVersion ( void )`

Description

Returns the FW version of the library.

Parameters

None

Return values

Current FW version

Example

```
#include "main.h"
int main(void)
{
  Uint8_t myFwVersion;
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Get FW version of the library*/
  myFwVersion = BSP_MotorControl_GetFwVersion();
}
```

2.5.27 BSP_MotorControl_GetMark()

`int32_t BSP_MotorControl_GetMark ( uint8_t deviceId)`

Description

Returns the mark position of the specified device.

Parameters

`[in] deviceId`	Id of the device (from 0 to 2)

Return values

`Mark`	Register value converted in a 32b signed integer

Example

```
#include "main.h"
int main(void)
{
  int32_t pos;
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to run in FORWARD direction at 200 steps/s */
  BSP_MotorControl_CmdRun(0, FORWARD, 200);
/* Wait for 5 seconds */
  HAL_Delay(5000);
/* Request device 0 to make a soft stop */
  BSP_MotorControl_CmdSoftStop(0);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
/* Set current position to become the Mark position of device 0*/
  BSP_MotorControl_SetMark(0);
/* Get the Mark position of device 0*/
  pos = BSP_MotorControl_GetMark(0);
}
```

2.5.28 BSP_MotorControl_GetNbDevices()

`uint8_t BSP_MotorControl_GetNbDevices ( void )`

Description

Returns the number of devices in the daisy chain.

Parameters

None

Return values

Number of devices in the daisy chain

Example

```
#include "main.h"
int main(void)
{
  Uint8_t numberOfDevices;
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 3 devices */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 3);
/* Get the number of devices in the daisy chain */
  numberOfDevices = BSP_MotorControl_GetNbDevices();
}
```

2.5.29 BSP_MotorControl_GetPosition()

`int32_t BSP_MotorControl_GetPosition ( uint8_t deviceId)`

Description

Returns the ABS_POSITION of the specified device.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)

Return values

`ABS_POSITION`	register value converted in a 32b signed integer

Example

```
#include "main.h"
int main(void)
{
  int32_t pos;
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to run in FORWARD direction at 200 steps/s */
  BSP_MotorControl_CmdRun(0, FORWARD, 200);
/* Wait for 5 seconds */
  HAL_Delay(5000);
/* Request device 0 to make a soft stop */
  BSP_MotorControl_CmdSoftStop(0);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
/* Get the position of device 0*/
  pos = BSP_MotorControl_GetPosition (0);
}
```

2.5.30 BSP_MotorControl_GoHome()

`void BSP_MotorControl_GoHome ( uint8_t deviceId)`

Description

Requests the motor to move to the home position (ABS_POSITION = 0)

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Set home position for device device 0*/
  BSP_MotorControl_SetHome(0);
/* Request device 0 to run in FORWARD direction at 200 steps/s */
  BSP_MotorControl_CmdRun(0, FORWARD, 200);
/* Wait for 5 seconds */
  HAL_Delay(5000);
/* Request device 0 to make a soft stop */
  BSP_MotorControl_CmdSoftStop(0);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
/* Request device 0 to go to home */
  BSP_MotorControl_GoHome(0);
}
```

2.5.31 BSP_MotorControl_GoMark()

`void BSP_MotorControl_GoMark ( uint8_t deviceId)`

Description

Requests the motor to move to the mark position.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to go to position 200 */
  BSP_MotorControl_GoTo(0, 200);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
/* Set current position to become the Mark position of device 0*/
  BSP_MotorControl_SetMark(0);
/* Request device 0 to run in FORWARD direction at 200 steps/s */
  BSP_MotorControl_CmdRun(0, FORWARD, 200);
/* Wait for 5 seconds */
  HAL_Delay(5000);
/* Request device 0 to make a soft stop */
  BSP_MotorControl_CmdSoftStop(0);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
/* Request device 0 to come back to its Mark position */
  BSP_MotorControl_GoMark(0);
}
```

2.5.32 BSP_MotorControl_GoTo()

```
void BSP_MotorControl_GoTo ( uint8_t deviceId,
             int32_t targetPosition
           )
```

Description

Requests the motor to move to the specified position.

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

- Accelerating phase using the device acceleration
- Steady phase where the motor turns at maximum speed
- Decelerating phase using the device deceleration

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

- Accelerating phase using the device acceleration
- Decelerating phase using the device deceleration

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)
`[in] targetPosition`	absolute position in steps

Return values

None

Example

```
#include "main.h"
int main(void)
```

```
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to go to position -200 */
  BSP_MotorControl_GoTo(0, -200);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
}
```

2.5.33 BSP_MotorControl_HardStop()

`void BSP_MotorControl_HardStop ( uint8_t deviceId)`

Description

Immediately stops the motor. The power stage remains enabled.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)
`[in] targetPosition`	absolute position in steps

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to run in FORWARD direction at 200 steps/s */
  BSP_MotorControl_CmdRun(0, FORWARD, 200);
/* Request device 0 to make an hard stop */
  BSP_MotorControl_HardStop(0);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
}
```

2.5.34 BSP_MotorControl_Init()

`void BSP_MotorControl_Init( uint16_t  id, uint8_t nbDevices)`

Description

Starts the motor control library.

The powerSTEP01 registers will be set to the predefined values from the file powerstep01_target_config.h

Parameters

`[in] id`	Motor driver board ID: BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01 for X-NUCLEO-IHM03A1
`[in] nbDevices`	Number of Powerstep01 devices to use (from 1 to MAX_NUMBER_OF_DEVICES)

Return values

None

Example

```
#include "main.h"
int main(void)
{
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 3 devices */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01,3);
}
```

2.5.35 BSP_MotorControl_IsDeviceBusy()

`bool BSP_MotorControl_IsDeviceBusy ( uint8_t deviceId)`

Description

Checks if the specified device is busy by reading the Busy flag bit of its status register.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)

Return values

True if device is busy, otherwise false

Example

```
#include "main.h"
int main(void)
{
  bool busy;
/* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 3 devices */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 3);
/* Request device 1 to run in FORWARD direction at 200 steps/s */
  BSP_MotorControl_CmdRun(1, FORWARD, 200);
/* Check if device 0 is busy */
  busy = BSP_MotorControl_IsDeviceBusy(0);
/* Check if device 1 is busy */
  busy = BSP_MotorControl_IsDeviceBusy(1);
/* Check if device 2 is busy */
  busy = BSP_MotorControl_IsDeviceBusy(2);
}
```

2.5.36 BSP_MotorControl_Move()

```
void BSP_MotorControl_Move ( uint8_t deviceId,
             motorDir_t direction,
             uint32_t stepCount
           )
```

Description

Moves the motor of the specified number of steps.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)
`[in] direction`	FORWARD or BACKWARD
`[in] stepCount`	Number of steps to perform

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to move of 200 steps in BACKWARD direction */
  BSP_MotorControl_Move(0, BACKWARD, 200);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
}
```

2.5.37 BSP_MotorControl_QueueCommands()

```
void BSP_MotorControl_QueueCommands ( uint8_t deviceId,
                uint8_t param,
                int32_t value
             )
```

Description

Put commands in queue before synchronous sending done by calling BSP_MotorControl_SendQueuedCommands. Any call to functions that use the SPI between the calls of BSP_MotorControl_QueueCommands and BSP_MotorControl_SendQueuedCommands will corrupt the queue. A command for each device in the daisy chain must be specified before calling BSP_MotorControl_SendQueuedCommands.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)
`[in] param`	Command to queue: all powerstep01 commands from powerstep01_Commands_t enum except POWERSTEP01_SET_PARAM, POWERSTEP01_GET_PARAM, POWERSTEP01_GET_STATUS

```
typedef enum {
  POWERSTEP01_NOP                = ((uint8_t)0x00),
  POWERSTEP01_SET_PARAM          = ((uint8_t)0x00),
  POWERSTEP01_GET_PARAM          = ((uint8_t)0x20),
  POWERSTEP01_RUN                = ((uint8_t)0x50),
  POWERSTEP01_STEP_CLOCK         = ((uint8_t)0x58),
  POWERSTEP01_MOVE               = ((uint8_t)0x40),
  POWERSTEP01_GO_TO              = ((uint8_t)0x60),
  POWERSTEP01_GO_TO_DIR          = ((uint8_t)0x68),
  POWERSTEP01_GO_UNTIL           = ((uint8_t)0x82),
  POWERSTEP01_GO_UNTIL_ACT_CPY   = ((uint8_t)0x8A),
  POWERSTEP01_RELEASE_SW         = ((uint8_t)0x92),
  POWERSTEP01_GO_HOME            = ((uint8_t)0x70),
  POWERSTEP01_GO_MARK            = ((uint8_t)0x78),
  POWERSTEP01_RESET_POS          = ((uint8_t)0xD8),
  POWERSTEP01_RESET_DEVICE       = ((uint8_t)0xC0),
  POWERSTEP01_SOFT_STOP          = ((uint8_t)0xB0),
  POWERSTEP01_HARD_STOP          = ((uint8_t)0xB8),
  POWERSTEP01_SOFT_HIZ           = ((uint8_t)0xA0),
  POWERSTEP01_HARD_HIZ           = ((uint8_t)0xA8),
  POWERSTEP01_GET_STATUS         = ((uint8_t)0xD0),
  POWERSTEP01_RESERVED_CMD1      = ((uint8_t)0xEB),
  POWERSTEP01_RESERVED_CMD2      = ((uint8_t)0xF8)
} powerstep01_Commands_t;
```

`[in] value`	Argument of the command to queue

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 3 devices */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 3);
/* Queue command for device 0 to go to absolute position 8000 using the
shortest path */
  BSP_MotorControl_QueueCommands(0, POWERSTEP01_GO_TO, 8000);
/* Queue command for device 1 to do nothing */
  BSP_MotorControl_QueueCommands(1, POWERSTEP01_NOP, 0);
/* Queue command for device 2 to go in the forward direction to absolute
position 10000 */
  BSP_MotorControl_QueueCommands(2, (uint8_t)POWERSTEP01_GO_TO_DIR |
(uint8_t)FORWARD, 10000);
}
```

2.5.38 BSP_MotorControl_ReleaseReset()

```
void BSP_MotorControl_ReleaseReset ( void )
```

Description

Releases the powerstep01 reset (pin set to high) of all devices.

Parameters

None

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 2 devices */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01,2);
/* Release powerSTEP01 reset of all devices */
  BSP_MotorControl_ReleaseReset();
}
```

2.5.39 BSP_MotorControl_Reset()

```
void BSP_MotorControl_Reset ( void )
```

Description

Resets the powerstep01 (reset pin set to low) of all devices.

Parameters

None

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 2 devices */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01,2);
/* powerSTEP01 reset pin set to low for all devices */
  BSP_MotorControl_Reset();
}
```

2.5.40 BSP_MotorControl_SelectStepMode()

```
void BSP_MotorControl_SelectStepMode ( uint8_t deviceId,
             motorStepMode_t stepMode
                )
```

Description

Sets the stepping mode.

Parameters

`[in] deviceId`	Id of the device (from 0 to 2)
`[in] stepMode`	from full step to 1/128 microstep as specified in enum motorStepMode_t

```
motorStepMode_t:
   typedef enum {
      STEP_MODE_FULL   = ((uint8_t)0x00), //full step
      STEP _MODE_HALF   = ((uint8_t)0x01),
      STEP _MODE_1_4    = ((uint8_t)0x02),
      STEP _MODE_1_8    = ((uint8_t)0x03),
      STEP _MODE_1_16   = ((uint8_t)0x04),
      STEP _MODE_1_32   = ((uint8_t)0x05),
      STEP _MODE_1_64   = ((uint8_t)0x06),
      STEP _MODE_1_128  = ((uint8_t)0x07)}
motorStepMode_t;
```

Return values

None

Note: The acceleration, deceleration, min and max speed do not scale with the stepping mode and thus there is no need to change them when the stepping mode is changed. The ABS_POS register is automatically reset to 0 when changing the stepping mode. The MARK register value becomes inconsistent.

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Select 1/16 microstepping mode for device 0 */
  BSP_MotorControl_SelectStepMode(0, STEP _MODE_1_16);
/* Request device 0 to go position 3200 */
  BSP_MotorControl_GoTo(0,3200);
}
```

2.5.41 BSP_MotorControl_SendQueuedCommands()

`void BSP_MotorControl_SendQueuedCommands( void )`

Description

Sets the stepping mode.

Parameters

None

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 3 devices */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 3);
/* Queue command for device 0 to go to absolute position 8000 using the
shortest path */
  BSP_MotorControl_QueueCommands(0, POWERSTEP01_GO_TO, 8000);
/* Queue command for device 1 to do nothing */
  BSP_MotorControl_QueueCommands(1, POWERSTEP01_NOP, 0);
/* Queue command for device 2 to go in the forward direction to absolute
position 10000 */
  BSP_MotorControl_QueueCommands(2, (uint8_t)POWERSTEP01_GO_TO_DIR |
(uint8_t)FORWARD, 10000);
/* Send commands in the queue to execute */
  BSP_MotorControl_SendQueuedCommands();
}
```

2.5.42 BSP_MotorControl_SetHome()

`void BSP_MotorControl_SetHome ( uint8_t deviceId)`

Description

Sets current position to be the Home position (ABS pos set to 0).

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Set home position for device device 0*/
  BSP_MotorControl_SetHome(0);
/* Request device 0 to run in FORWARD direction at 200 steps/s */
  BSP_MotorControl_CmdRun(0, FORWARD, 200);
/* Wait for 5 seconds */
  HAL_Delay(5000);
/* Request device 0 to make a soft stop */
  BSP_MotorControl_CmdSoftStop(0);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
/* Request device 0 to go to home */
  BSP_MotorControl_GoHome(0);
}
```

2.5.43 BSP_MotorControl_SetMark()

`void BSP_MotorControl_SetMark ( uint8_t deviceId)`

Description

Sets current position to be the Mark position.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to go to position 200 */
  BSP_MotorControl_GoTo(0, 200);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
/* Set current position to become the Mark position of device 0*/
  BSP_MotorControl_SetMark(0);
/* Request device 0 to run in FORWARD direction at 200 steps/s */
  BSP_MotorControl_CmdRun(0, FORWARD, 200);
/* Wait for 5 seconds */
  HAL_Delay(5000);
/* Request device 0 to make a soft stop */
```

```
  BSP_MotorControl_CmdSoftStop(0);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
/* Request device 0 to come back to its Mark position */
  BSP_MotorControl_GoMark(0);
}
```

2.5.44 BSP_MotorControl_StartStepClock()

`void BSP_MotorControl_ StartStepClock ( uint16_t newFreq)`

Description

Starts the step clock by using the given frequency. The frequency is the speed of the device.

Parameters

`[in] newFreq`	New frequency in Hz of the step clock.

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to enter step clock mode and impose backward direction
*/
  BSP_MotorControl_CmdStepClock(0, BACKWARD);
/* Wait for 1 second */
  HAL_Delay(1000);
/* Enable the step clock at 333 Hz */
  BSP_MotorControl_StartStepClock(333);
}
```

2.5.45 BSP_MotorControl_StopStepClock()

`void BSP_MotorControl_StopStepClock ( void )`

Description

Stop the step clock.

Parameters

None

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to enter step clock mode and impose backward direction
*/
  BSP_MotorControl_CmdStepClock(0, BACKWARD);
/* Wait for 1 second */
  HAL_Delay(1000);
/* Enable the step clock at 333 Hz */
  BSP_MotorControl_StartStepClock(333);
/* Let the motor runs for 5 second at 333 step/s */
  HAL_Delay(5000);
/* Stop the step clock */
  BSP_MotorControl_StopStepClock();
}
```

2.5.46 BSP_MotorControl_WaitForAllDevicesNotBusy()

```
void BSP_MotorControl_WaitForAllDevicesNotBusy( void )
```

Description

Locks until all devices are no longer busy.

Parameters

None

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 3 devices */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 3);
/* Queue command for device 0 to go to absolute position 8000 using the
shortest path */
  BSP_MotorControl_QueueCommands(0, POWERSTEP01_GO_TO, 8000);
/* Queue command for device 1 to do nothing */
  BSP_MotorControl_QueueCommands(0, POWERSTEP01_NOP, 0);
/* Queue command for device 2 to go in the forward direction to absolute
position 10000 */
  BSP_MotorControl_QueueCommands(2, (uint8_t)POWERSTEP01_GO_TO_DIR |
(uint8_t)FORWARD, 10000);
/* Send commands in the queue to execute */
  BSP_MotorControl_SendQueuedCommands();
/* Wait for all devices not busy */
  BSP_MotorControl_WaitForAllDevicesNotBusy();
}
```

2.5.47 BSP_MotorControl_WaitWhileActive()

```
void BSP_MotorControl_WaitWhileActive ( uint8_t deviceId)
```

Description

Locks until the device state becomes Inactive. The use of this function is particularly useful if the user waits for the stop of a given device before sending it a new moving command (GoTo, GoMark, GoHome, Move, Run). Without using a WaitWhileActive, the last moving command will be executed before the completion of the previous one.

Parameters

`[in] deviceId`	Id of the device (from 0 to MAX_NUMBER_OF_DEVICES - 1)

Return values

None

Example

```
#include "main.h"
int main(void)
{
  /* STM32 HAL library initialization */
  HAL_Init();
/* Configure the system clock */
  SystemClock_Config();
/* Start the motor control library to use 1 device */
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_POWERSTEP01, 1);
/* Request device 0 to go to position -200 */
  BSP_MotorControl_GoTo(0, -200);
/* Wait for device 0 end moving */
  /* Without this command, the GoTo -200 will be interrupted */
  /* by the GoTo 200 */
  BSP_MotorControl_WaitWhileActive(0);
/* Request device 0 to go to position 200 */
  BSP_MotorControl_GoTo(0, 200);
/* Wait for device 0 end moving */
  BSP_MotorControl_WaitWhileActive(0);
}
```

2.6 Sample application description

Several example applications using the X-NUCLEO-IHM03A1 expansion board with either NUCLEO-F401RE, NUCLEO-F030R8 or NUCLEO-L053R8 boards are provided in the “Projects” directory. Ready-to-be-built projects are available for multiple IDEs (see *Section 2.3.2* for details).

3 System setup guide

3.1 Hardware description

This section describes the hardware components needed for developing a stepper motor driver-based application.

The following subsections describe the individual components.

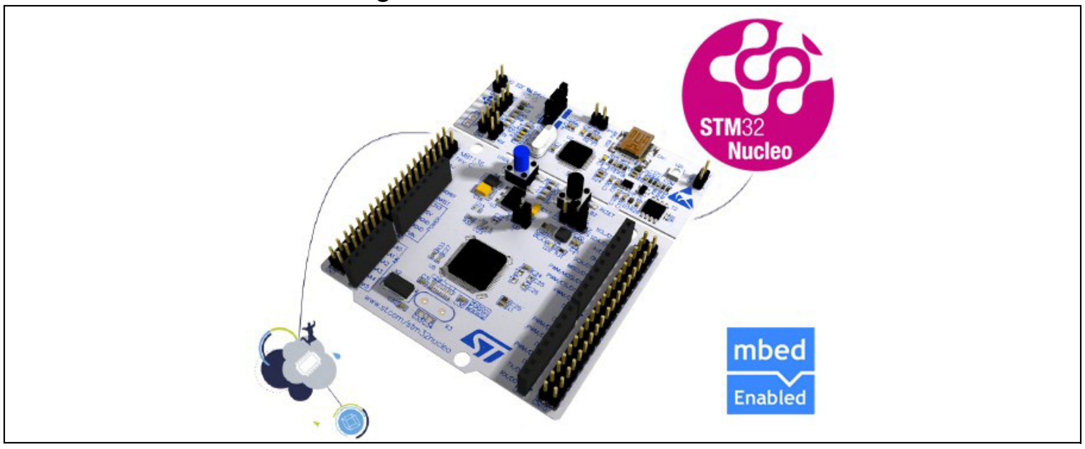
3.1.1 STM32 Nucleo platform

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 probe 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 at:

http://www.st.com/stm32nucleo

Figure 3. STM32 Nucleo board

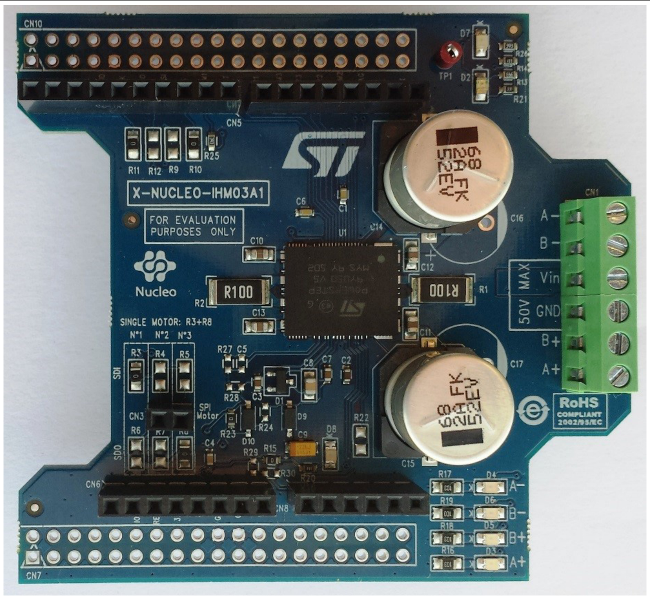


3.1.2 X-NUCLEO-IHM03A1 expansion board

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

The X-NUCLEO-IHM03A1 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 4. X-NUCLEO-IHM03A1 high power stepper motor driver expansion board for STM32 Nucleo



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

3.1.3 Miscellaneous HW components

To complete the HW setup, you will need:

- From 1 to 3 stepper motors
- an external DC power supply with 2 electrical cables for X-NUCLEO-IHM03A1 board
- a USB cable type A to mini-B to connect the 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-SPN3: an expansion for STM32Cube dedicated to powerSTEP01 motor driver application development. The X-CUBE-SPN3 firmware and related documentation is available on st.com.
- A development tool-chain and compiler. Three tool-chains are supported:
  - Keil RealView Microcontroller Development Kit (MDK-ARM) toolchain V5.12
  - IAR Embedded Workbench for ARM (EWARM) toolchain V7.20
  - Atollic TrueSTUDIO® for ARM® Pro V5.1.1

3.3 Hardware and software setup

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

3.3.1 Common setup to drive 1, 2 or 3 motors

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

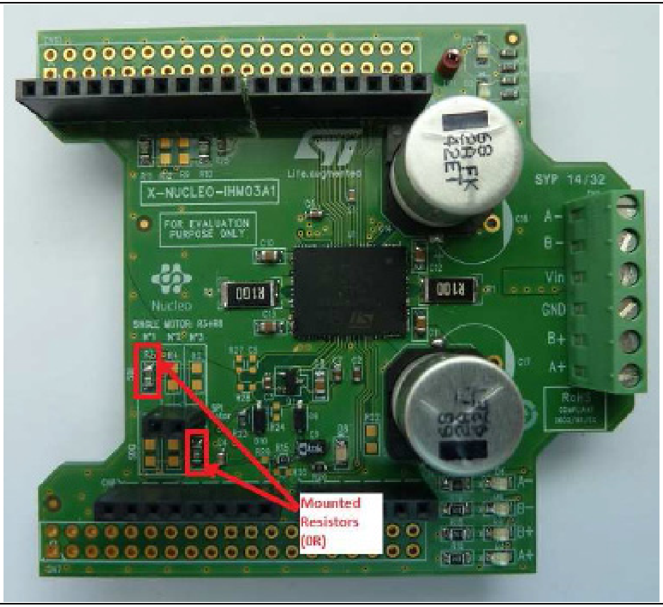
- JP1 off
- JP5 (PWR) on UV5 side
- JP6 (IDD) on

3.3.2 Setup to drive 1 motor

The X-NUCLEO-IHM03A1 expansion board must have:

- Mounted resistors (0R) on R3 and R8
- Unmounted resistors on R4, R5, R6, and R7

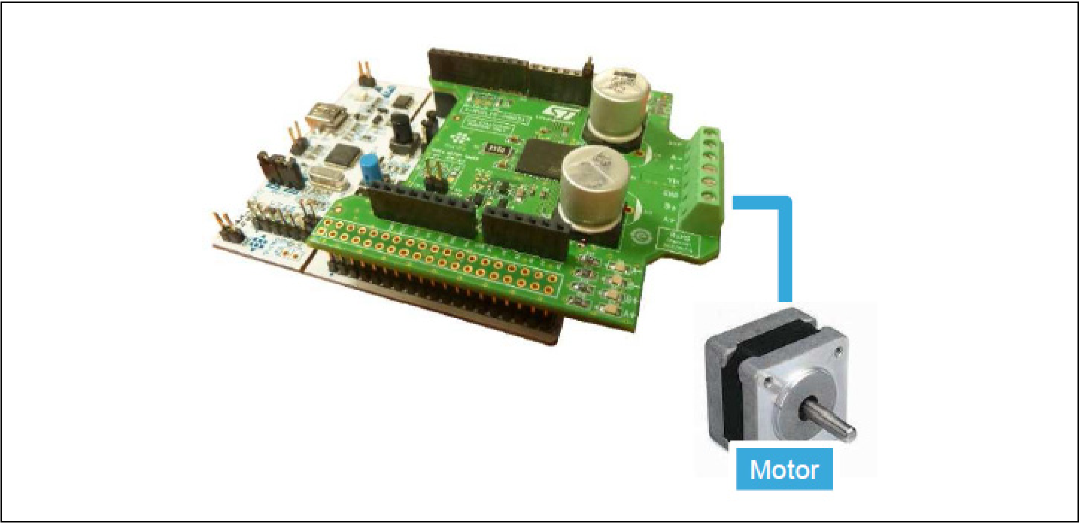
Figure 5. X-NUCLEO-IHM03A1 high power stepper motor driver configuration to drive 1 motor



Once the boards are properly configured:

- Plug the X-NUCLEO-IHM03A1 expansion board on top of the STM32 Nucleo by using 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 on the X-NUCLEO-IHM03A1 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 by the stepper motor.
- Connect the stepper motor to the X-NUCLEO-IHM03A1 bridge connectors A+/- and B+/-

Figure 6. Board connections to drive 1 motor



Once the system setup is ready:

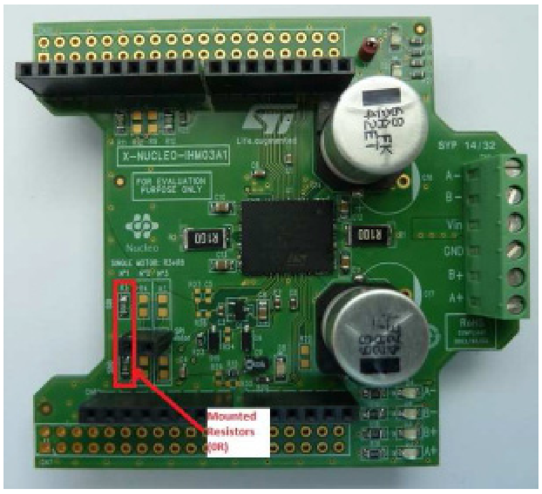
- Open your preferred toolchain (MDK-ARM from Keil, EWARM from IAR, or Atollic TrueStudio)
- Depending on the STM32 Nucleo board used, open the software project from:
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor1Motor\YourToolChainName\STM32F401RE-Nucleo for Nucleo STM32F401
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor1Motor\YourToolChainName\STM32F030R8-Nucleo for Nucleo STM32F030
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor1Motor\YourToolChainName\STM32L053R8-Nucleo for Nucleo STM32L053
- In order to adapt the default parameters which are used by the powerSTEP01 depending of your stepper motor characteristics, open the file: stm32_cube\Drivers\BSP\Components\powerstep01\powerstep01_target_config.h and modify the parameters which are postfixed by "_DEVICE_0".
- Rebuild all files and load your image into target memory
- Run the example. The motor automatically starts (see main.c to have the detailed demo sequence).

3.3.3 Setup to drive 2 motors

The X-NUCLEO-IHM03A1 expansion board for first motor must have:

- Mounted resistors (0R) on R3 and R6.
- Unmounted resistors on R4, R5, R7 and R8

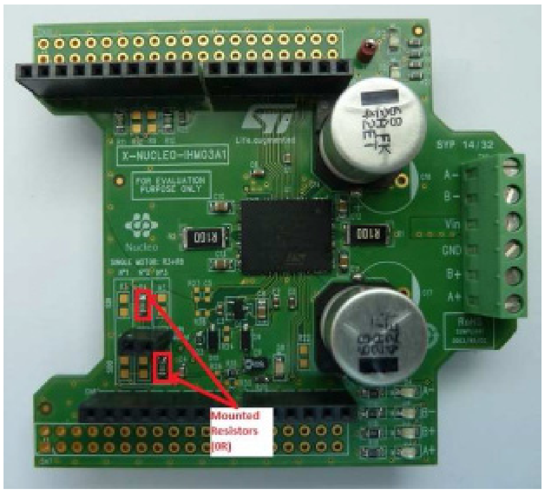
Figure 7. X-NUCLEO-IHM03A1 high power stepper motor driver configuration to drive motor 1/2



The X-NUCLEO-IHM03A1 expansion board for second motor must have:

- Mounted resistors (0R) on R4 and R8
- Unmounted resistors on R3, R5, R6 and R7

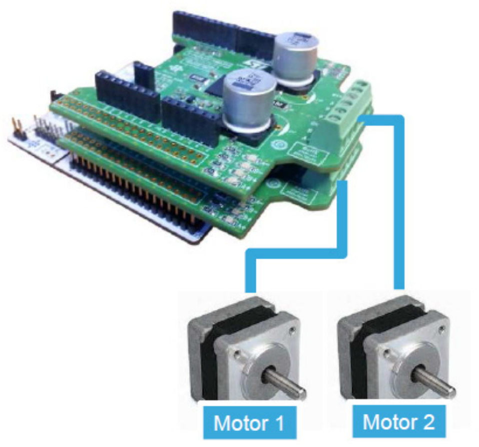
Figure 8. X-NUCLEO-IHM03A1 high power stepper motor driver configuration to drive motor 2/2



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 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 connectors Vin and Gnd to the DC power supply. The DC supply must be set to deliver the required voltage by the stepper motors.
- Connect each stepper motor to the bridge connectors A+/- and B+/- of their dedicated X-NUCLEO-IHM03A1 board

Figure 9. Board connections to drive 2 motors



Once the system setup is ready:

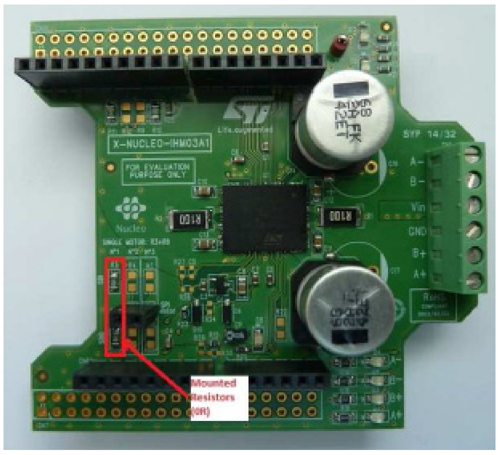
- Open your preferred toolchain (MDK-ARM from Keil, EWARM from IAR, or Atollic TrueStudio)
- Depending on the STM32 Nucleo board used, open the software project from:
  - stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor2Motors\YourToolChainName\STM32F401RE-Nucleo for Nucleo STM32F401
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor2Motors\YourToolChainName\STM32F030R8-Nucleo for Nucleo STM32F030
  - \stm32_cube\Projects\Multi\Examples\MotionControl\IHM03A1_ExampleFor2Motors\YourToolChainName\STM32L053R8-Nucleo for Nucleo STM32L053
- In order to adapt the default parameters which are used by the powerSTEP01 depending on your stepper motor characteristics, open the file: stm32_cube\Drivers\BSP\Components\powerstep01\powerstep01_target_config.h and modify the parameters which are postfixed by "_DEVICE_0" for first motor, and postfixed by "_DEVICE_1" for second motor
- Rebuild all files and load your image into target memory
- Run the example. The motors automatically start (see main.c to have the detailed demo sequence)

3.3.4 Setup to drive 3 motors

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

- Mounted resistors (0R) on R3 and R6.
- Unmounted resistors on R4, R5, R7 and R8

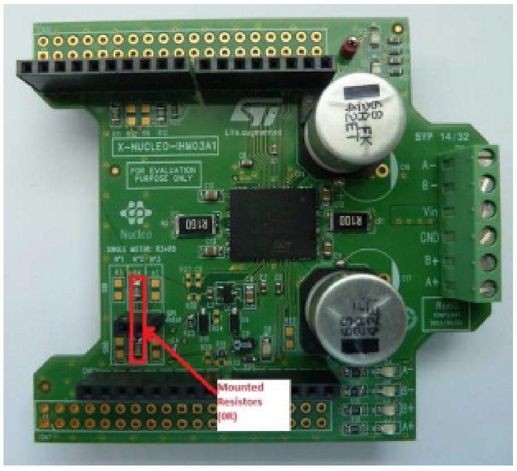
Figure 10. X-NUCLEO-IHM03A1 high power stepper motor driver configuration to drive motor 1/3



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

- Mounted resistors (0R) on R4 and R7
- Unmounted resistors on R3, R5, R6 and R8

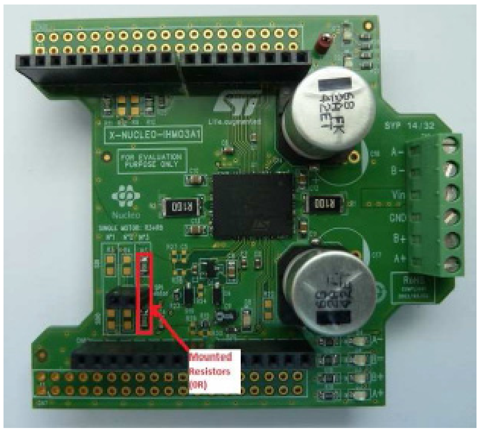
Figure 11. X-NUCLEO-IHM03A1 high power stepper motor driver configuration to drive motor 2/3



The X-NUCLEO-IHM03A1 expansion board for the third motor must have:

- Mounted resistors (0R) on R5 and R8
- Unmounted resistors on R3, R4, R6 and R7

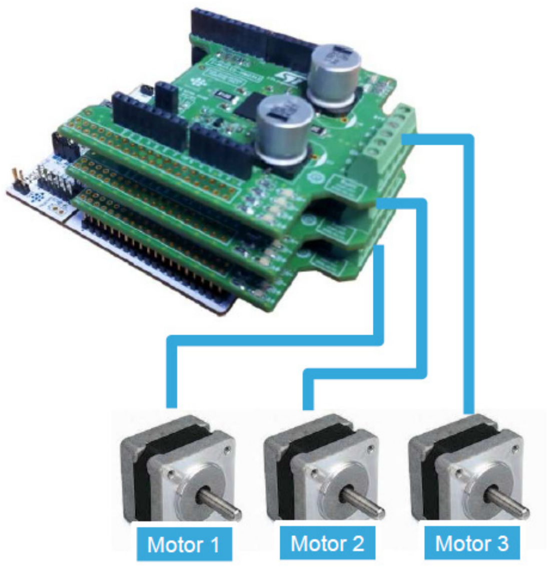
Figure 12. X-NUCLEO-IHM03A1 high power stepper motor driver configuration to drive motor 3/3



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 by the stepper motors.
- Connect each stepper motor to the bridge connectors A+/- and B+/- of their dedicated X-NUCLEO-IHM03A1 board.

Figure 13. Expansion boards connection to drive 3 motors



Once the system setup is ready:

- Open your preferred toolchain (MDK-ARM from Keil, EWARM from IAR, or Atollic TrueStudio)
- 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_ExampleFor3Motors\YourToolChainName\STM32L053R8-Nucleo for Nucleo STM32L053
- In order to adapt the default parameters which are used by the powerSTEP01 depending on your stepper motor characteristics, open the file: stm32_cube\Drivers\BSP\Components\powerstep01\powerstep01_target_config.h and modify the parameters which are postfixed by "_DEVICE_0" for the first motor, postfixed by "_DEVICE_1" for the second motor and by "_DEVICE_2" for the third motor
- Rebuild all files and load your image into target memory
- Run the example. The motors automatically start (see main.c to have the detailed demo sequence).

4 Acronyms and abbreviations

Table 2. 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

5 Revision history

Table 3. Document revision history

Date	Revision	Changes
22-Jul-2015	1	Initial release.