UM2022
User manual

SPINFamily evaluation tool

Introduction

The SPINFamily evaluation tool allows STMicroelectronics users to easily evaluate functionalities and performances of the STSPIN family devices.

- L6206: DMOS dual full bridge driver
- L6208: DMOS driver for bipolar stepper motor
- L6470: dSPIN™ fully integrated microstepping motor driver with motion engine and SPI
- L6472: Fully integrated microstepping motor driver
- L6474: Fully integrated microstepping motor driver
- L6480: Microstepping motor controller with motion engine and SPI
- L6482: Microstepping motor controller with motion engine and SPI
- powerSTEP01: System-in-package integrating microstepping controller and 10 A power MOSFETs

This software is designed to work with the STEVAL-PCC009V2 interface board, or the STM32 Nucleo boards, and the device starter-kit boards (EVAL64xx-DISC).

Before starting, please take some time to visit the Motor Control ICs webpage: www.st.com/web/en/catalog/sense_power/FM142/CL851. There you can find updated datasheets, application notes and the latest version of this software.

Contents

1 Hardware requirements

1.1 PC prerequisites

This evaluation software is a Windows® based software and requires Microsoft™ .NET Framework 2.0. It is possible to free download and install this framework from: www.microsoft.com.

1.2 Supported boards

STEVAL-PCC009V2 interface board connected with:

- EVAL6470H, EVAL6470PD
- EVAL6472H, EVAL6472PD
- EVAL6474H, EVAL6474PD
- EVAL6480H
- EVAL6482H
- EVLPOWERSTEP01
- EVAL6206Q
- EVAL6208Q

Discovery boards:

- EVAL6470H-DISC
- EVAL6472H-DISC
- EVAL6474H-DISC
- EVAL6480H-DISC
- EVAL6482H-DISC

STM32 Nucleo (NUCLEO-F030R8, NUCLEO-F401RE, NUCLEO-L053R8) connected with:

- X-NUCLEO-IHM01A1 (L6474)
- X-NUCLEO-IHM03A1 (powerSTEP01)
- X-NUCLEO-IHM04A1 (L6206PD)
- X-NUCLEO-IHM05A1 (L6208PD)

2 Quick start guide

The first steps for easy use of the SPINFamily evaluation tool.

Note: The software is designed to work in the demonstration mode, so all GUI functionalities can be explored even if no communication board is present.

2.1 STEVAL-PCC009V2 + demonstration board

1. First time only: update the MCU firmware following the procedure described in *Section Appendix A: Firmware update on page 72*.
2. Connect the STEVAL-PCC009V2 to the PC through the USB cable.
3. Connect the STEVAL-PCC009V2 to the evaluation board through the 10 poles or 30 poles connector.
   Some demonstration boards support the daisy chain connection (i.e. controlling more boards with a single MCU board). Refer to the user manual of each board for more details.
4. Supply the evaluation board.
5. Start the SPINFamily evaluation tool (by default you can find it in your Start menu > All programs > STMicroelectronics > SPINFamily Evaluation Tool).
6. When the application is started select the device under evaluation from the **PCC009V2** and **EVAL** tab.
7. Wait a few seconds for initialization.
8. Click on the “Connect” button (USB symbol) to connect the demonstration board to the communication board.
   If a warning message is shown, please check if the evaluation board is properly supplied.
9. Enjoy.

2.2 Discovery boards

1. **First time only:** update the MCU firmware following the procedure described in *Section Appendix A: Firmware update on page 72*.
2. Check if the BOOT jumper is closed. If not, close it before starting.
3. Connect the board to the PC through the USB cable.
4. Supply the evaluation board.
5. Reset the MCU pushing the “Reset” button.
6. All the LEDs blink in sequence. If not, check the board supply and reset the MCU again.
7. Start the SPINFamily evaluation tool (by default you can find it in your Start menu > All programs > STMicroelectronics > SPINFamily Evaluation Tool).
8. When the application is started select the device under evaluation from the **PCC009V2** and **EVAL** tab. Wait a few seconds for initialization.
9. Click on the “Connect” button (USB symbol) to connect the demonstration board to the communication board.
   If a warning message is shown, please check if the evaluation board is properly supplied.
10. Enjoy.

2.3 Nucleo platform

1. **First time only:** update the MCU firmware following the procedure described in *Section Appendix A: Firmware update*.
2. Connect the Nucleo development board to the PC through the USB cable.
3. Connect the Nucleo development board to the expansion board.
4. Some expansion boards support the daisy chain connection (i.e. controlling more boards with a single Nucleo development board). Refer to the user manual of each board for more details.
5. Supply the expansion board.
6. Start the SPINFamily evaluation tool (by default you can find it in your Start menu > All programs > STMicroelectronics > SPINFamily Evaluation Tool).
7. When the application is started select the proper Nucleo board from the “Serial port” drop-down list and the device under evaluation from the Nucleo tab. Wait a few seconds for initialization.
8. Click on the “Connect” button (USB symbol) to connect the demonstration board to the communication board.
   If a warning message is shown, please check if the expansion board is properly supplied.
9. Enjoy.

3 GUI windows

Depending on the selected STSPIN device, the application will show different panels.

The menu is common to all STSPIN devices.

3.1 Menu, toolbar and status bar

The menu and toolbar provide access to extra tools and allow opening/saving the device configuration.

The status bar on the bottom side of the form shows the current interface board and FLAG/BUSY lines status.

Following you can find a brief description of menu items with the corresponding toolbar icon.

- **File** menu allows managing the configuration files:
  - **Open:** load a single or a group of configuration files and write them into the device(s) (see *Section 3.5.1 on page 40*).
  - **Save:** save a single or a group of configuration files (refer to *Section 3.5.1*).
  - **Exit:** close the application.
- **Tools** menu provides access to extra tools and allows changing the application options:
  - **Connect board:** connect or disconnect the communication board. If checked, it indicates that the communication board is connected.
  - **Select Device…**: change the current device or board.
  - **Wizard:** show the Wizard (see *Section 3.3 on page 28*) (for L6470, L6472, L6480, L6482 and powerSTEP01).
  - **Register map:** open the Register Map tool (see *Section 3.4 on page 36*).
  - **Device configuration:** open the Device Configuration tool (see *Section 3.5 on page 40*).
  - **BEMF compensation:** (L6470 and L6480 only): open the BEMF compensation evaluator tool (see *Section 3.6 on page 50*).
  - **Script editor:** open the Scripts Editor environment (see *Section 3.8 on page 54*). If the tool is already open, it is brought to top.
  - **Export Header File:** open the save dialog to save the generated header file for the current FW platform with current registers value.
  - **Options:** open the Application Options window (see *Section 3.7 on page 53*).
- ? **Help** menu contains support information:
  - "**About**": give detailed information about the software.
  - "**Web**": open the Motor Control ICs webpage on the STMictroelectronics site.

3.2 Main panel

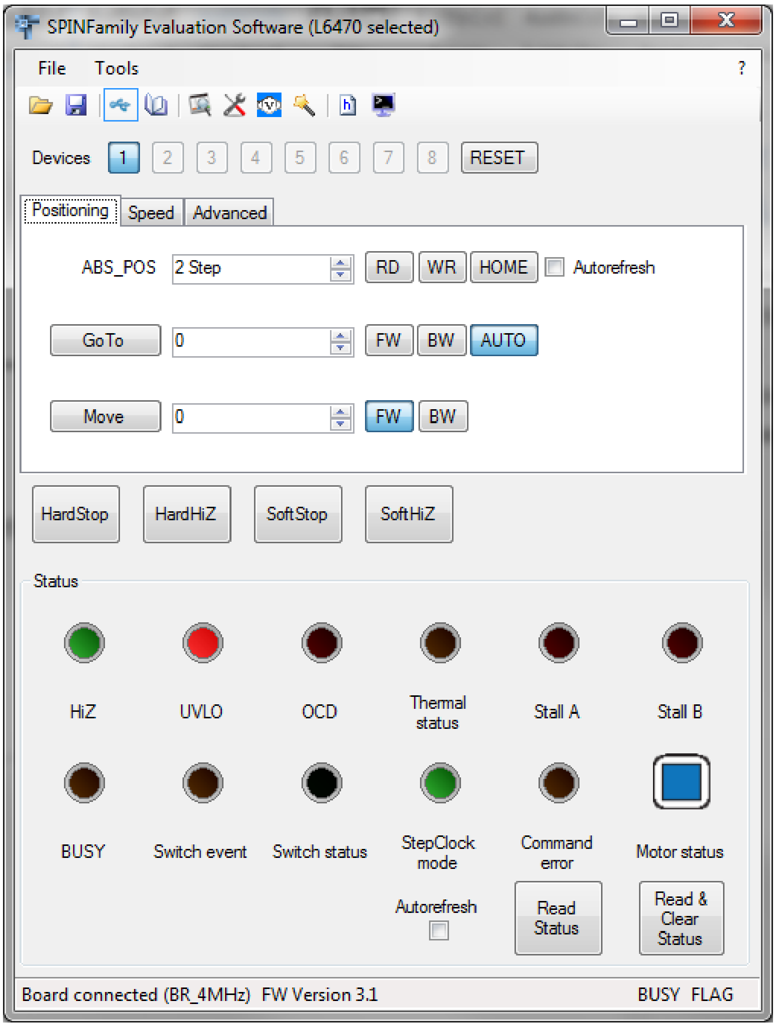
The main panel is divided into two sections:

- The command section, on the top, that collects all the device commands and allows reading/writing the absolute position and the speed registers. Three types of the panel are displayed depending on the connected device:
  - Stepper command panel (powerSTEP01 and L64xx devices)
  - L6208 stepper command panel
  - L6206 brush DC command panel
- The device status display, on the bottom, that shows the last information collected from the status register. It also depends on the current device.

3.2.1 Stepper command panel (powerSTEP01 and L64xx devices)

Figure 1 shows the main panel for the L6470, L6472, L6480, L6482 and powerSTEP01 devices.

Figure 1. Main panel for L6470, L6472, L6480, L6482 and powerSTEP01



SPINFamily Evaluation Software (L6470 selected)

ABS_POS: 2 Step

GoTo: 0

Move: 0

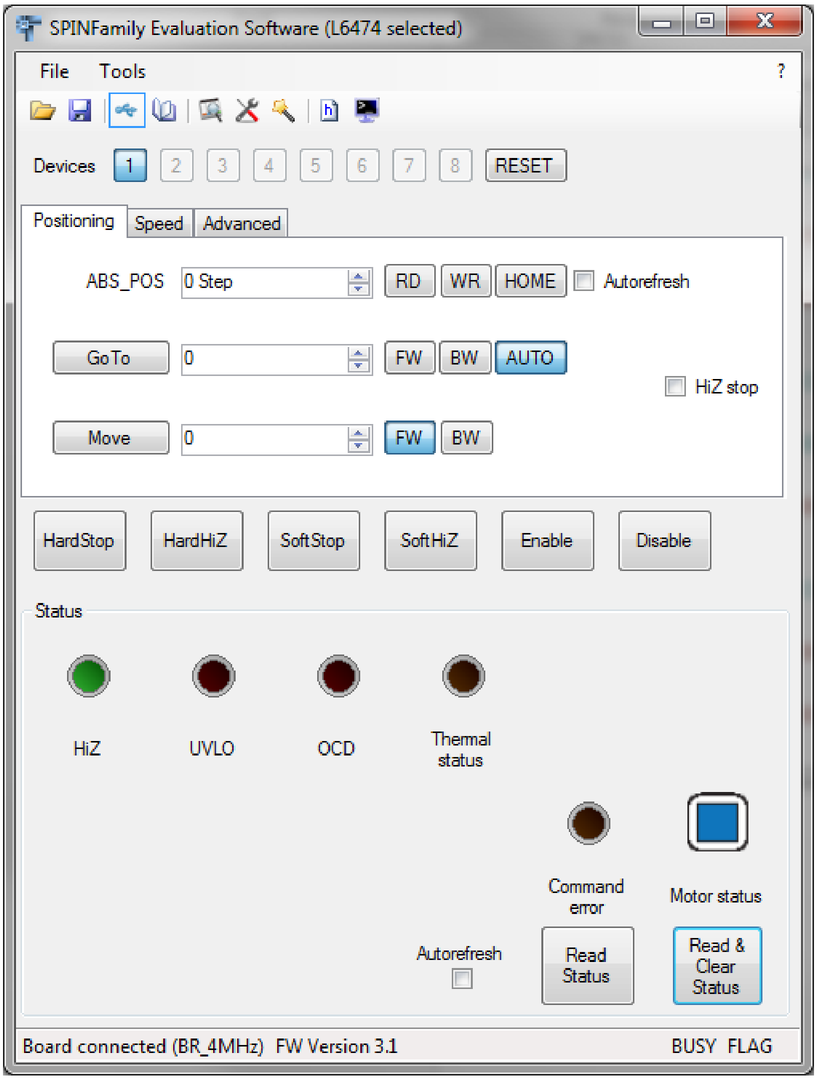
Board connected (BR_4MHz) FW Version 3.1 BUSY FLAG

The command section collects all commands and allows reading and writing the ABS_POS, MARK and SPEED registers.

For a detailed description of the command set of the device, please refer to the device datasheet.

The L6474 device is supported by a main panel different from other L64xx devices and the powerSTEP01; it has less registers and some features are not available in the GUI.

Figure 2. Main panel (L6474)



SPINFamily Evaluation Software (L6474 selected)

ABS_POS 0 Step

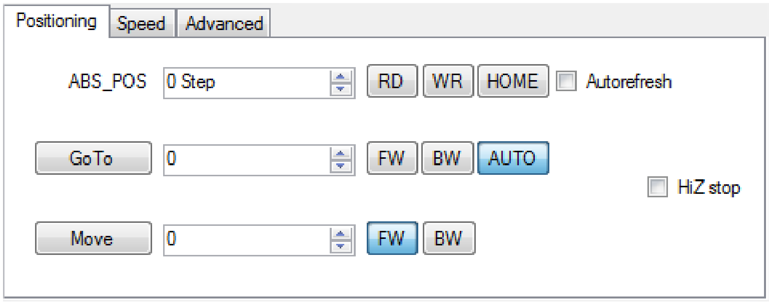
GoTo 0

Move 0

Board connected (BR_4MHz) FW Version 3.1

Positioning tab

Figure 3. Positioning tab



ABS_POS 0 Step

GoTo 0

Move 0

- **Goto**

GoTo and GoTo_DIR commands can be sent to the selected device clicking on **GoTo** button or pushing the return key in the adjacent numeric box.

The position argument is set by a numeric box next to the button and can vary within -2097152 and 2097151 (absolute position is expressed in a 2's complement format). You can write the target position directly or you can change it using up and down arrows positioned on the right side of the box.

If the **AUTO** button is selected, the GoTo command is sent and the motion direction is selected by the IC minimum path algorithm. **FW** and **BW** buttons force a forward or backward direction sending a GoTo_DIR command.

While working with the L6474 device and the **AUTO** button is selected, the GoTo command is sent and the GUI calculates the direction to have a minimum path.

- **Move**

Clicking on the **Move** button, a Move command is sent to the connected device. The motion direction is selected through the **FW** (forward) and **BW** (backward) buttons and the number of steps is set by a numeric box next to the button.

This value goes from 0 to 4194303. You can directly write it within the box or you can vary it using up and down arrows. The Move command can be also sent pushing the **Return** key into the numeric box.

- **ABS_POS**

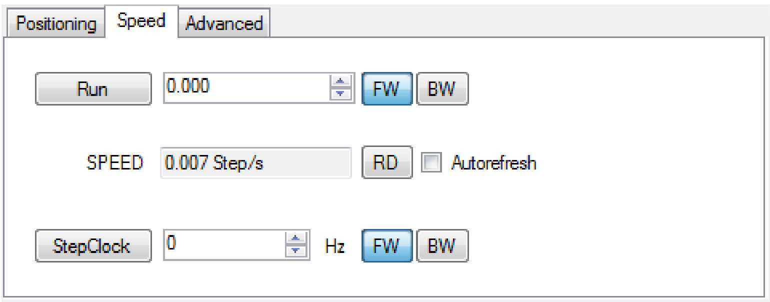
You have quick access to the ABS_POS register. The **RD** button reads the ABS_POS of the selected device, the numeric box displays the value. If the **Autorefresh** checkbox is checked, the absolute position is automatically updated at the selected polling rate.

To write the register, set the value into the numeric box (2097152 to 2097151) and push the return key or click on the **WR** button.

The **HOME** button resets the ABS_POS register to the home position (zero).

Speed tab

Figure 4. Speed tab



The **Run** button sends a Run command to the selected device. The direction is selected through the **FW** (forward) and **BW** (backward) button.

The value ranges from 0 to 15624.985 (expressed in step/s) and can be directly written in the box or it can be changed using the up and down arrows. Pushing the **Return** key into the numeric box sends also the Run command.

The SPEED numeric box shows the current SPEED register by clicking the **RD** button. If the **Autorefresh** checkbox is checked, the SPEED value is automatically updated at the selected polling rate.

Advanced tab

The Advanced tab is different depending on the selected device.

Figure 5. Advanced tab for L6470, L6472, L6480, L6482 and powerSTEP01

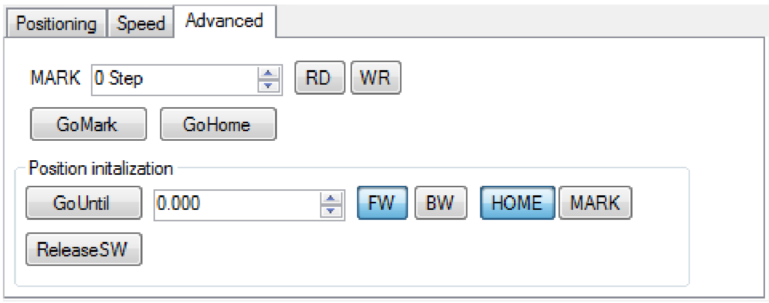
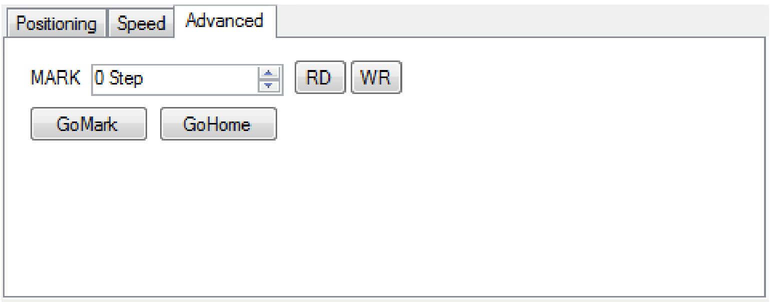


Figure 6. Advanced tab for L6474



- **MARK**

The **GoMark** and **GoHome** buttons send the respective command to the selected device.

A quick access to the MARK register is also provided. The **RD** button reads the current MARK value.

The register value can be written setting the desired value into the numeric box (-2097152 to 2097151) and pushing the **Return** key or clicking on the **WR** button.

- **Position initialization** (not available for the L6474)

The **GoUntil** button sends the respective command using the parameters indicated by the adjacent controls:

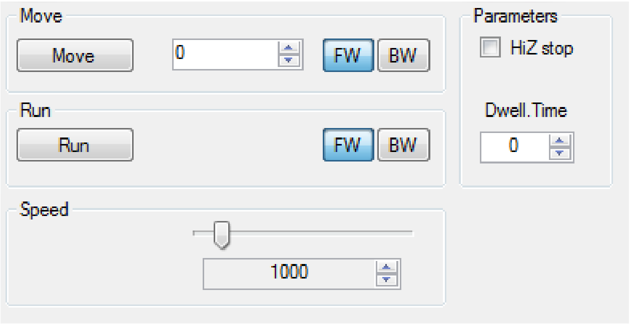
- The **numeric box** defines the target speed (expressed in step/s). Its value ranges from 0 to 15624.985 and can be set directly or by means of the up and down arrows.
- The **FW** and **BW** buttons select the motion direction.
- The **HOME** and **MARK** buttons select the action performed at the SW pin falling edge. If **HOME** is selected the ABS_POS register is set to zero (home position), otherwise its value is stored into the MARK register.

Pushing the **Return** key in the numeric box does not send a GoUntil command, the **GoUntil** button has to be used.

The **ReleaseSW** button sends a ReleaseSW command using the parameters indicated by the same controls used for the GoUntil command. In this case, the **HOME** and **MARK** buttons select the action performed at the SW pin rising edge.

3.2.2 L6208 stepper command panel

Figure 7. L6208 stepper command panel



- **Move**

Clicking on the **Move** button, a Move command is sent to the current active device. The motion direction is selected through the **FW** (forward) and **BW** (backward) buttons and the number of steps is set by a numeric box next to the button. This value goes from 0 to 4194303. You can directly write it within the box or you can vary it using up and down arrows.

The Move command can be also sent pushing the **Return** key in the numeric box.

- **Run**

The **Run** button sends a Run command to the selected device. The direction is selected through the **FW** (forward) and **BW** (backward) button.

- **Speed**

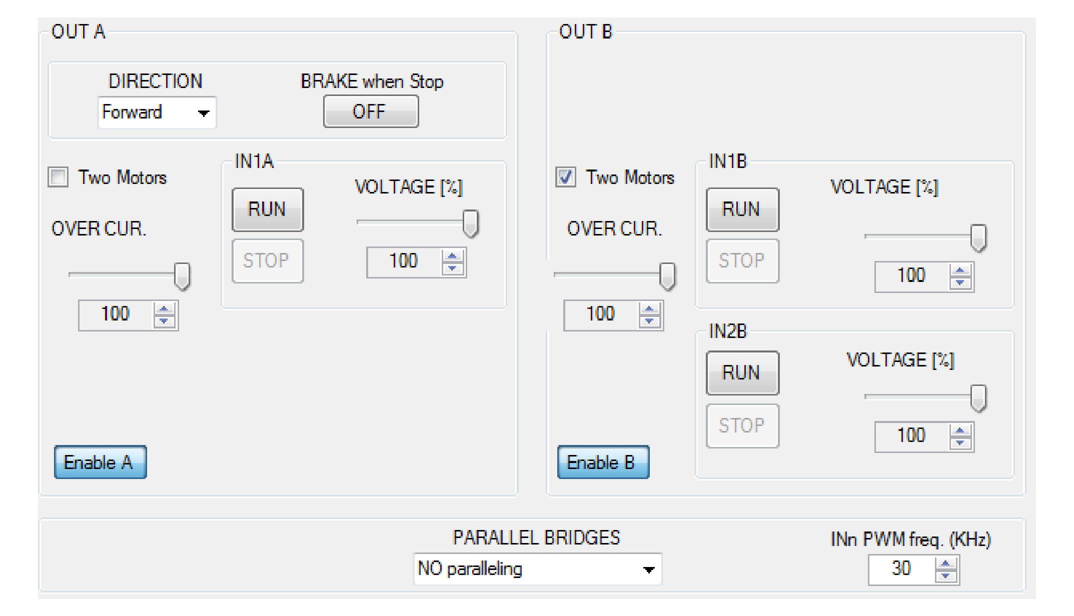
The value range of the speed is from 0 to 15624.985 (expressed in step/s) and can be directly written in the box or it can be changed using the up and down arrows or the horizontal slide.

- **Parameters**

If the **HiZ stop** box is checked, at the end of the motion the outputs power bridges are all turned off, otherwise they are kept enabled. The deceleration torque is kept for the Dwelling time (ms) waiting time before the hold torque or the tristate is set.

3.2.3 L6206 brush DC command panel

Figure 8. L6206 brush DC command panel



OUT A: DIRECTION Forward; BRAKE when Stop OFF; □ Two Motors; OVER CUR. 100; IN1A VOLTAGE [%] 100

OUT B: ☑ Two Motors; OVER CUR. 100; IN1B VOLTAGE [%] 100; IN2B VOLTAGE [%] 100

PARALLEL BRIDGES NO paralleling; INn PWM freq. (KHz) 30

Each power bridge (A and B) can be enabled/disabled independently and the relative parameters can be set in the dedicated section (OUTA and OUTB).

- **Inputs signals (IN1A and IN1B)**

The Inputs PWM frequency can be set by a numeric box in the **INnPWM freq** field in the bottom right side of the panel. This value goes from 10 to 100 kHz with a step of 10 kHz, you can directly write it within the box or you can vary it using up and down arrows.

The value range of input signal duty-cycle **VOLTAGE** can be varied from 0 to 100 % and can be directly written in the box or changed using the up and down arrows or the horizontal slide.

The user can set the IN1 parameters, the companion IN2 of the same bridge is driven with the same PWM but with inverted polarity (if the **Two motors** box is not checked).

- **RUN** and **DIRECTION**

The **RUN** button sends a Run command to the selected device. The direction of a current mode controlled DC motor connected to the IN1A - IN2A or IN1B - IN2B is selected through the **DIRECTION** drop-down menu: **Forward** or **Backward** options.

- **STOP**

The **STOP** button sends a stop command to the selected bridge. The interested bridge is disabled putting the bridge outputs in high impedance status (if **BRAKE when STOP** control is OFF).

- **BRAKE when Stop**

If the **BRAKE when Stop** button is **ON**, pushing the **STOP** button will cause both high-side power MOSFET of the interested bridge to turn ON, making a short-circuit across the motor winding.

- **Overcurrent**

The overcurrent detection threshold can be set varying the duty-cycle of the PWM signal applied to the PROGCL pin (through the low-pass filter mounted on the L6206Q board). The duty-cycle value **OVER CUR.** can be varied from 0 to 100 % and can be directly written in the box or changed using the up and down arrows or the horizontal slide.

- **Enable/Disable**

The **Enable/Disable** buttons disable the interested bridge by pulling its EN input down to ground.

Enable is automatically set when the **RUN** button is clicked, depending on the selected parallel mode and **Two motors** option, disable is not automatically set. For example, if **Two motors** is checked, **RUN** will enable the bridge, but stopping one motor will not disable the bridge if the other motor is running.

- **Two motors**

If the **Two motors** box is checked, you can drive two motors independently. The **IN2** section will be available to manage the PWM duty-cycle and send a separate RUN or STOP command. The **DIRECTION** and **Brake when Stop** option are disabled (see *Section : Parallel Bridges* for more details).

- **Parallel bridges**

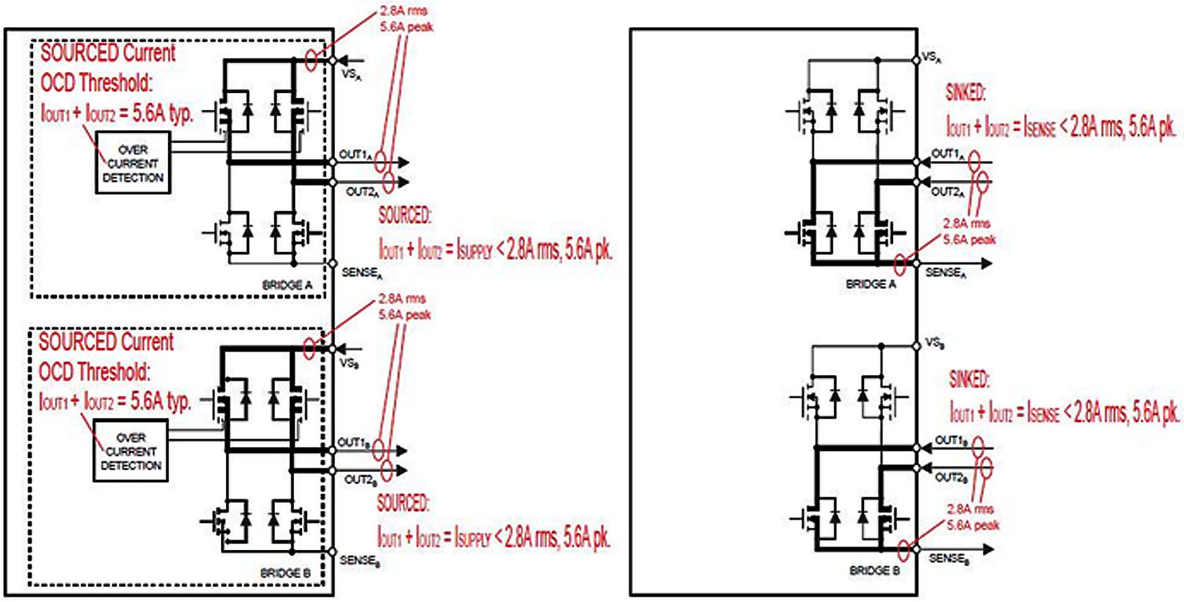
The **PARALLEL BRIDGES** drop-down menu allows to select the device parallel mode. See *Section : Parallel Bridges* for more details.

Parallel Bridges

The outputs of the L6206 can be paralleled to increase the output current capability or reduce the power dissipation in the device at a given current level. It must be noted, however, that the internal wire bond connections from the die to the power or sense pins of the package must carry the current in both of the associated half bridges (see *Figure 9*).

When the two halves of one full bridge (for example OUT1A and OUT2A) are connected in parallel, the peak current rating is not increased since the total current must still flow through one bond wire on the power supply or sense pin. In addition, the overcurrent detection senses the sum of the current in the upper devices of each bridge (A or B) so connecting the two halves of one bridge in parallel does not increase the overcurrent detection threshold.

Figure 9. VS and SENSE pins maximum current handling



The **Parallel bridges** drop-down menu let you select one of the device parallel mode, following the description of each mode.

Table 1. Device parallel mode options

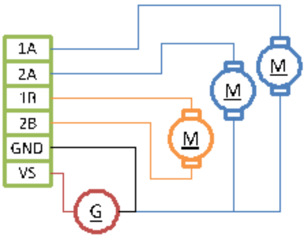
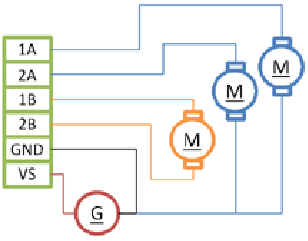
Parallel mode	Two motors A	Two motors B	Motors configuration
No paralleling	Unchecked	Unchecked	 **Two DC motors bidirectional** Each motor driver accepts a maximum of 2.8 A
No paralleling	Checked	Unchecked	 **Two DC motors A + one bidirectional motor B** Each motor driver accepts a maximum of 2.8 A

Table 1. Device parallel mode options (continued)

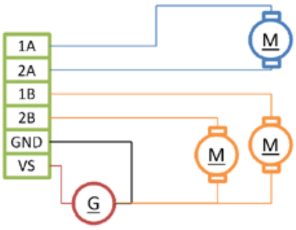
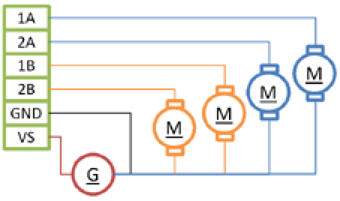
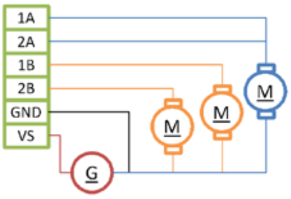
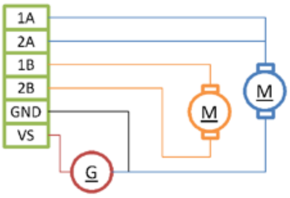
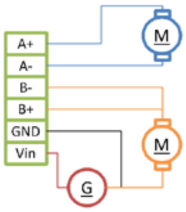
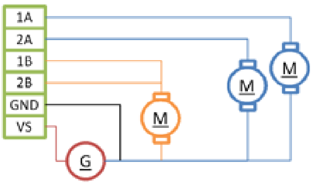
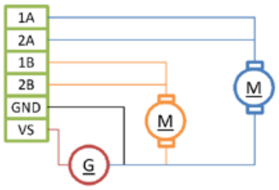
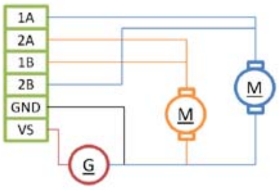
Parallel mode	Two motors A	Two motors B	Motors configuration
No paralleling	Unchecked	Checked	 **One bidirectional motor A + two DC motors B** Each motor driver accepts a maximum of 2.8 A
No paralleling	Checked	Checked	 **Four DC motors unidirectional** The supply voltage of motors is 50 V DC maximum.
IN1A + IN2A	NA	Unchecked	 **One DC motor unidirectional A low overcurrent,** **two DC motors unidirectional B**
IN1A + IN2A	NA	Checked	 **One DC motor unidirectional A low overcurrent,** **one bidirectional motor B**

Table 1. Device parallel mode options (continued)

Parallel mode	Two motors A	Two motors B	Motors configuration
IN1B + IN2B	Unchecked	NA	 One bidirectional motor A, one DC motor unidirectional B low overcurrent
IN1B + IN2B	Checked	NA	 Two DC motor unidirectional A, one bidirectional motor B low current
IN1A + IN2A IN1B + IN2B	NA	NA	 Two DC motor unidirectional low current
IN1A + IN1B IN2A + IN2B	NA	NA	 Two DC motor unidirectional high current

3.2.4 Stop buttons

At the bottom of the command section the stop commands can be found. Clicking on the **HardStop**, **HardHiZ**, **SoftStop** or **SoftHiZ** buttons the respective command is sent to the selected device.

The L6474 device and has two more buttons allowing to enable or disable the power bridges.

Figure 10. Stop buttons for L6470, L6472, L6480, L6482 and powerSTEP01

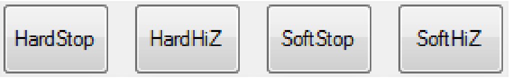
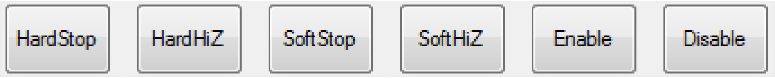


Figure 11. Stop buttons for L6474



3.2.5 Status panel

The status panel shows the last STATUS register value of the current active device. This panel is updated every time the STATUS register is read through a *GetStatus* or a *GetParam* command.

The **Read Status** button allows reading the STATUS register. The **Read** and **Clear Status** button read the register and clear the failure conditions by sending a *GetStatus* command.

If the **Autorefresh** box is checked, the display is automatically updated at selected System polling _System_pollingrate, but error/failure flags are not cleared (*GetParam* command is used to get the *STATUS* value instead of GetStatus command).

Detailed differences between the *GetParam* and *GetStatus* command can be found on the device datasheet.

Not all status description are applicable to all type of the device. See from *Figure 12* to *Figure 15* to verify which status is applicable to your selected device.

Figure 12. Status panel for L6470, L6472, L6480, L6482 and powerSTEP01

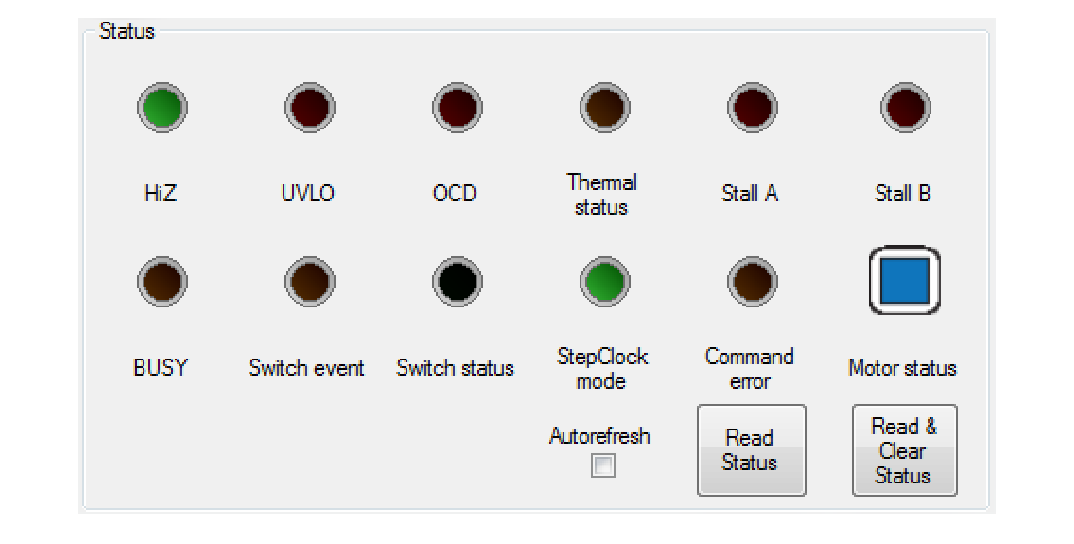


Figure 13. Status panel for L6474

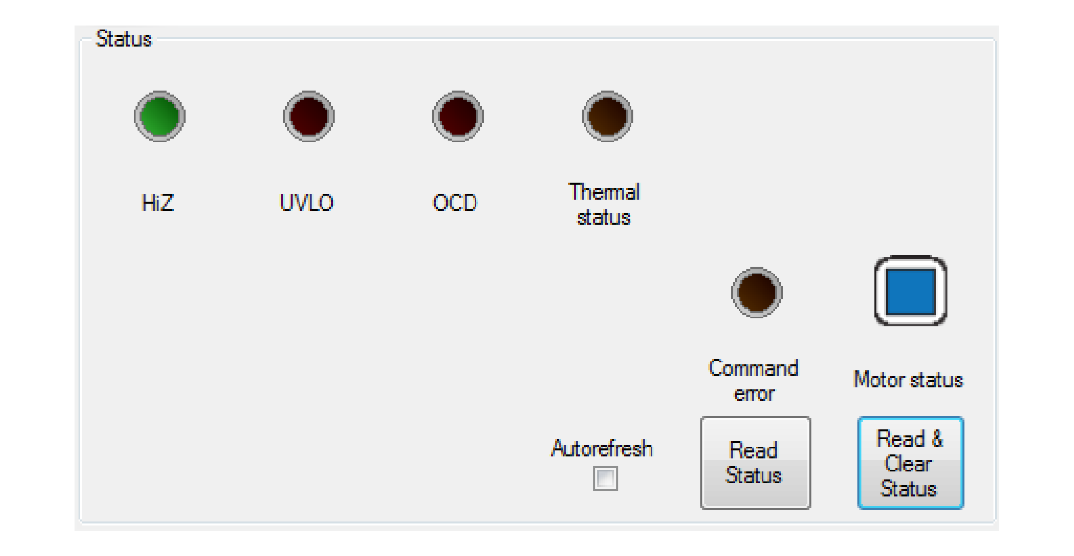


Figure 14. Status panel for L6208

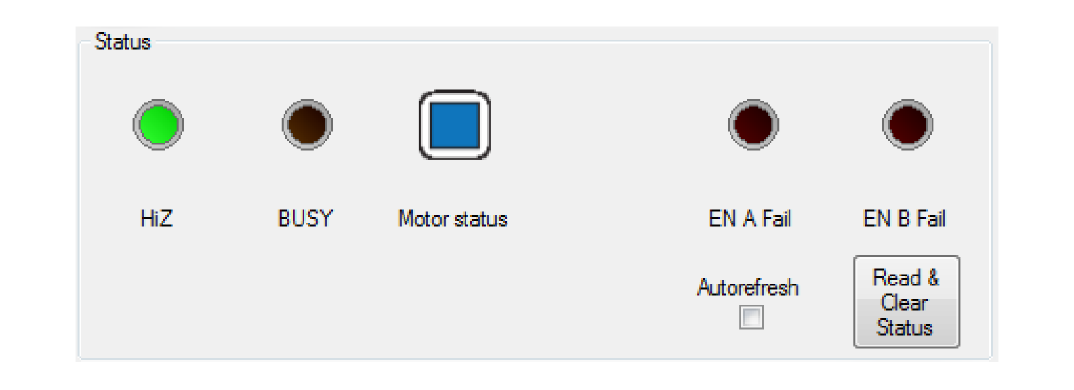
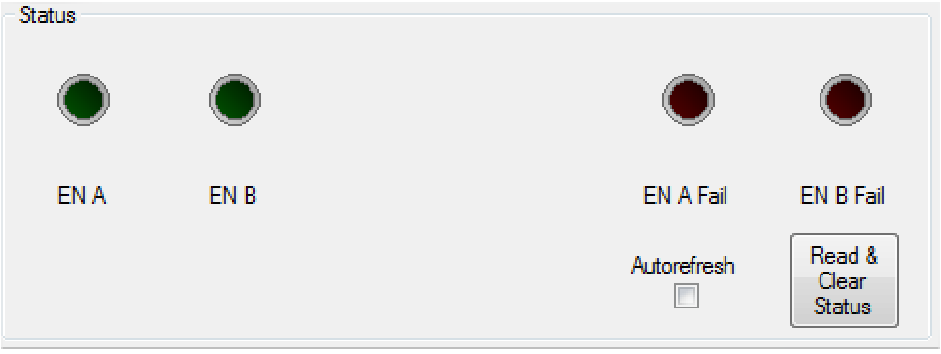


Figure 15. Status panel for L6206



HiZ

The **HiZ** LED indicates the high impedance status: if it is on (green), the device outputs are disabled, otherwise the outputs are active (black).

The LED status is related to the HiZ bit value of the STATUS register according to *Table 2*.

Table 2. HiZ status

HiZ	LED	Description
1	🟢	Device outputs disabled
0	⚫	Device outputs active

UVLO

When the **UVLO** LED is red an undervoltage or the reset/power-up event occurred. If it is off (black) no fails are present.

When using an L648x or the powerSTEP01 device, the LED reports the UVLO_ADC failure through a yellow light.

The LED status is related to the UVLO bit or eventually to the UVLO_ADC bit values of the STATUS register according to *Table 3*.

Table 3. UVLO status

UVLO	UVLO_ADC (L648x or powerSTEP01)	LED	Description
1	1	⚫	No failure
1	0	🟡	UVLO on ADC event
0	X	🔴	UVLO or reset/power-up event

OCD

The **OCD** LED indicates that an overcurrent event has been detected: if it is on (red) an overcurrent event occurred; otherwise no fails are present (black).

The LED status is related to the OCD bit value of the STATUS register according to *Table 4*.

Table 4. OCD status

OCD	LED	Description
1	⚫	No failure
0	🔴	Overcurrent event

Thermal status

When a device of the L647x family is used, the LED status is related to TH_WRN and TH_SD bit values.

When a device of the L648x family or powerSTEP01 is used, it is related to the TH_STATUS parameter value.

The relations are described in *Table 5*.

Table 5. Thermal status

TH_WRN (L647x)	TH_SD (L647x)	TH_STATUS (L648x and powerSTEP01)	LED	Description
1	1	00	⚫	The device temperature is below the warning threshold.
0	1	01	🟠	The warning temperature has been reached.
0	0	10	🔴	The device temperature reached the shutdown threshold: a thermal shutdown event occurred.
NA	NA	11	🟣	A device shutdown event occurred (L648x only).
1	0	NA	NA	Invalid condition.

Stall A and Stall B

Stall A and **Stall B** LEDs indicate a stall detection warning. If the LED is on (red) a stall event occurred in the respective bridge; no failures are present otherwise (black).

The LED statuses are related to the STEP_LOSS_A and STEP_LOSS_B bit values of the STATUS register according to *Table 6*.

Table 6. Stall A or Stall B status

STEP_LOSS_X	LED	Description
1	⚫	No failure
0	🔴	A stall event occurs

Note: This indication is available in the L6470 and L6480 devices only.

BUSY

The **BUSY** LED is turned on (yellow) during a command execution. When it is off (black), the last command has been executed and the device is idle.

The LED status is related to the BUSY bit value of the STATUS register according to *Table 7*.

Table 7. Busy status

BUSY	LED	Description
1	⚫	Last command executed, idle device
0	🟡	Command execution

SW event

If the **SW event** LED is on (yellow), the SW input has been forced low (switch turn-on event); otherwise no falling edges has been detected on the SW input (black).

The LED status is related to the SW_EVN bit value of the STATUS register according to *Table 8*.

Table 8. SW event status

SW_EVN	LED	Description
0	⚫	No SW input event
1	🟡	SW input pin transition event (falling edge)

SW status

The **SW status** LED indicates the SW input status: if it is on (green) the SW input is low (switch closed); otherwise (black) the SW input is high (switch open).

The LED status is related to the SW_F bit value of the STATUS register according to *Table 9*.

Table 9. SW status

SW_F	LED	Description
0	⚫	SW input high (external switch open)
1	🟢	SW input low (external switch closed)

StepClock mode

If the **StepClock mode** LED is on (green), the device is operating in the step-clock mode; otherwise the device is operating in the standard mode (black).

The LED status is related to the SCK_MOD bit value of the STATUS register according to *Table 10*.

Table 10. StepClock mode status

SCK_MOD	LED	Description
0	⚫	Device operating in standard mode
1	🟢	Device operating in step-clock mode

Command error

If the **Command error** LED is on (yellow), a wrong or a not performable command has been sent to the device; otherwise all sent commands have been correctly executed (black).

When a device of the L647x family is used, the LED status is related to WRONG_CMD and NOTPERF_CMD bit values.

When a device of the L648x family or powerSTEP01 is used, it is related to the CMD_ERROR parameter value.

The relations are described in *Table 11*.

Table 11. Command error status

NOTPERF_CMD (L647x)	WRONG_CMD (L647x)	CMD_ERROR (L648x or powerSTEP01)	LED	Description
0	0	0	⚫	All sent commands correctly executed
X	1	NA	🟡	A wrong command is sent to the device (L647x).
1	X	NA	🟡	The sent command cannot be performed
NA	NA	1	🟡	The command received by SPI cannot be performed or it is wrong (L648x)

Motor status

The **Motor status** icon indicates the current status of the motor. Different icons represent the acceleration, deceleration, constant speed and holding status in both directions.

For the L6470, L6472, L6480, L6482 and powerSTEP01, the displayed icon depends on DIR and MOT_STATUS parameters of the STATUS register according to *Table 12*.

Table 12. Motor status for L6470, L6472, L6480, L6482 and powerSTEP01

Motor Status	DIR	MOT_STATUS	LED
Stopped	X	00	■
Accelerating in forward direction	1	01	▷▶
Decelerating in forward direction	1	10	▶▷
Running in forward direction	1	11	▶
Accelerating in backward direction	0	01	◀◁
Decelerating in backward direction	0	10	◁◀
Running in backward direction	0	11	◀

For the L6474, the displayed icon depends on the DIR register and MOT_STATUS command according to*Table 13*.

Table 13. Motor Status for L6474

Motor Status	DIR	MOT_STATUS	LED
Stopped	X	0	■
Running in forward direction	1	1	▶
Running in backward direction	0	1	◀

EN A and EN B

The **EN A** and **EN B** LEDs indicate the status of the EN_A and EN_B pin of the L6206 device.

The LED status is related to the EN_A and EN_B bit value of the STATUS command according to *Table 14*.

Table 14. EN A / EN B status

EN_A or EN_B	LED	Description
1	🟢	Enable pin high
0	●	Enable pin low

EN A Fail and EN B Fail

If the **EN A Fail** or **EN B Fail** LED is on (red), a limit A or B failure has been detected on the L6206 device.

The LED status is related to the *ENAFail* and *ENBFail* bit value of the STATUS command according to *Table 15*.

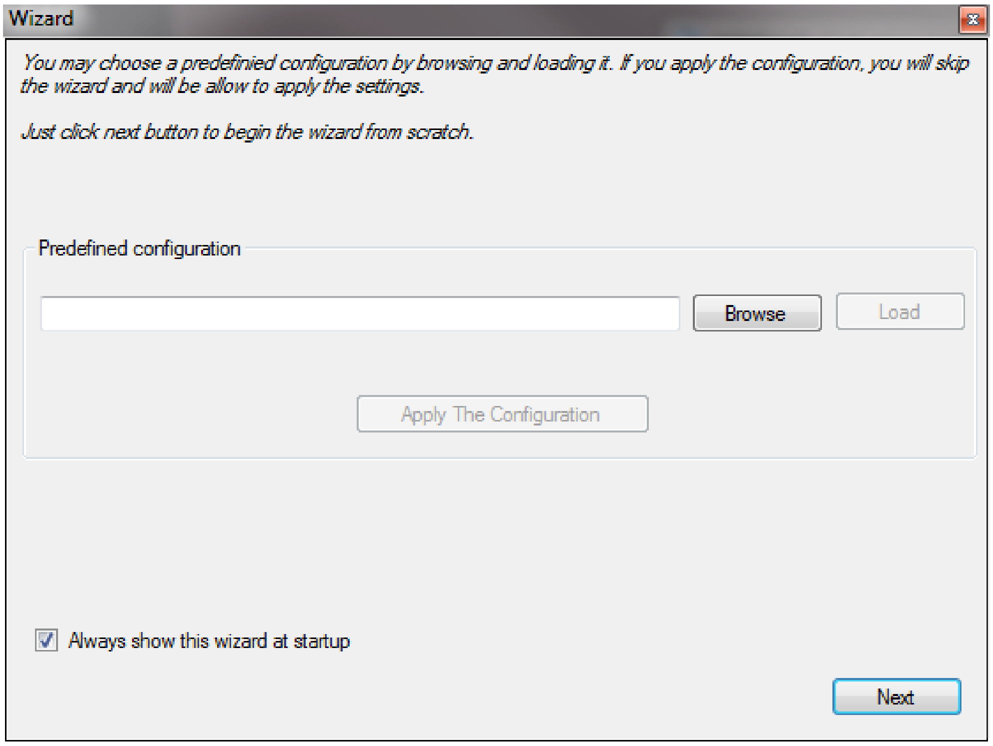
Table 15. EN A Fail and EN B Fail status

EN A Fail EN B Fail	LED	Description
0	●	
1	🔴	Overcurrent limit threshold reached

3.3 Wizard

The wizard window helps the user to configure the devices by using predefined values depending on the hardware (board, motor, etc.). Clicking on the button in the toolbar or selecting **Wizard** in the **Tools** menu opens the **Wizard** panel.

Figure 16. Wizard configuration panel



You may choose a predefinied configuration by browsing and loading it. If you apply the configuration, you will skip the wizard and will be allow to apply the settings.

Just click next button to begin the wizard from scratch.

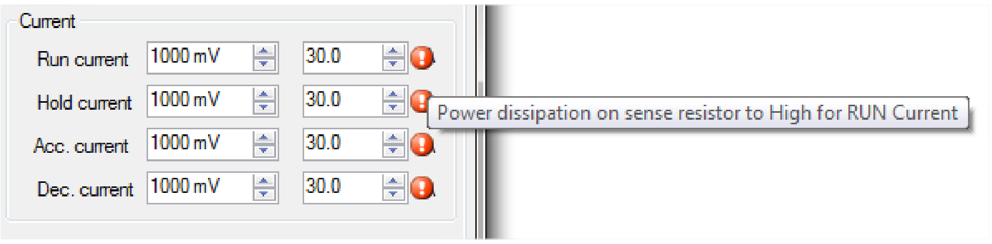
☑ Always show this wizard at startup

The wizard can modify existing configuration or create new one. When the configuration is finished, it can be send to the connected device and saved to a configuration file. For all panels, except the last **Apply and save settings**, clicking the **Next button** displays the next panel.

At any time you can return backward using the **Back** button.

If necessary, a check of the entered value is done. If there's an error, a popup window will display the error message and the next panel will be displayed only if the check is passed. If possible, numeric boxes provoking the error are identified with an exclamation point and the error is displayed when the mouse is on the exclamation icon, an example is shown in *Figure 17*.

Figure 17. Wizard error example



Current

Run current	1000 mV	30.0
Hold current	1000 mV	30.0
Acc. current	1000 mV	30.0
Dec. current	1000 mV	30.0

Power dissipation on sense resistor to High for RUN Current

- **Configuration**

In the wizard panel, you can choose an existing configuration file by clicking the **Browse** button. When the file is selected, use the **Load** button to load the file in the wizard. After loading the configuration, you can apply it to the current selected device. When the wizard is launched at the startup, the selected device is the first in the chain.

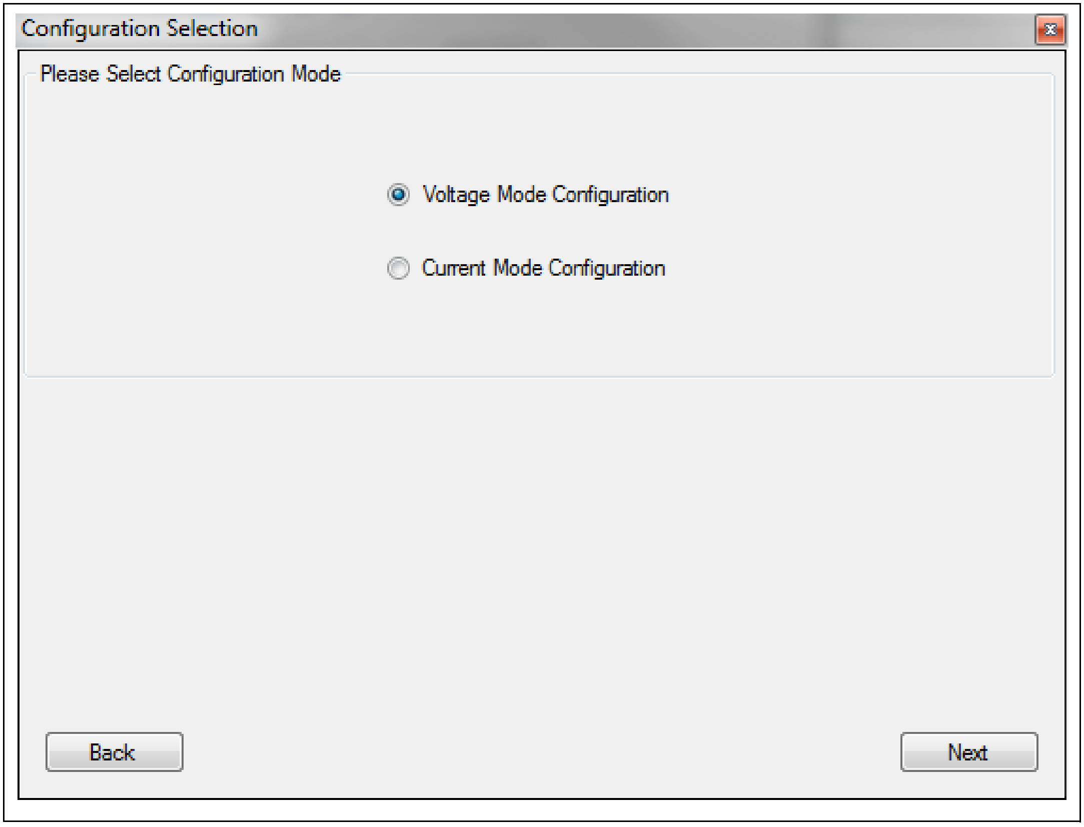
Note: If no configuration is set, the wizard will use the default values.

- **Startup option**

The Always show this wizard at startup checkbox allows disabling or enabling the launch of the wizard at the startup. When it is disabled, you can still launch the wizard clicking on the button in the toolbar or selecting **Wizard** in the **Tools** menu.

Only for the powerSTEP01 device it is possible to choose the Configuration Mode: a Voltage Mode or a Current Mode Configuration are available (see *Figure 18*).

Figure 18. Wizard configuration selection panel (powerSTEP01 only)

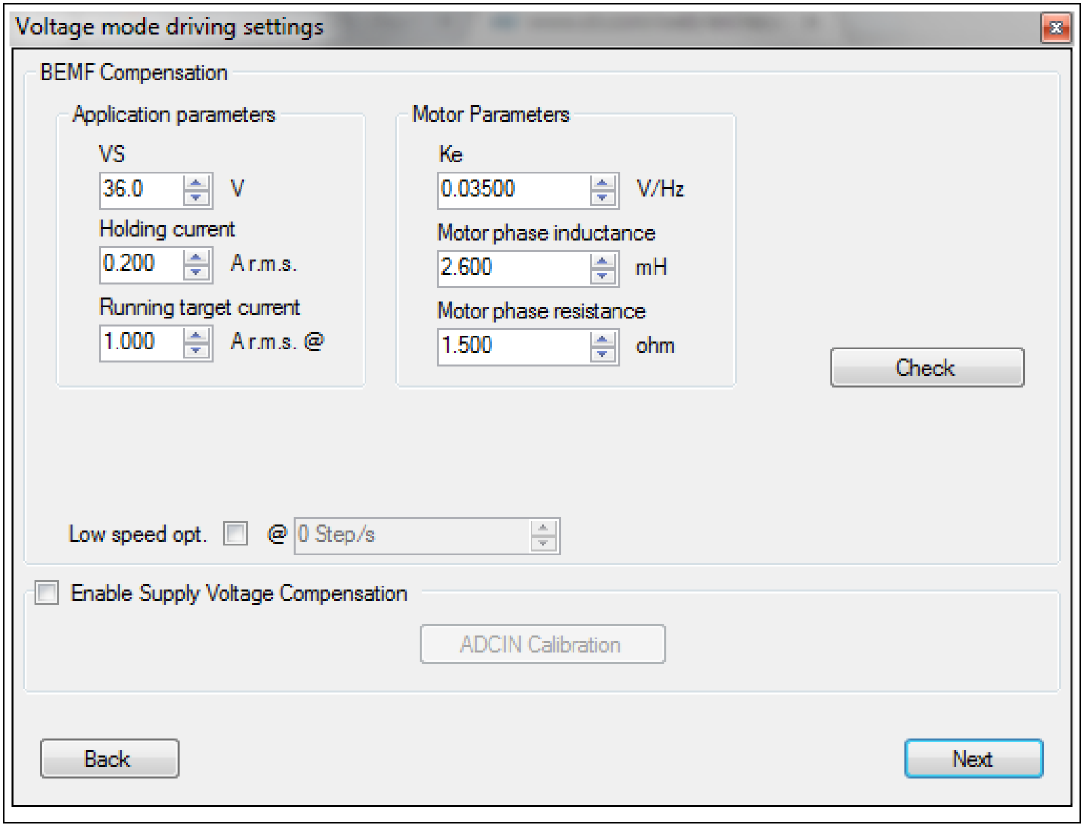


● Voltage Mode Configuration

○ Current Mode Configuration

3.3.1 Voltage mode driving settings panel

Figure 19. Wizard voltage mode driving panel



VS: 36.0 V

Holding current: 0.200 A r.m.s.

Running target current: 1.000 A r.m.s. @

Ke: 0.03500 V/Hz

Motor phase inductance: 2.600 mH

Motor phase resistance: 1.500 ohm

Low speed opt. □ @ 0 Step/s

□ Enable Supply Voltage Compensation

BEMF Compensation

- **Application parameters:**
  - **VS** is the supply voltage.
  - **Holding current** is the target r.m.s. current when the motor is stopped.
  - **Ke** is the electric constant of the motor. It can be measured through an oscilloscope.
- **Motor parameters**
  - Motor phase inductance is the inductance of the motor phases.
  - Motor phase resistance is the resistance of the motor phases.

By clicking the **Check** button, entered values are computed using internal algorithms, if result values are out of ranges, a popup window will be displayed.

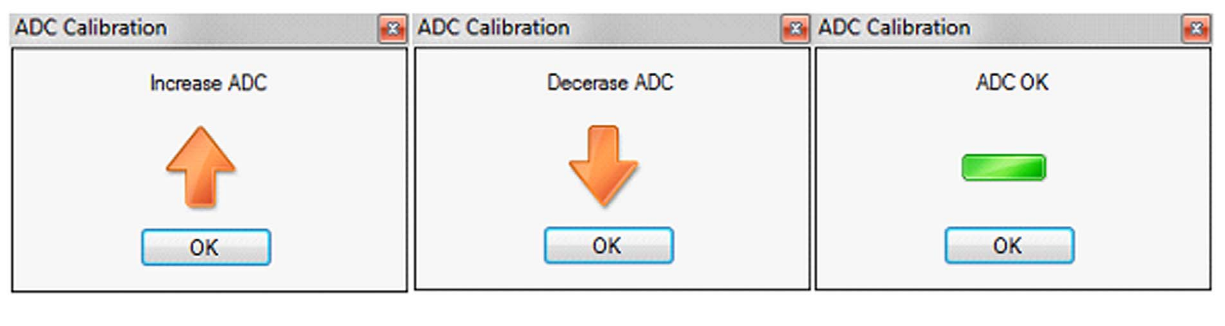
- **Low speed optimization**

If check, enables the Low Speed Optimization at the selected min. speed in the numeric box.

- **Supply Voltage Compensation**

The checkbox **Enable Supply Voltage Compensation** updates the *EN_VSCOMP* register, if checked; the motor supply voltage compensation is enabled and the **ADCIN Calibration** button is enable. Clicking the **ADCIN Calibration** button, a new window is open to help the user to calibrate the ADCIN.

Figure 20. ADCIN calibration



ADC Calibration

Increase ADC

OK

ADC Calibration

Decerase ADC

OK

ADC Calibration

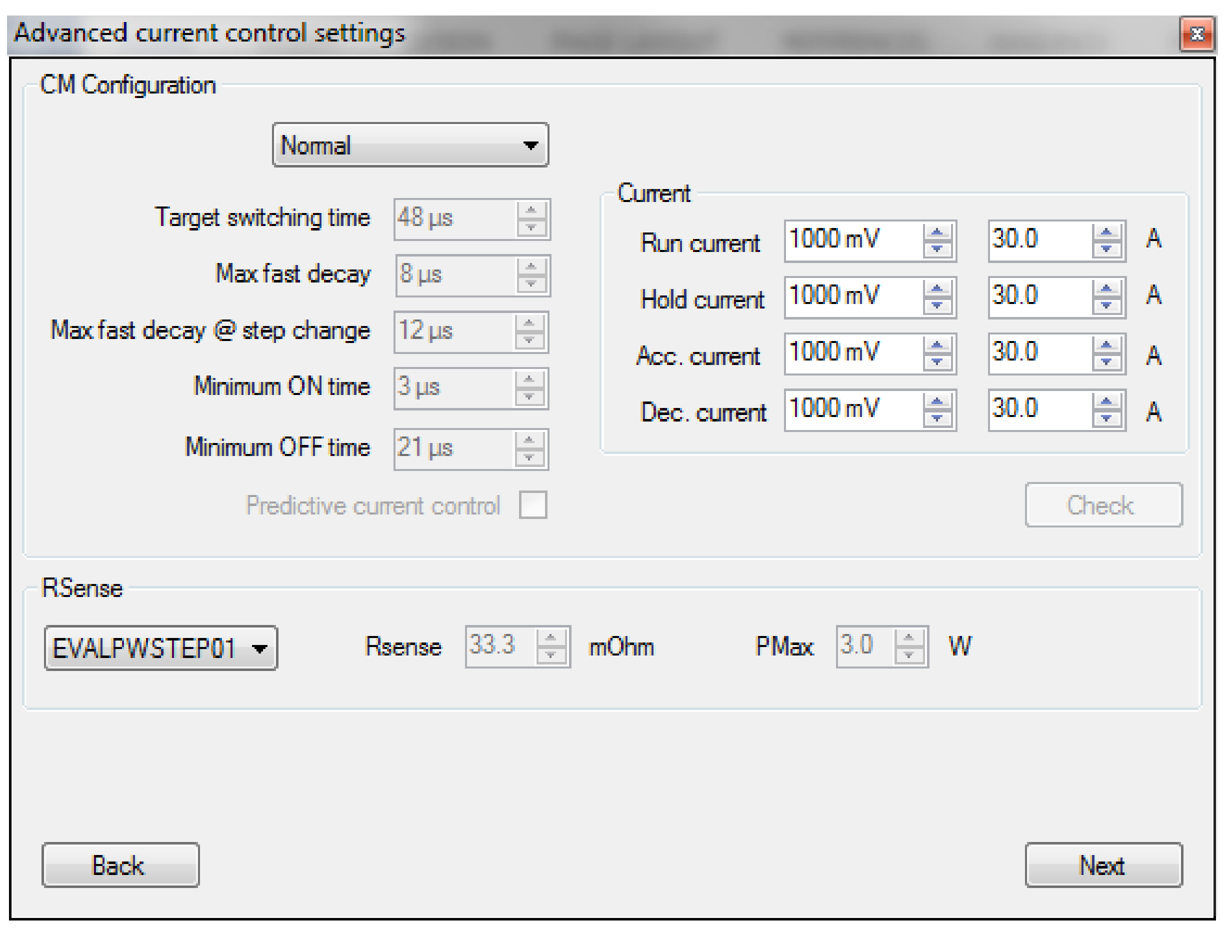
ADC OK

OK

The windows suggests how the voltage divider must be set by reading the ADC_OUT register. The register must be equal to 00010000.

3.3.2 Advanced current control panel

Figure 21. Advanced current control settings panel



Advanced current control settings

CM Configuration

Normal

Target switching time 48 µs

Max fast decay 8 µs

Max fast decay @ step change 12 µs

Minimum ON time 3 µs

Minimum OFF time 21 µs

Predictive current control □

Current

Run current	1000 mV	30.0	A
Hold current	1000 mV	30.0	A
Acc. current	1000 mV	30.0	A
Dec. current	1000 mV	30.0	A

Check

RSense

EVALPWSTEP01

Rsense 33.3 mOhm

PMax 3.0 W

Back

Next

- **CM Configuration**
  - **Minimum ON time** box contains the TON_MIN register value. Allowed formats are decimal (e.g. '3.5') and hexadecimal (e.g. '0x5').
  - **Minimum OFF time** box contains the TOFF_MIN register value. Allowed formats are decimal (e.g. '3.5') and hexadecimal (e.g. '0x5').
  - **Max fast decay and Max fast decay @ step change** boxes contain respectively the *TOFF_FAST* and the *FAST_STEP* parameters (T_FAST register). Allowed formats are decimal (e.g. '12') and hexadecimal (e.g. '0x9').
  - **Target switching time** box contains the *TSW* parameter of the CONFIG register. Allowed formats are decimal (e.g. '40') and hexadecimal (e.g. '0x36').
  - **Current**
    For the L6482 and powerSTEP01: **Acc. current, Dec. current, Run current** and **Hold current** boxes show the TVAL_X register values expressed in volts with the converted value in Ampere using the RSense value (Vth = RSense * Ith).
    For the L6472: **Acc. current, Dec. current, Run current** and **Hold current** are expressed in Amperes of the reference voltage.
    For the L6474: **Hole Current** boxes show the TVAL register in Amperes of the reference voltage.
  - **Predictive current control** (not applicable to the L6474)
    Checking the **Predictive current control** box the predictive mode will be enabled. Please, refer to the related datasheet section for more details.
- **RSense** (L6482 and powerSTEP01 only)

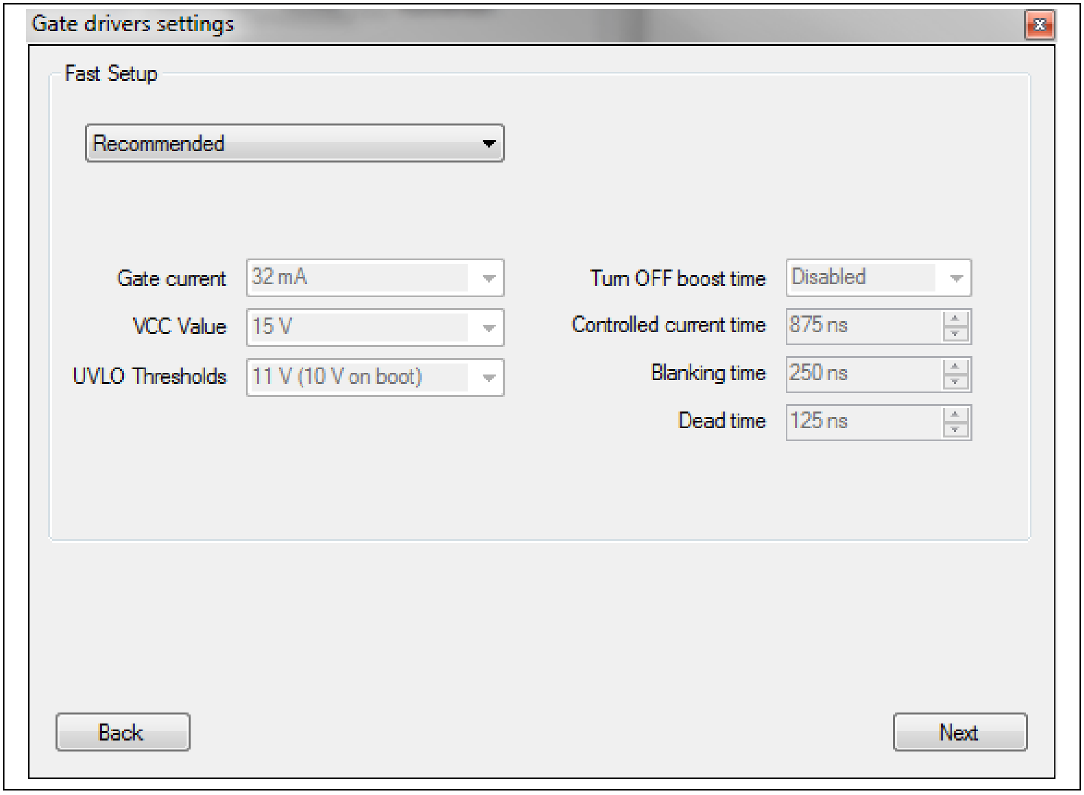
You can select a predefined value in the **RSense** list according to your board or select a custom value. **Pmax** is used to check the power dissipation on the sense resistor.

3.3.3 Gate drivers settings panel

This tab is used to configure the gate drivers of the device.

You can select a predefined configuration depending of the evaluation board connected or use custom values, in this case the **Check** button allows checking the entered value. For the L648x devices, in the MOSFET Characteristic group you can enter the specific parameters of the selected part number.

Figure 22. Gate drivers settings panel



Gate drivers settings

Fast Setup

Recommended

Gate current: 32 mA

VCC Value: 15 V

UVLO Thresholds: 11 V (10 V on boot)

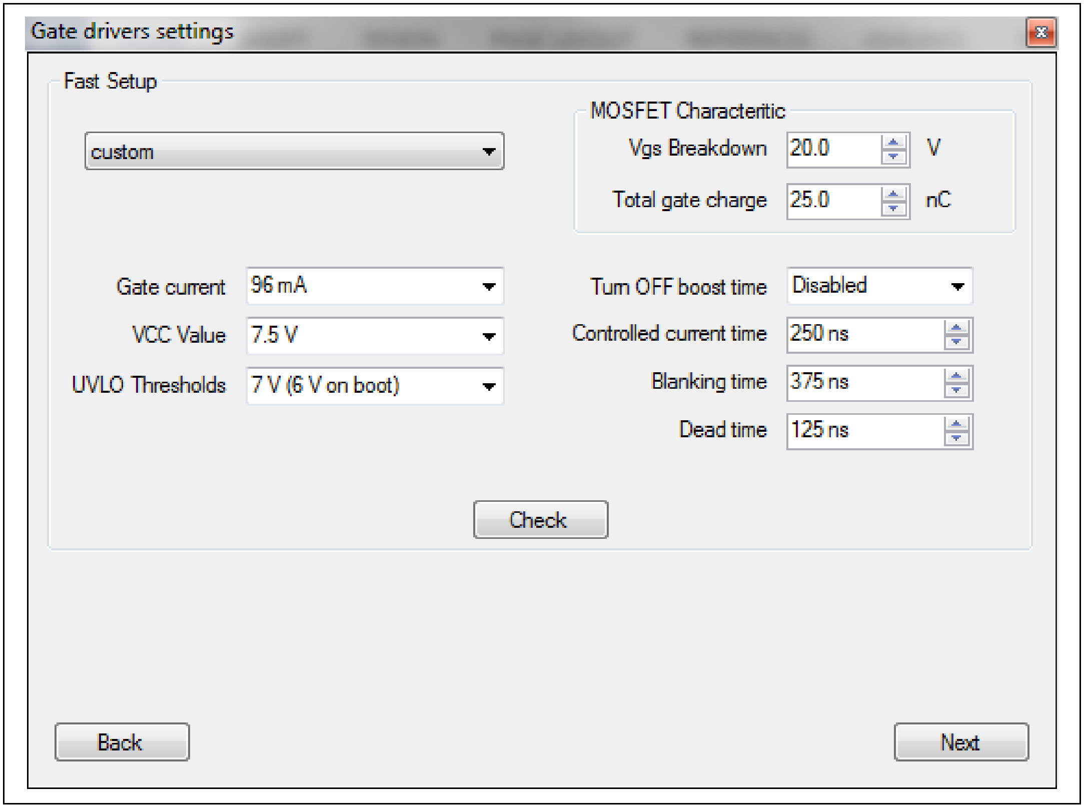
Turn OFF boost time: Disabled

Controlled current time: 875 ns

Blanking time: 250 ns

Dead time: 125 ns

Figure 23. Gate drivers settings panel (L648x only)



Gate drivers settings

Fast Setup

custom

MOSFET Characteritic

Vgs Breakdown: 20.0 V

Total gate charge: 25.0 nC

Gate current: 96 mA

VCC Value: 7.5 V

UVLO Thresholds: 7 V (6 V on boot)

Turn OFF boost time: Disabled

Controlled current time: 250 ns

Blanking time: 375 ns

Dead time: 125 ns

The **Gate current** drop box selects the gate current between the available values (*IGATE* parameter).

The **Controlled current time**, **Blanking time** and **Dead time** boxes contain the *TCC*, *TBLANK* and *TDT* parameters respectively. Allowed formats are decimal (e.g. '250') and hexadecimal (e.g. '0x1').

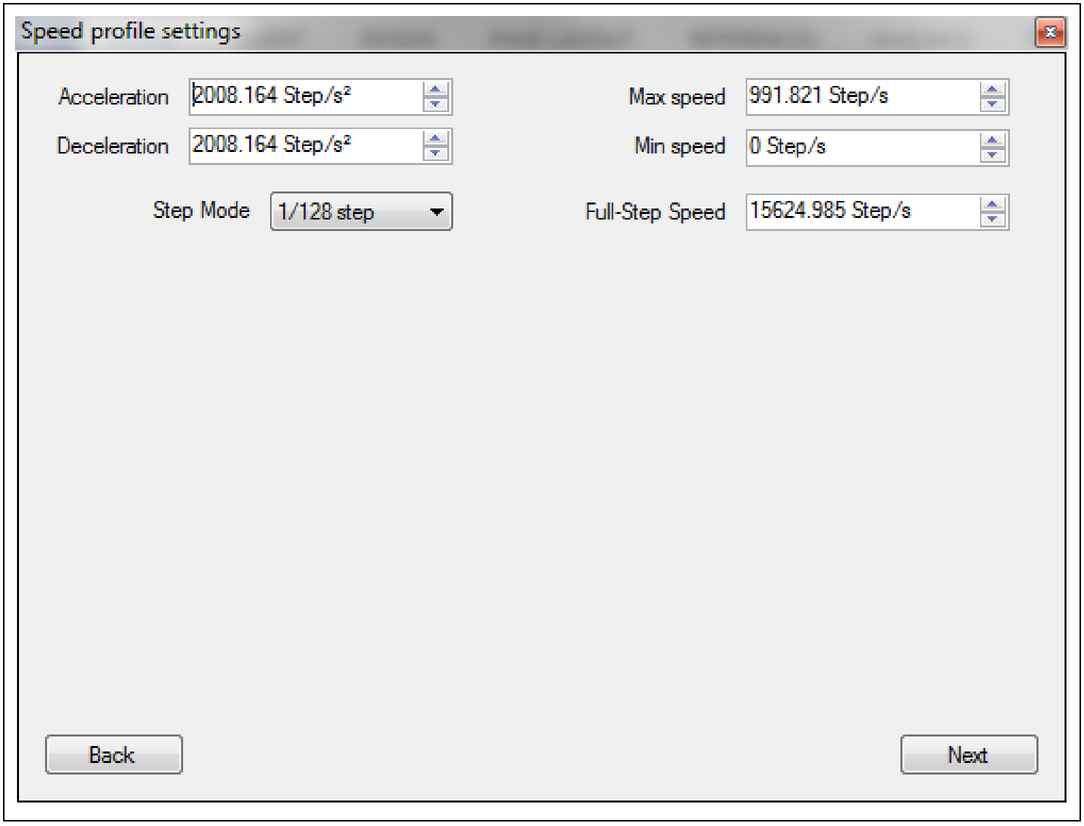
The **Turn OFF boost time** drop box enables the respective feature and selects the duration of the boost time (*TBOOST* parameter).

The **VCC value** and **UVLO thresholds** drop boxes select the output voltage of the VCC voltage regulator and set the UVLO protection thresholds.

3.3.4 Speed profile settings panel

This tab collects all the device parameters that are related to speed profile boundaries.

Figure 24. Wizard speed profile settings



Speed profile settings

Acceleration 2008.164 Step/s²

Deceleration 2008.164 Step/s²

Step Mode 1/128 step

Max speed 991.821 Step/s

Min speed 0 Step/s

Full-Step Speed 15624.985 Step/s

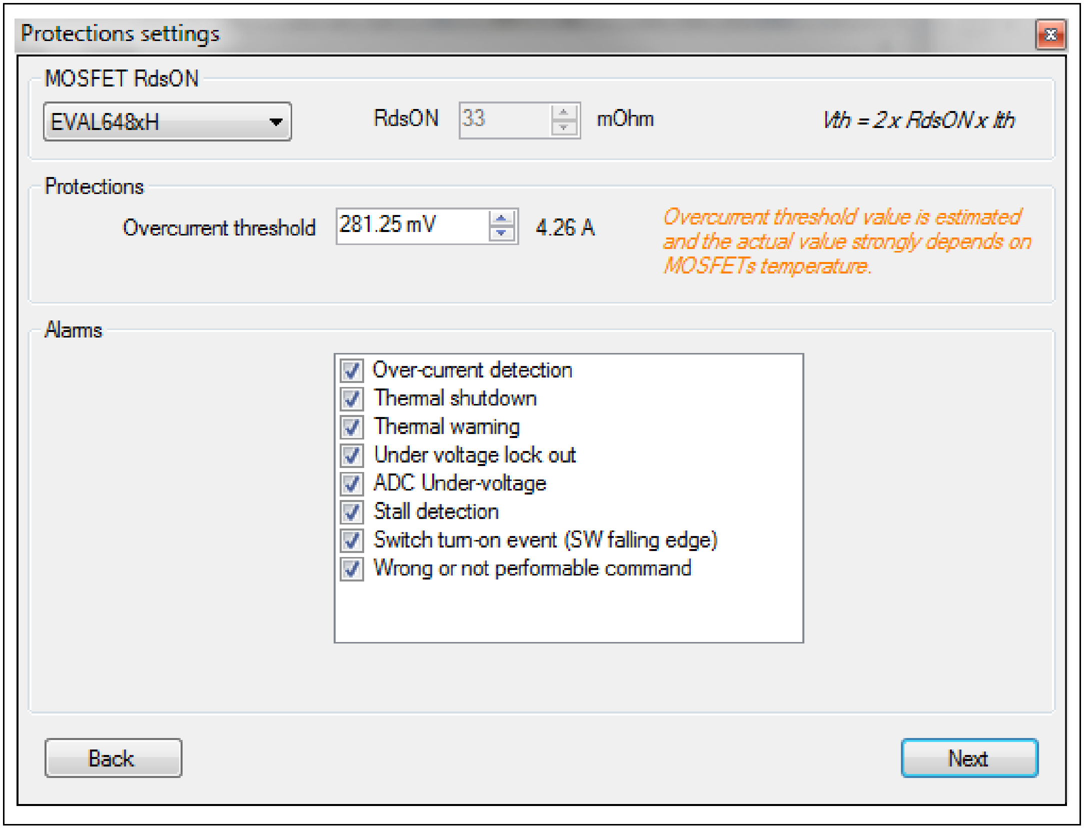
Note: *Full Step Speed is not available for the L6474 device.*

The values can be changed writing the new value in a common format (step/s^2 for acceleration/deceleration and step/s for speed) or in the hexadecimal format using the '0x' prefix. In the first case the value is rounded to the nearest available.

You can selected the current step resolution in the **Step Mode** drop-down menu.

3.3.5 Protection settings panel

Figure 25. Wizard protection setting panel



Protections settings

MOSFET RdsON

EVAL648xH

RdsON 33 mOhm

Vth = 2x RdsON x Ith

Protections

Overcurrent threshold 281.25 mV 4.26 A

Overcurrent threshold value is estimated and the actual value strongly depends on MOSFETs temperature.

Alarms

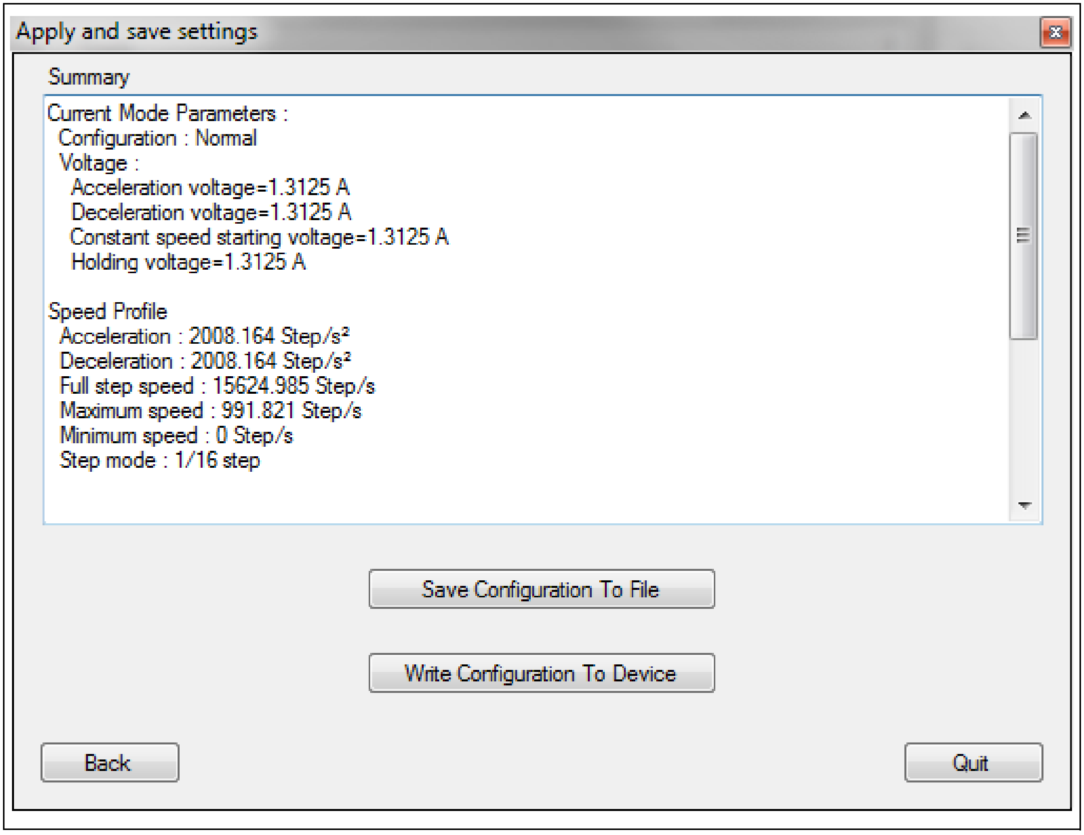
☑ Over-current detection
☑ Thermal shutdown
☑ Thermal warning
☑ Under voltage lock out
☑ ADC Under-voltage
☑ Stall detection
☑ Switch turn-on event (SW falling edge)
☑ Wrong or not performable command

- **MOSFET RdsON** (L648x and powerSTEP01 only)
  You can choose a predefined value of the RdsON by selecting the board in the list. If you select **Custom**, you will be able to enter a desired value in the **RdsON** numeric box.
- **Protections**
  For the L648x and powerSTEP01, the overcurrent threshold is entered in volt, the converted value depending on the RdsON is displayed at the left of the numeric box. For other devices, the overcurrent threshold is entered directly in Ampere.
- **Alarms**
  Select the error events that will cause the *FLAG* output to be forced low in the **Alarm** check list.

3.3.6 Apply and save settings panel

This last panel shows a summary of the values entered or computed from user input.

Figure 26. Wizard apply and save settings



```
Current Mode Parameters :
 Configuration : Normal
 Voltage :
  Acceleration voltage=1.3125 A
  Deceleration voltage=1.3125 A
  Constant speed starting voltage=1.3125 A
  Holding voltage=1.3125 A

Speed Profile
 Acceleration : 2008.164 Step/s²
 Deceleration : 2008.164 Step/s²
 Full step speed : 15624.985 Step/s
 Maximum speed : 991.821 Step/s
 Minimum speed : 0 Step/s
 Step mode : 1/16 step
```

You can save the configuration into a *.*sfc* file by clicking the **Save Configuration to File** button. A save dialog box will open and allows choosing the save filename.

The configuration can be sent to the device by clicking the **Write Configuration to Device** button. A popup window displays the result of the operation.

The **Quit button** closes the wizard, a popup window warns the user if the configuration has not be saved.

3.4 Register map

The register map tool gives an overview of the current device register values. Clicking on the icon in the toolbar or selecting **Register map** in the **Tools** menu opens the **Register Map** panel.

If more than one device is connected to the communication board, the active device can be selected through the drop list on the top right corner of the toolbar.

The registers are represented as rows of a table with information stored in different columns:

- **Name**: contains the register mnemonic name.
- **Address**: indicates the register address in the hexadecimal format.
- **Description**: contains a brief description of the register content.
- **Value**: shows the current register “decoded” value.
- **Hex**: reports the value of the register in the hexadecimal format. This is the only writable column and it can be used to change the register value.
- **Default**: reports the default value of the register

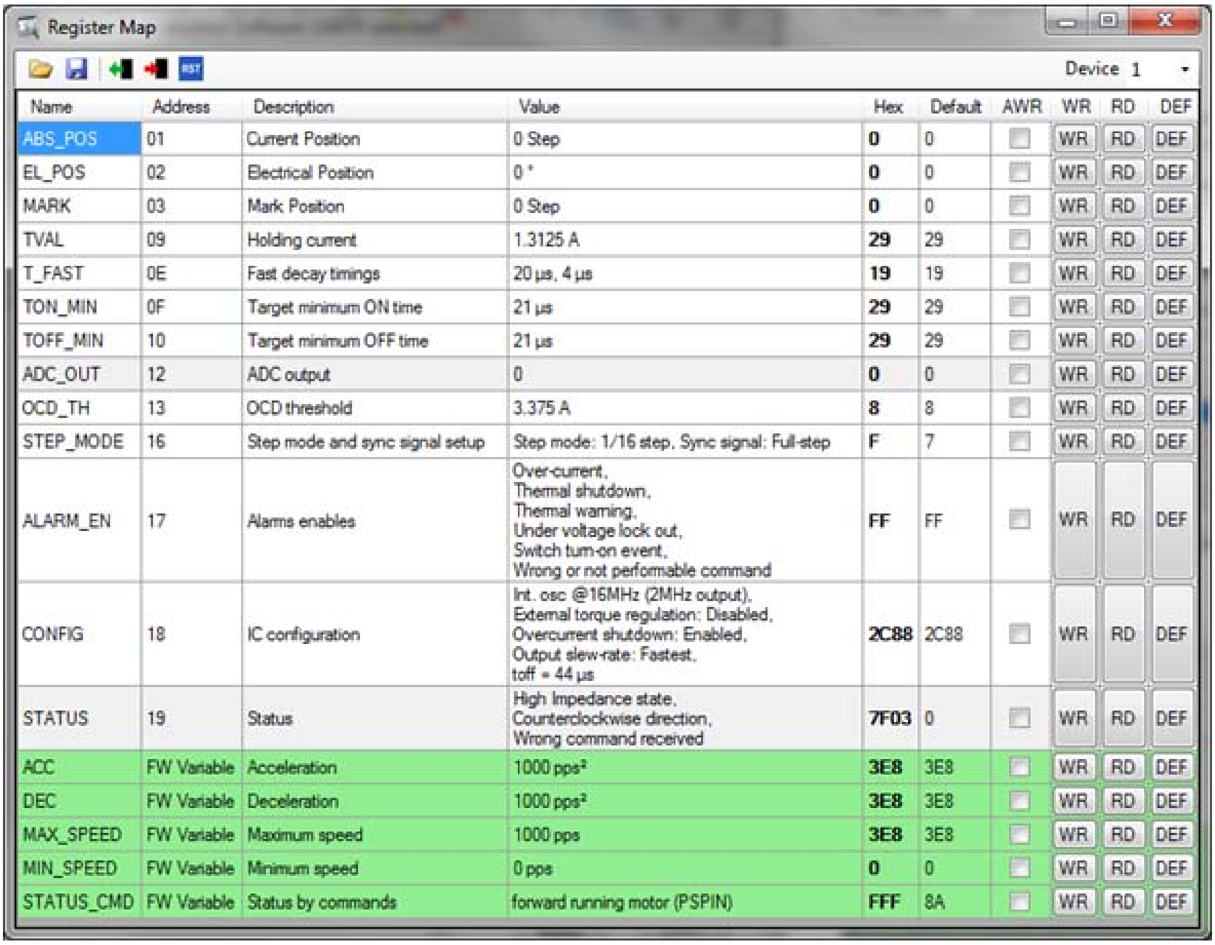
There are also columns that allow reading/writing registers:

- Clicking on the **WR** column buttons write the register value.
- Clicking on the **RD** column buttons read the register value.
- Clicking on the **DEF** column buttons set the register value to its default value.
- Checking the A**WR** boxes automatically write the register value every time its value in the **Hex** column is changed.

Row colors are used to identify register status:

- Black text on the white background indicates that the register is writable and that the displayed value is updated (*GetParam* or *SetParam* commands refreshed the **Hex** column value).
- Red text on the white background indicates that the **Hex** column value has been changed by the user but not written yet.
- Black text on the grey background indicates read-only registers.
- Black text on the green background indicates that the register is not really an IC register but a parameter in the FW that acts like a register. For example (see *Figure 27*), the STATUS register does not exist in the L6474, to ease the management of the status a fake STATUS_CMD allows to easily get a status of the chip.

Figure 27. Register Map (L6474 example)



Name	Address	Description	Value	Hex	Default	AWR	WR	RD	DEF
ABS_POS	01	Current Position	0 Step	0	0	□	WR	RD	DEF
EL_POS	02	Electrical Position	0 °	0	0	□	WR	RD	DEF
MARK	03	Mark Position	0 Step	0	0	□	WR	RD	DEF
TVAL	09	Holding current	1.3125 A	29	29	□	WR	RD	DEF
T_FAST	0E	Fast decay timings	20 µs, 4 µs	19	19	□	WR	RD	DEF
TON_MIN	0F	Target minimum ON time	21 µs	29	29	□	WR	RD	DEF
TOFF_MIN	10	Target minimum OFF time	21 µs	29	29	□	WR	RD	DEF
ADC_OUT	12	ADC output	0	0	0	□	WR	RD	DEF
OCD_TH	13	OCD threshold	3.375 A	8	8	□	WR	RD	DEF
STEP_MODE	16	Step mode and sync signal setup	Step mode: 1/16 step, Sync signal: Full-step	F	7	□	WR	RD	DEF
ALARM_EN	17	Alarms enables	Over-current, Thermal shutdown, Thermal warning, Under voltage lock out, Switch turn-on event, Wrong or not perfomable command	FF	FF	□	WR	RD	DEF
CONFIG	18	IC configuration	Int. osc @16MHz (2MHz output), External torque regulation: Disabled, Overcurrent shutdown: Enabled, Output slew-rate: Fastest, toff = 44 µs	2C88	2C88	□	WR	RD	DEF
STATUS	19	Status	High Impedance state, Counterclockwise direction, Wrong command received	7F03	0	□	WR	RD	DEF
ACC	FW Variable	Acceleration	1000 pps^2	3E8	3E8	□	WR	RD	DEF
DEC	FW Variable	Deceleration	1000 pps^2	3E8	3E8	□	WR	RD	DEF
MAX_SPEED	FW Variable	Maximum speed	1000 pps	3E8	3E8	□	WR	RD	DEF
MIN_SPEED	FW Variable	Minimum speed	0 pps	0	0	□	WR	RD	DEF
STATUS_CMD	FW Variable	Status by commands	forward running motor (PSPIN)	FFF	8A	□	WR	RD	DEF

Warning: **Reading/writing rows are only displayed when the communication board is correctly connected to the PC. They will be hidden also when a script code is running.**

3.4.1 Writing registers

Single registers can be written changing their value in the **Hex** column (hexadecimal format) and clicking on the **WR** button.

When the **Hex** value is changed the **Value** column is immediately updated showing the new decoded value, but the register is not written. This situation is indicated by the row text color in red. This way you can “preview” how the new register value will affect the device configuration without make this change effective.

You can set if a specific register has to be automatically written every time the **Hex** value is changed checking the respective **AWR** column box.

Warning: When auto-write (AWR) is enabled, any accidental change in the Hex column will cause the respective register be written.

You can also write the whole register map by clicking on the ➡ button. This way all registers (whatever their value has been changed or not) will be written according to the current **Hex** column value.

Clicking on the RST button all registers (whatever their value has been changed or not) will be set to the respective default value.

Read-only registers are indicated with black text on the grey background rows.

Warning: If the ABS_POS register autorefresh in the main form is enabled, the ABS_POS row is updated at the current polling rate. This makes difficult to set the ABS_POS register to the desired value.
If you want to change the ABS_POS register value, please disable the Autorefresh option first.

3.4.2 Reading registers

Single registers can be read clicking on the **RD** button. The **Hex** and **Value** columns will be updated according to the current register value. Unwritten changes (red text) will be rejected (black text).

You can also read the whole register map by clicking on the ⬅ button. This way all registers will be read updating **Hex** and **Value** columns.

3.4.3 Loading and saving configurations

Clicking on the 💾 button saves the current device configuration. A file selection dialog will be opened to choose if creating a new configuration file (*.dsc) or overwriting an existing one.

Warning: Saved values are the Hex column ones, whatever they are actually written into the device or not.

Configuration files can be loaded by clicking on the 📂 button; it opens a selection dialog to choose the configuration file.

When it is loaded, the **Hex** column is updated, but no value is written into the device (changed registers will be highlighted with red text) even if the respective **AWR** box is checked. To write the configuration you should use the ➡ button or the **WR** column.

3.5 Device configuration

This form allows configuring the connected devices. All registers content is displayed through a user-friendly interface and the parameters are converted in a common format.

The device configuration form can be opened from the application main form toolbar clicking on the 🛠 button or selecting the **Tools > Device configuration** menu.

If more than one device is connected to the communication board, the active device can be selected through the drop list on the top right corner of the toolbar.

The register map dialog while be updated accordingly to updated value in this form.

3.5.1 Open and save device configuration

Configuration files can be loaded clicking on the 📂 button. A file selection dialog will be opened to choose the configuration file.

Warning: New configuration is NOT written into the device. To write the new values the ➡ button should be used.

Clicking on the 💾 button saves the current device configuration. A file selection dialog will be opened where you can choose to create a new configuration file (*.sfc) or overwrite an existing one.

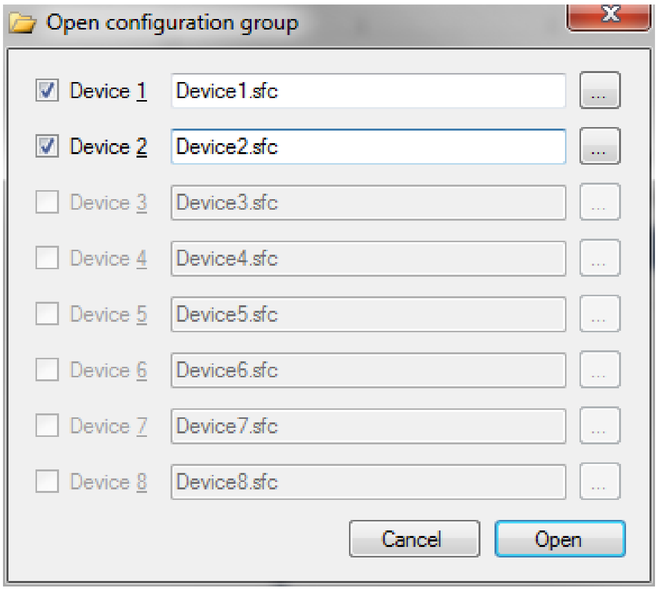
Warning: The configuration file includes ALL writable registers (ABS_POS, MARK, etc.). Make sure that the value of these registers is coherent with the desired one before saving the configuration.

Opening and saving multiple device configurations

From the main form, it is possible to save and set the configuration of all the devices in the chain at the same time.

Clicking on the 📂 button or selecting the **File > Open** menu, a dialog similar to *Figure 28* is shown.

Figure 28. Open configuration group dialog box



Open configuration group

☑ Device 1 Device1.sfc

☑ Device 2 Device2.sfc

□ Device 3 Device3.sfc

□ Device 4 Device4.sfc

□ Device 5 Device5.sfc

□ Device 6 Device6.sfc

□ Device 7 Device7.sfc

□ Device 8 Device8.sfc

Following these steps:

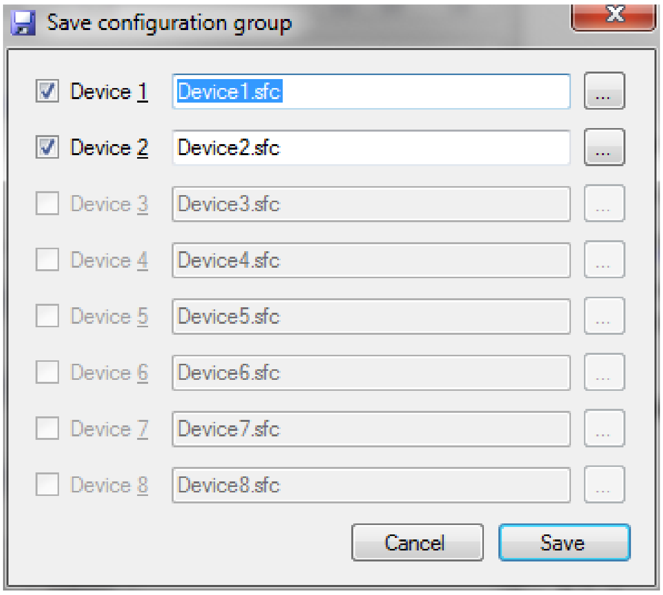
- Select which devices have to be configured checking the related box
- Write the target file path in the text box or browse it clicking on the "**...**" button
- Click on the **OK** button.

If one or more of the file does not exist an error message is shown. The new configurations are immediately written into the devices.

Warning: The information included into a configuration file is strictly related to the respective device (L6470, L6472, L6480 or L6482). Any attempt to open a configuration file of a device type into another one will cause an error.

Clicking on the button or selecting the **File > Save** menu, a dialog similar to *Figure 29* is shown.

Figure 29. Save configuration group dialog box



Save configuration group

☑ Device 1	Device1.sfc	...
☑ Device 2	Device2.sfc	...
□ Device 3	Device3.sfc	...
□ Device 4	Device4.sfc	...
□ Device 5	Device5.sfc	...
□ Device 6	Device6.sfc	...
□ Device 7	Device7.sfc	...
□ Device 8	Device8.sfc	...

Following these steps:

- Select which devices have to be configured checking the related box
- Write the target file path in the text box or browse it clicking on the "**...**" button
- Click on the **OK** button.

If one or more of the file already exist a warning message is shown. If one or more of the file is read-only an error message is shown.

All the writable registers of the selected devices are updated before saving the configuration.

3.5.2 Write and Read configuration

Changing the form control values does not modify the actual value of the device registers.

Configuration is written into the device and changes are made effective by clicking on the button or clicking on the **Apply** button.

Clicking on the **OK** button the new configuration is written into the selected device and the form is closed. Clicking on the "**Cancel**" button the form is closed without writing the new configuration into the selected device.

Actual configuration can be loaded clicking on the button.

Default device configuration can be set clicking on the RST button.

3.5.3 Speed Profile tab

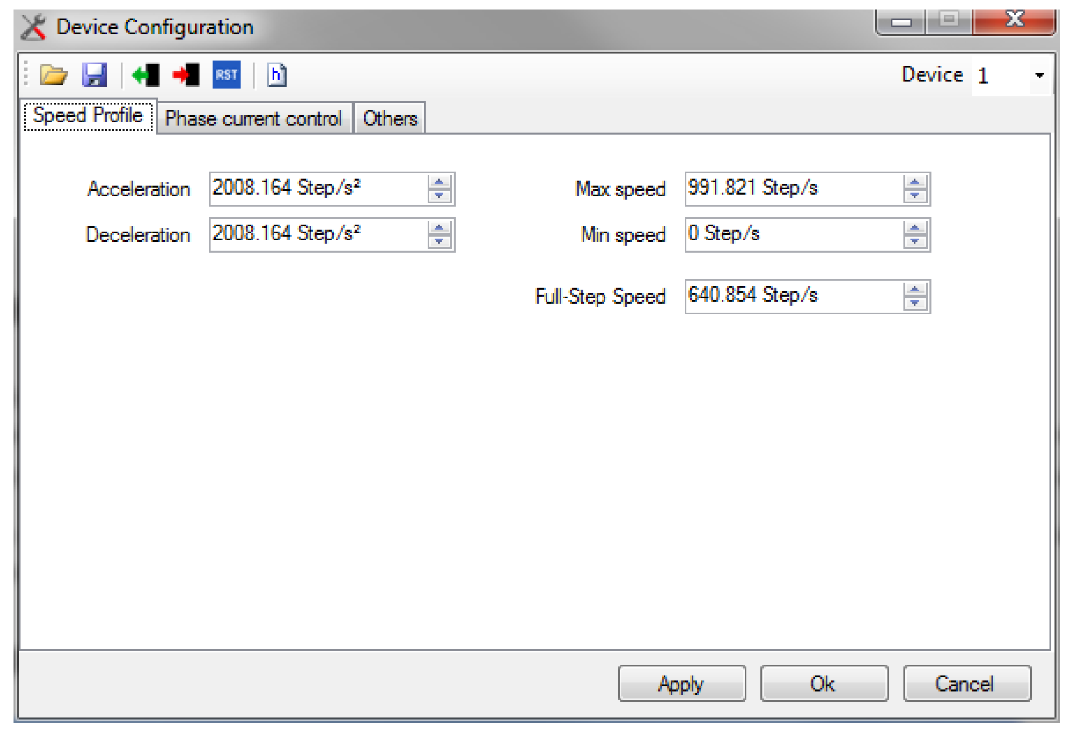
This tab collects all the device parameters that are related to speed profile boundaries.

The values can be changed writing the new value in a common format (step/s^2 for acceleration/deceleration and step/s for speed) or in the hexadecimal format using the '0x' prefix. In the first case the value is rounded to the nearest available. If a value greater than the maximum is written, the change is ignored.

For example, the **Acceleration** can be set to 2008.164 step/s^2 writing '2010' or '0x8A'.

Clicking on up and down buttons the parameter value is increased or decreased of one unit according to its resolution.

Figure 30. Speed Profile tab



Device Configuration

Device 1

Speed Profile | Phase current control | Others

Acceleration: 2008.164 Step/s²

Deceleration: 2008.164 Step/s²

Max speed: 991.821 Step/s

Min speed: 0 Step/s

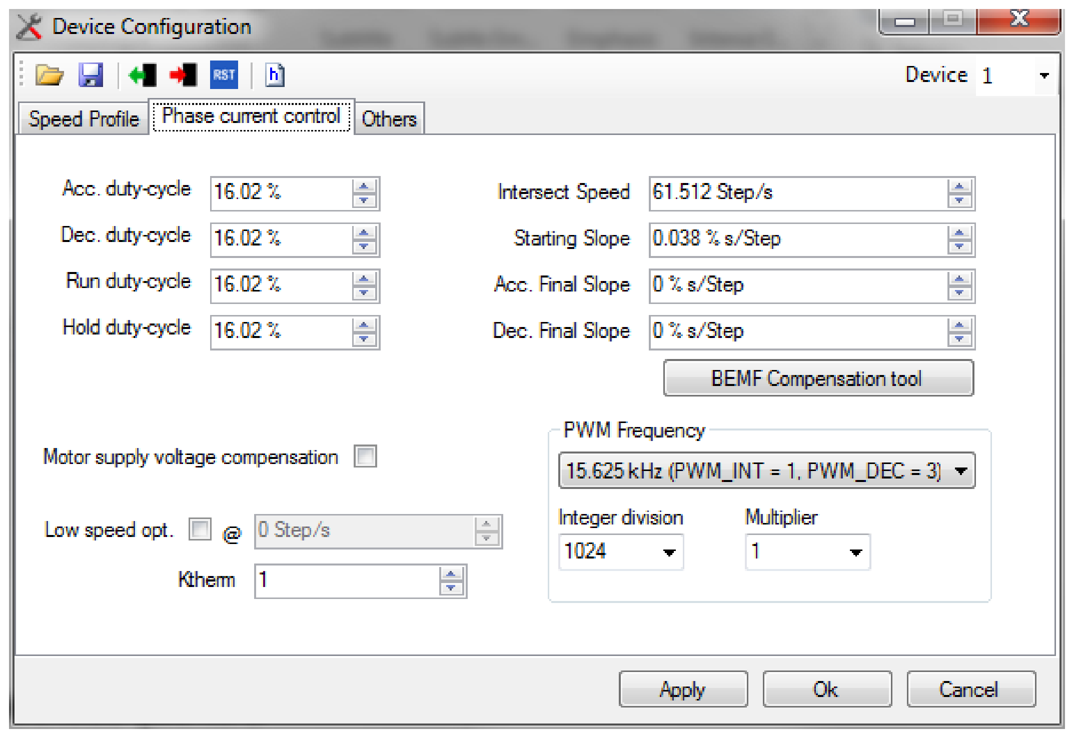
Full-Step Speed: 640.854 Step/s

Note: Full-Step Speed is NOT available for the L6474 device.

3.5.4 Phase current control tab (L6470, L6480 and powerSTEP01 in VM)

This tab lists all the parameters related to the voltage mode control. Values in numeric boxes can be changed writing the new value or using up/down arrows.

Figure 31. Phase current control tab (L6470, L6480 and powerSTEP01 in VM)



Device Configuration

Device 1

Acc. duty-cycle 16.02 %

Dec. duty-cycle 16.02 %

Run duty-cycle 16.02 %

Hold duty-cycle 16.02 %

Intersect Speed 61.512 Step/s

Starting Slope 0.038 % s/Step

Acc. Final Slope 0 % s/Step

Dec. Final Slope 0 % s/Step

Motor supply voltage compensation □

Low speed opt. □ @ 0 Step/s

Ktherm 1

PWM Frequency 15.625 kHz (PWM_INT = 1, PWM_DEC = 3)

Integer division 1024

Multiplier 1

- **Duty-cycle**

The **Acc. duty-cycle**, **Dec. duty-cycle**, **Run duty-cycle** and **Hold duty-cycle** boxes show *KVAL_X* register values expressed in the percentage format. Allowed formats are decimal (e.g. '0.25'), percentage (e.g. '25 %') and hexadecimal (e.g. '0x40').

- **BEMF**

The **Intersect Speed** box contains the INT_SPEED register value. Allowed formats are decimal (e.g. '230.5') and hexadecimal (e.g. '0x53A').

The **Starting Slope**, **Acc. Final Slope** and **Dec. Final Slope** correspond to BEMF compensation slopes. Allowed formats are decimal (e.g. '0.00048') and hexadecimal (e.g. '0x1F').

Clicking on the **BEMF Compensation tool** button opens the BEMF compensation tool (see *Section 3.6 on page 50*).

- **PWM Frequency**

The PWM frequencies are listed according to the oscillator frequency, *F_PWM_INT* and *F_PWM_DEC* values. The drop-down menu shows the current frequency, the Integer division and the Multiplier text box show *F_PWM_INT* and *F_PWM_DEC* values.

Changing the selected frequency in the drop-down menu will update the Integer division and Multiplier text box. Changing the Integer division and Multiplier text box will update the frequency in the combo-box.

- **Motor supply voltage compensation**

The checkbox updates the *EN_VSCOMP* bit in the register. This bit sets if the motor supply voltage compensation is enabled or not.

- **Low speed optimization**

If checked, enable the Low speed optimization at the selected minimum speed in the numeric box.

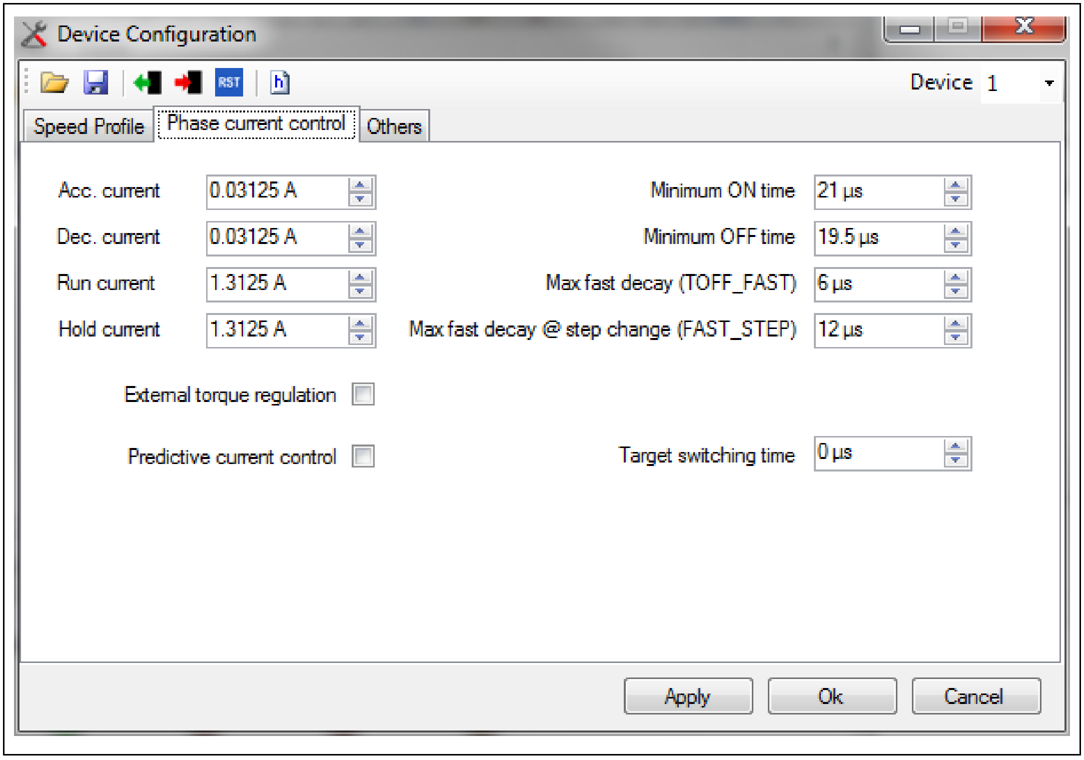
- **Ktherm**

This setting corresponds to the *K_THERM* register, it is used to compensate the phase resistance increment due to the temperature rising.

3.5.5 Phase current control tab (L6472, L6482, L6474, and powerSTEP01 in CM)

All the parameters related to the advanced current control are listed in this tab. Values in numeric boxes can be changed writing the new value or using up/down arrows.

Figure 32. Phase current control tab for current mode devices (L6472, L6482 and powerSTEP01 in CM)



Device 1

Acc. current 0.03125 A

Dec. current 0.03125 A

Run current 1.3125 A

Hold current 1.3125 A

External torque regulation □

Predictive current control □

Minimum ON time 21 µs

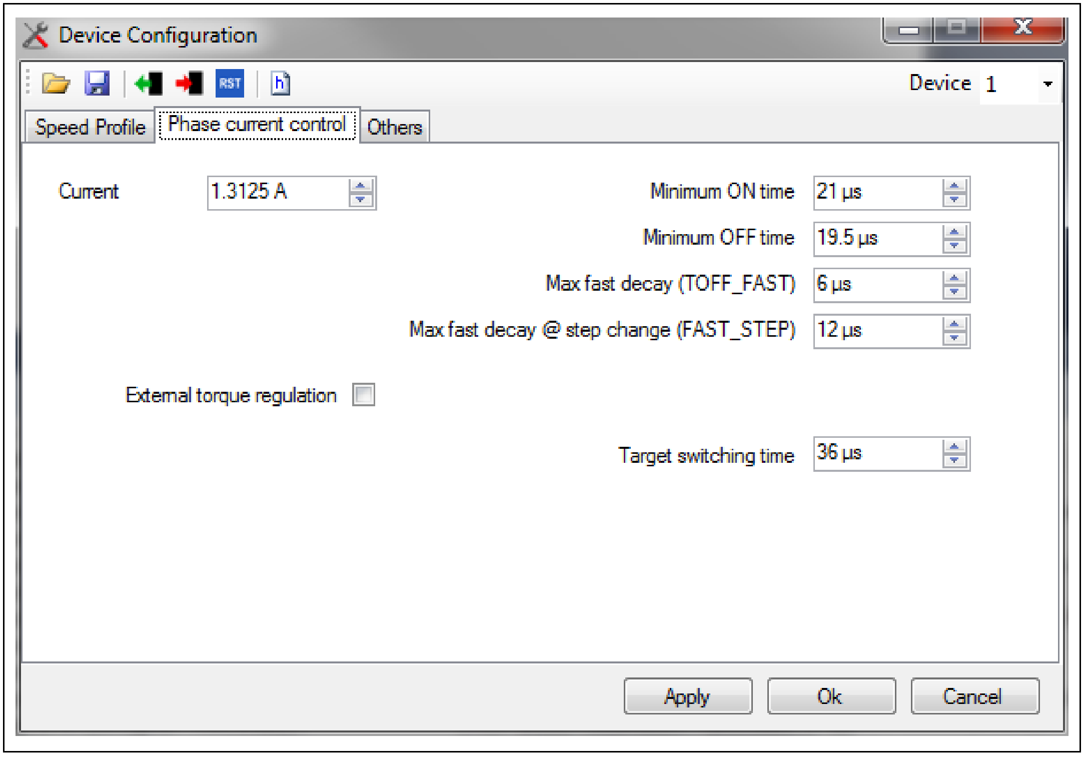
Minimum OFF time 19.5 µs

Max fast decay (TOFF_FAST) 6 µs

Max fast decay @ step change (FAST_STEP) 12 µs

Target switching time 0 µs

Figure 33. Phase current control tab for current mode device (L6474)



Current: 1.3125 A

Minimum ON time: 21 µs

Minimum OFF time: 19.5 µs

Max fast decay (TOFF_FAST): 6 µs

Max fast decay @ step change (FAST_STEP): 12 µs

External torque regulation □

Target switching time: 36 µs

- **Current**

For the L6472, L6482, L6474 and powerSTEP01, the **Acc. current**, **Dec. current**, **Run current** and **Hold current** boxes show TVAL_X register values expressed in Amperes or the reference voltage expressed in Volts in case of the L6482 and powerSTEP01 device. Allowed formats are decimal (e.g. '0.25') and hexadecimal (e.g. '0x40').

For the L6474, current boxes show the TVAL register (holding current).

- **Decay**

The **Minimum ON time** and **Minimum OFF time** boxes contain respectively the *TON_MIN* and *TOFF_MIN* registers value. Allowed formats are decimal (e.g. '3.5') and hexadecimal (e.g. '0x5').

The **Max fast decay** and **Max fast decay @ step change** boxes contain respectively the *TOFF_FAST* and *FAST_STEP* parameters (*T_FAST* register). Allowed formats are decimal (e.g. '12') and hexadecimal (e.g. '0x9').

- **Target switching time**

The **Target switching time** box contains the *TSW* parameter of the *CONFIG* register. Allowed formats are decimal (e.g. '40') and hexadecimal (e.g. '0x36').

- **External torque regulation**

Checking the **External torque regulation** box the device will use the *ADCIN* voltage to set the *TVAL* value instead of the respective registers. In this case please check that the *ADCIN* input is correctly driven.

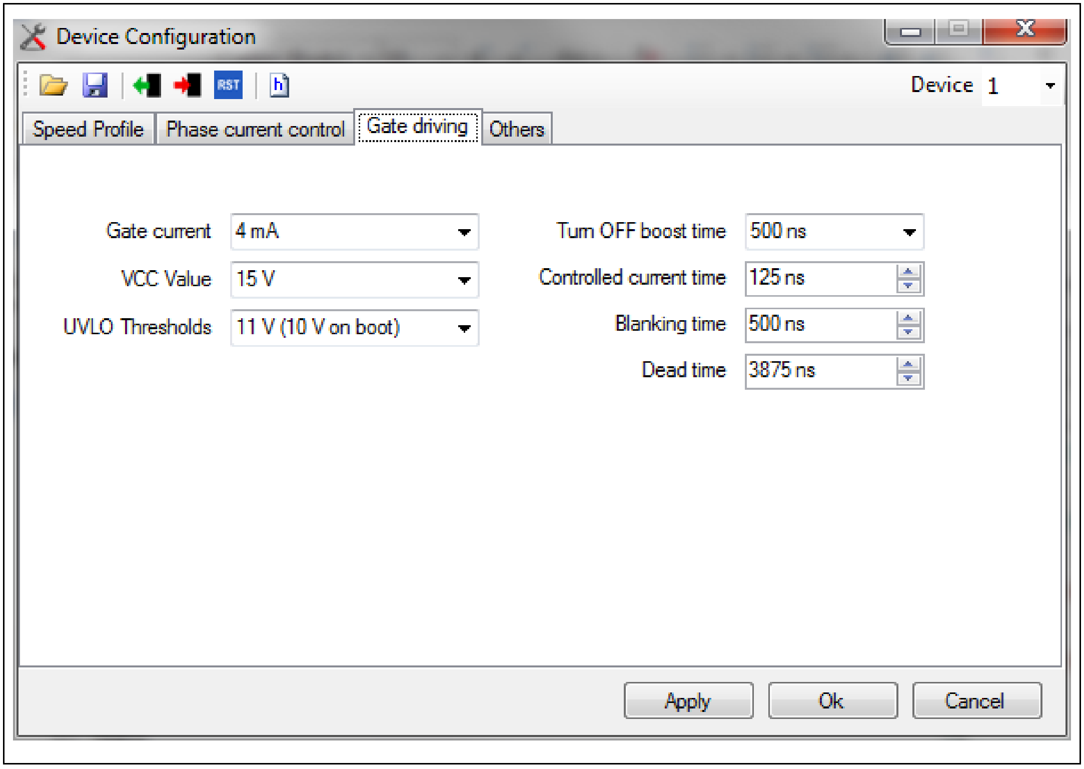
- **Predictive current control (not available for L6474)**

Checking the **Predictive current control** box the predictive mode will be enabled. Please, refer to the related datasheet section for more details.

3.5.6 Gate driving tab (only for L6480, L6482 and powerSTEP01)

This tab is used to configure the gate drivers of the device.

Figure 34. Gate driving tab



Device 1

Gate current 4 mA

VCC Value 15 V

UVLO Thresholds 11 V (10 V on boot)

Turn OFF boost time 500 ns

Controlled current time 125 ns

Blanking time 500 ns

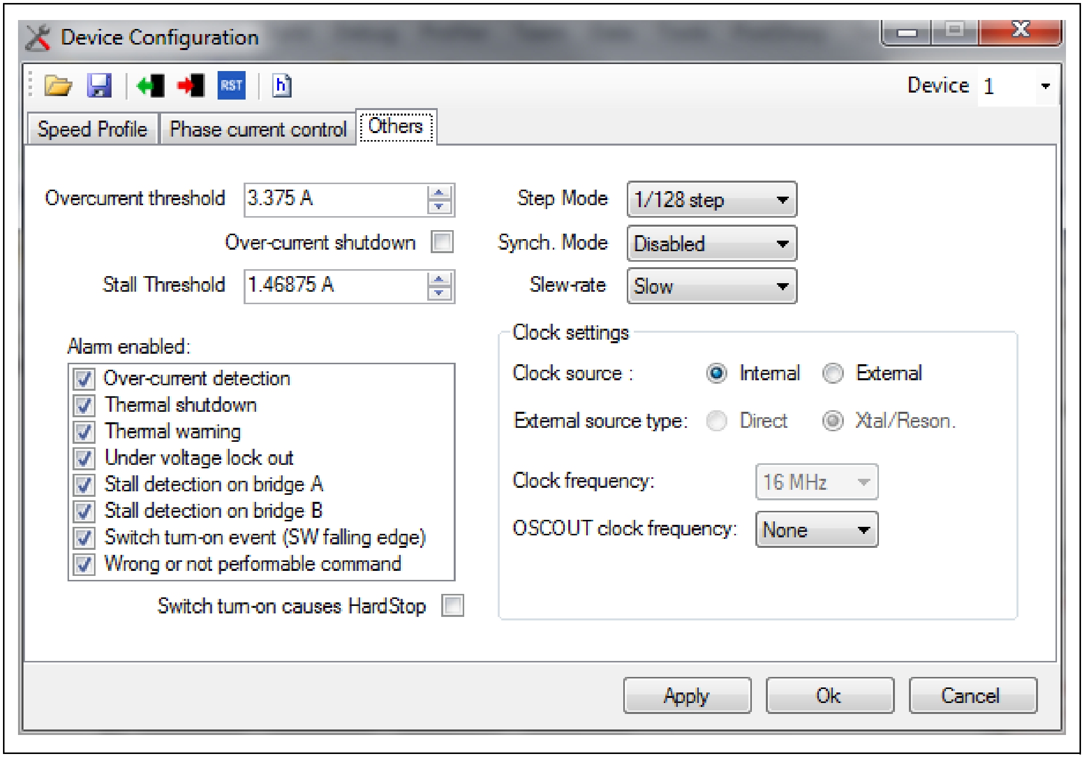
Dead time 3875 ns

- The **Gate current** drop box selects the gate current between the available values (*IGATE* parameter).
- The **VCC value** and **UVLO thresholds** drop boxes select the output voltage of the VCC voltage regulator and set the correspondent UVLO protection thresholds.
- The **Controlled current time**, **Blanking time** and **Dead time** boxes contain the *TCC, TBLANK* and *TDT* parameters respectively. Allowed formats are decimal (e.g. '250') and hexadecimal (e.g. '0x1').
- The **Turn OFF boost time** drop box enables the respective feature and selects the duration of the boost time (*TBOOST* parameter).

3.5.7 Others tab

This tab is used to configure others device features.

Figure 35. Others tab



Device 1

Overcurrent threshold 3.375 A

Over-current shutdown ☐

Stall Threshold 1.46875 A

Step Mode 1/128 step

Synch. Mode Disabled

Slew-rate Slow

Alarm enabled:

- ☑ Over-current detection
- ☑ Thermal shutdown
- ☑ Thermal warning
- ☑ Under voltage lock out
- ☑ Stall detection on bridge A
- ☑ Stall detection on bridge B
- ☑ Switch turn-on event (SW falling edge)
- ☑ Wrong or not performable command

Switch turn-on causes HardStop ☐

Clock settings

Clock source : ● Internal ○ External

External source type: ○ Direct ● Xtal/Reson.

Clock frequency: 16 MHz

OSCOUT clock frequency: None

- **Thresholds**

The **Overcurrent** detection and **stall** detection thresholds can be set through the respective numeric box. Values can be changed by writing the new value in the decimal or in the hexadecimal format using the '0x' prefix. Up and down arrows can be used also.

If the **Overcurrent shutdown** box is checked the overcurrent events will cause the power stage bridges to turn-off.

- **Step Mode**

You can selected the current step resolution in the **Step Mode** list and configure and enable the synchronization signal (BUSY/SYNC device output) in the **Synch. Mode** list.

The output slew-rate is selected using the **Slew-rate** list.

- **Alarms**

Select the error events that will cause the FLAG output to be forced low in the **Alarm enabled** check list.

To enable *HardStop* interrupt on the SW input falling edge the **Switch turn-on causes HardStop** box needs to be checked, otherwise the SW input does not interfere with the motor motion.

- **Clock settings**

Device clock configuration is shown on a dedicated panel: all possible clock configuration can be set.

The **Clock source: Internal** enables the integrated 16 MHz oscillator, **External** will use an external clock source.

External source type: Direct if the clock is directly applied to the *OSCIN* input, **Xtal/Reson**. if a resonator or a crystal have been connected.

Clock frequency: selects the current clock frequency. When an internal oscillator is used, this value is forced to 16 MHz.

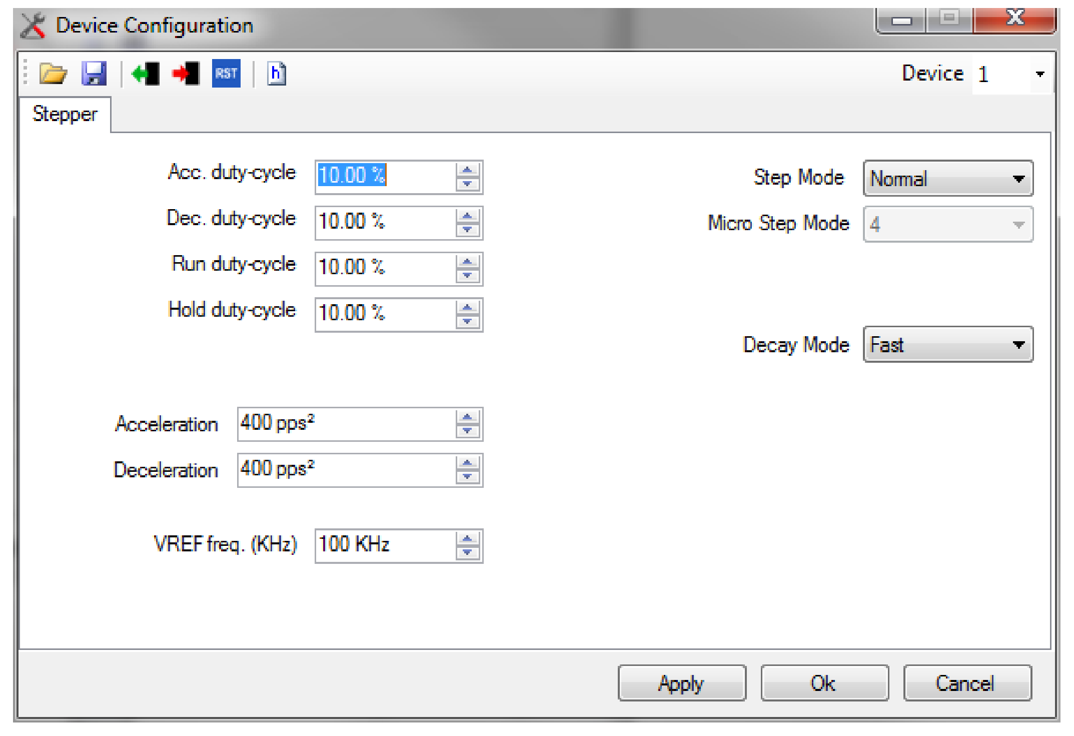
OSCOUT clock frequency: select the frequency that will be supplied by the *OSCOUT* pin (available with the internal oscillator only).

In case of the L6480, L6482 and powerSTEP01 devices the **External clock watchdog** checkbox is available: when it is checked the device is protected from the failures of the external clock source.

3.5.8 Stepper tab (L6208 only)

This tab lists all the parameters related to the L6208 device. Values in numeric boxes can be changed writing the new value or using up/down arrows.

Figure 36. Stepper tab (L6208 only)



Device 1

Acc. duty-cycle 10.00 %

Dec. duty-cycle 10.00 %

Run duty-cycle 10.00 %

Hold duty-cycle 10.00 %

Step Mode Normal

Micro Step Mode 4

Decay Mode Fast

Acceleration 400 pps²

Deceleration 400 pps²

VREF freq. (KHz) 100 KHz

- **Duty-cycle**

The **Acc. duty-cycle**, **Dec. duty-cycle**, **Run duty-cycle** and **Hold duty-cycle** boxes show *DCVAL_X* parameters values expressed in the percentage format. Allowed formats are decimal (e.g. '0.25'), percentage (e.g. '25 %') and hexadecimal (e.g. '0x40').

- **Acceleration and deceleration**

This tab collects all the device parameters that are related to speed profile boundaries.

The values can be changed writing the new value in a common format (pps^2) or in the hexadecimal format using the '0x' prefix. Clicking on up and down buttons the parameter value is increased or decreased of one.

If a value greater than the maximum is written, the change is ignored.

- **Step Mode**

The **Step Mode** allows the user to select between Normal, Half Step and Microstepping.

The **Micro Step Mode** selects the microstepping mode number of microsteps per period/4 and it is available only in the microstepping mode.

- **Decay Mode**

The **Decay Mode** allows to select a Fast or Slow decay mode.

- **Vref freq**

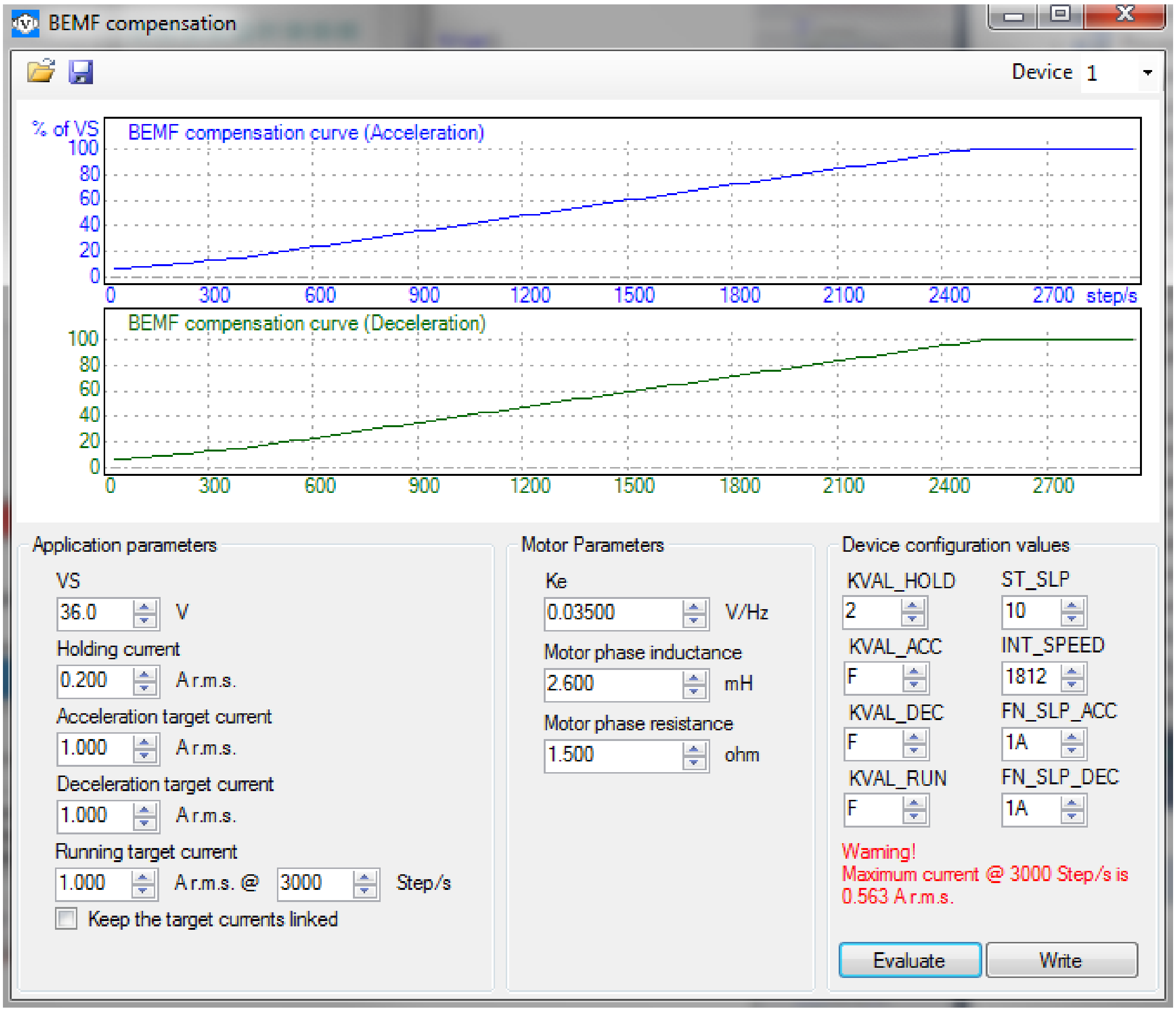
The device *VREF PWM* frequency can be tuned by using the **VREF freq. (KHz)** box.

3.6 BEMF compensation tool

This tool helps the user to set the BEMF compensation parameters according to your application settings. It can be opened from the application main form clicking on the button or selecting **BEMF compensation** in the **Tools** menu.

If more than one device is connected to the communication board (daisy chain configuration), the active device can be selected through the drop list on the top right corner of the toolbar.

Figure 37. BEMF compensation panel



Application parameters

VS
36.0 V

Holding current
0.200 A r.m.s.

Acceleration target current
1.000 A r.m.s.

Deceleration target current
1.000 A r.m.s.

Running target current
1.000 A r.m.s. @ 3000 Step/s

□ Keep the target currents linked

Motor Parameters

Ke
0.03500 V/Hz

Motor phase inductance
2.600 mH

Motor phase resistance
1.500 ohm

Device configuration values

KVAL_HOLD 2	ST_SLP 10
KVAL_ACC F	INT_SPEED 1812
KVAL_DEC F	FN_SLP_ACC 1A
KVAL_RUN F	FN_SLP_DEC 1A

Warning!
Maximum current @ 3000 Step/s is 0.563 A r.m.s.

3.6.1 Setting application and motor parameters

The BEMF compensation algorithm is based on the application requirements, as the supply voltage, target current and motor characteristics.

In the **Application parameters** section the motor supply voltage and load currents need to be set:

- **VS** is the supply voltage.
- **Holding current** is the target r.m.s. current when the motor is stopped.
- **Acceleration target current** is the target r.m.s. current when the motor is accelerating.
- **Deceleration target current** is the target r.m.s. current when motor is decelerating.
- **Running target current** is the target r.m.s. current when the motor is running at constant speed.

By checking the **Keep target currents linked** checkbox, acceleration, deceleration and running target currents are forced to the same value (Running target current).

In the **Motor parameters** section the motor characteristics need to be set:

- **Ke** is the electric constant of the motor. It can be measured through an oscilloscope.
- **Motor phase inductance** is the inductance of the motor phases.
- **Motor phase resistance** is the resistance of the motor phases.

3.6.2 Compensation values evaluation

When the application parameters and motor characteristics have been inserted, the device register values can be calculated clicking on the **Evaluate** button.

Resulting configuration settings are showed on the respective numeric box in the hexadecimal format, implemented compensation curves are shown on the graph on the top of the panel.

If the compensation algorithm fails to evaluate parameters, a warning message is displayed.

This error could be due to wrong application or motor data: for example you cannot obtain a holding current of 1.5 A with a supply voltage of 10 V and phase resistance of 10 Ω (maximum current value is 1 A).

Clicking on the **Write** button write the resulting compensation parameters into the device registers.

Warning: The resulting compensation parameters COULD BE NOT the optimal ones. Better performances can be obtained with fine tuning of the parameters.

3.6.3 Saving and loading application and motor parameters

Clicking on the 💾 button will save the current application and motor parameters. A file selection dialog will be opened to choose if creating a new BEMF setup file (*.dsb) or overwriting an existing one.

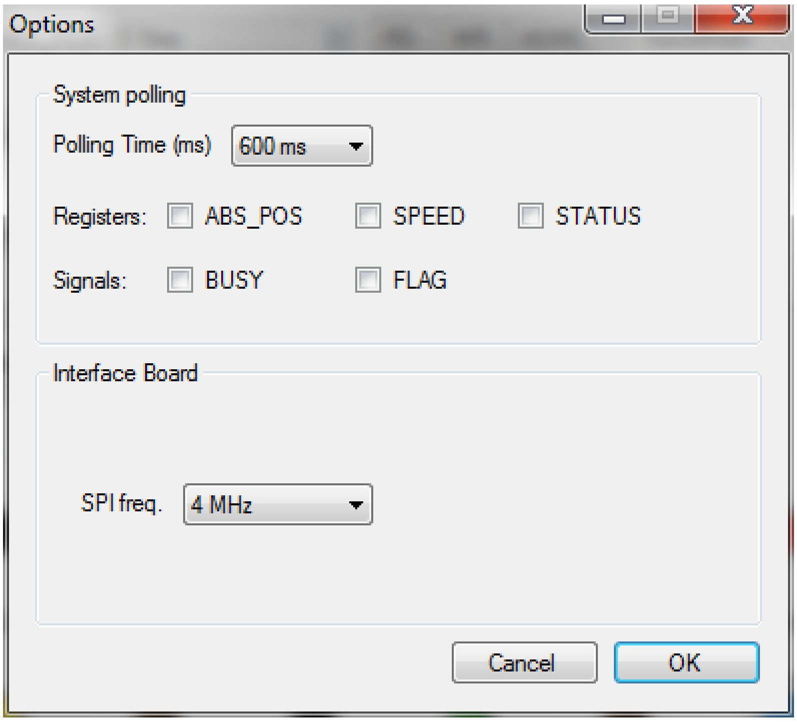
The BEMF setup files can be loaded by clicking on the 📂 button. Choose the target file in the file selection dialog.

When the selected file is loaded, the new BEMF compensation parameters are immediately evaluated (but NOT written into the device).

3.7 Options panel

The **Options** panel allows setting some application options. It can be opened from the main form selecting **Options** on the **Tools** menu.

Figure 38. Options panel (L647x, L648x, and powerSTEP01)



System polling

Polling Time (ms): 600 ms

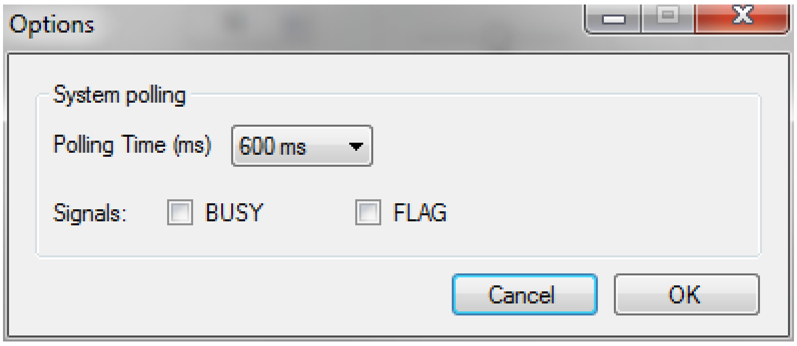
Registers: □ ABS_POS □ SPEED □ STATUS

Signals: □ BUSY □ FLAG

Interface Board

SPI freq. 4 MHz

Figure 39. Options (L6206 and L6208)



System polling

Polling Time (ms): 600 ms

Signals: □ BUSY □ FLAG

- **System polling**

The application can be configured to cyclically refresh the value of the *ABS_POS, SPEED* and *STATUS* registers. You can choose the refresh time in the **Polling Time** list.

The registers refresh can be enabled and disabled through relative checkboxes.

You can also enable the *BUSY/FLAG* lines automatic refresh. In this case, the line values are shown in the main form status bar.

- **Interface board**

Here the SPI speed, **SPI freq**. list, can be selected.

3.8 Scripts editor

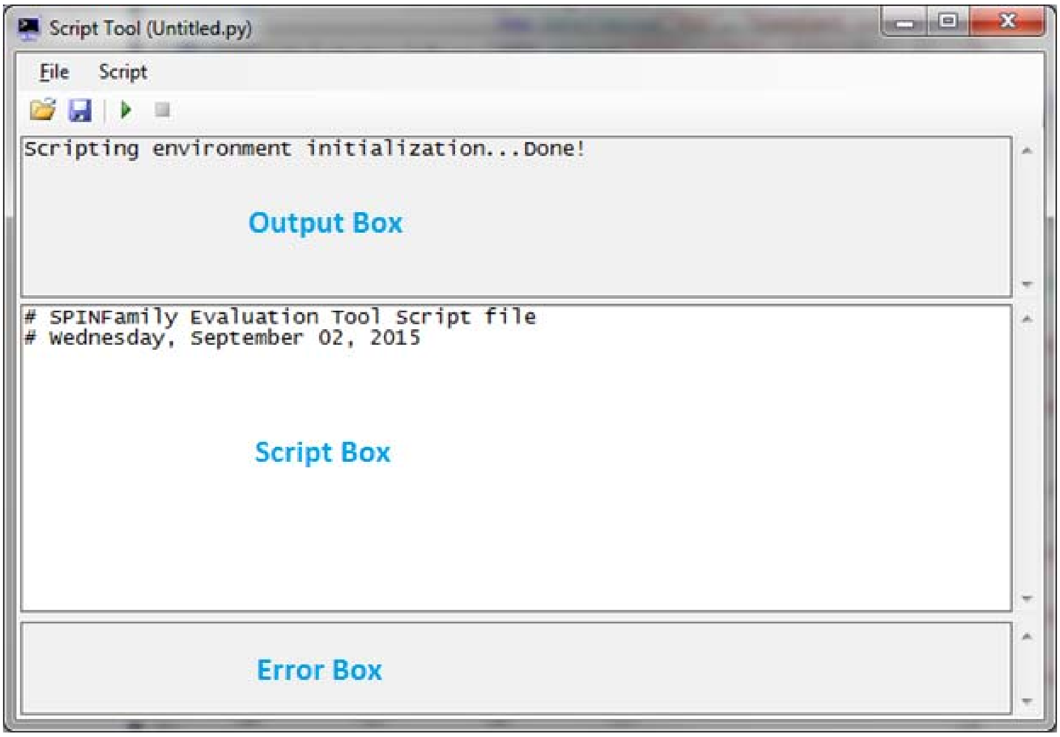
The script language is a tailor made Python extension for the SPINFamily device. The scripting environment is based on IronPython, an open-source implementation of the Python programming language on .NET.

3.8.1 Script Tool

The scripting environment is divided into 3 text boxes (see *Figure 40*):

- Script box: script code
- Output box: displays the script messages output (print function and the execution log when verbose is set to true)
- Error box: displays the error messages.

Figure 40. Script form



```
Scripting environment initialization...Done!
```

```
# SPINFamily Evaluation Tool Script file
# Wednesday, September 02, 2015
```

The menu bar allows managing script files (**File**), start and stop scripting (**Script**).

The toolbar provides shortcuts to the most common commands as **Open** (📂), **Save** (💾), **Run script** (▶) and **Stop script** (■).

3.8.2 Writing a script

The SPINFamily evaluation software scripting environment is based on Python. If you do not know this program language, please spend some time to read the brief introduction in *Appendix D: Python introduction on page 79*.

More detailed information can be found on the **Python documentation** website.

If you already know the Python language, you can find below a detailed description of the extensions implemented in order to easily use the device features.

When you start the tool, a new scripting file named "*Untitled.py*" is generated. You can create a new scripting file anytime through the **New** command in the **File** menu.

You can load an existing script code (*.py files) clicking on the 📂 button or selecting **Open** in the **File** menu. The script area shows the code, here you can edit it.

Save the changes by clicking on the 💾 button or selecting **Save** in the **File** menu. If you want to save the script as a different file keeping the original one unchanged you can do it selecting **SaveAs** in the **File** menu. If you are working on a new scripting file, **Save** and **SaveAs** act in the same way.

You can restore the current script file to its last saved version using the **Reload** command in the **File** menu.

Standard copy, cut, paste and undo commands can be used writing the script.

3.8.3 Execute the script

When the script code is ready, it can be executed selecting the **Run script** in the **Script** menu or by clicking on the ▶ button. When the script is started, the script box is disabled and the ▶ button is checked.

During the script execution output messages are displayed on the output box. At the end of the script execution the result message is shown in the error box, the script box is enabled and the ▶ button is unchecked.

The script execution can be stopped by selecting the **Stop script** in the **Script** menu or by clicking on the ■ button. Stopping the code execution a HardHiZ command will be sent to all devices connected to the chain.

Warning: **At the end of script execution all the variables are kept in the memory. If needed, variable initialization is mandatory at each script execution!**

3.8.4 Python extensions

The integrated scripting environment extends the embedded Python features adding some useful functions. Extended commands help the user to obtain complex behavior with low effort.

Following the list of the extension commands made available to the user. In all the command examples, it is assumed that the verbose script execution mode is enabled.

Verbose

Sets or clears the verbose the script execution mode. If no argument is passed, the actual state is returned.

When the verbose mode is enabled, the description of each executed command is shown on the output text box.

By default this mode is disabled.

Warning: At the end of the script execution the verbose mode status is NOT restored to the default.

- **syntax**

Verbose (*mode*)

- **parameters**

mode - True = enable verbose mode, False = disable verbose mode (boolean)

- **return**

True if the verbose mode is enabled, False otherwise

- **example**

```
Verbose(True)
print Verbose()
Verbose(False)
Output:
Verbose mode set True
True

Verbose mode set False
```

Delay

Waits for the specified delay time expressed in milliseconds.

- **syntax**

Delay (*msDelay*)

- **parameters**

msDelay - delay time expressed in milliseconds (integer)

- **return**

Nothing

- **example**

```
Delay(30)
Output:
30 ms delay
```

GetNdevices

Returns the number of devices connected to the communication board.

- **syntax**

GetNdevices()

- **parameters**

Nothing

- **return**

The number of devices

- **example**

```
print "The chain is composed by %i devices" % GetNdevices()
Output:
The chain is composed by 3 devices
```

AutoWaitBUSY

Sets or clears the automatic BUSY waiting mode. If no argument is passed, the actual state is returned.

When this mode is enabled, the execution of any motion command waits for the release of the *BUSY* line (WaitBUSY function with default timeout value).

By default this mode is disabled.

Warning: **When the automatic BUSY waiting is enabled the BUSY/SYNC output of every device in the chain should be configured in the BUSY mode.**

At the end of the script execution the automatic BUSY waiting mode status is NOT restored to the default.

- **syntax**

AutoWaitBUSY (*mode*)

- **parameters**

mode - True = enable verbose mode, False = disable verbose mode (boolean)

- **return**

True if the automatic BUSY waiting mode is enabled, False otherwise

- **example**

```
AutoWaitBUSY(True)
print AutoWaitBUSY()
AutoWaitBUSY(False)
 Output:
BUSY synchronization set True
True
BUSY synchronization set False
```

WaitBUSY

Waits for the BUSY line of the chain is released. If the BUSY line is forced low for more than *sTimeout* seconds, the function generates a timeout error stopping the script execution.

If no argument is passed, the timeout value is set to 60 seconds.

- **syntax**

WaitBUSY (*sTimeout*)

- **parameters**

sTimeout - Timeout value expressed in seconds (integer)

- **return**

Nothing

- **example**

```
Move(0, "fw", 1000)
WaitBUSY()
Move(0, "fw", 1000)
WaitBUSY(1)
Output:
Move - device: 0 dir: fw n.step: 0X0003E8
Timeout timer started
Waiting for BUSY line (Timeout @ 60 seconds)...
Timeout timer stopped
... BUSY line released
Move - device: 0 dir: fw n.step: 0X0003E8
Timeout timer started
Waiting for BUSY line (Timeout @ 1 seconds)...
... WaitBUSY timeout

Execution error: WaitBUSY timeout
```

ConfirmPopup

Displays a message box to get the user to confirm a question.

- **syntax**

ConfirmPopup(*message, caption*)

- **parameters**

message - confirm question text (string)

caption - title of the message box (string)

- **return**

True if “Yes” button has been clicked, False otherwise

- **example**

```
print ConfirmPopup("Question?","Ask user")
Output:
True
```

StopMessage

Exits the script and shows the specified message in the script exit messages text box.

- **syntax**

StopMessage(*message*)

- **parameters**

message - exit message (string)

- **return**

Nothing

- **example**

```
if (ConfirmPopup("Stop the script execution?", "Quit")):
    StopMessage("Execution stopped by user")
Script exit message output:
Execution error: Execution stopped by user
```

LoadConfiguration

Loads the specified configuration file.

- **syntax**

LoadConfiguration(*device, path*)

- **parameters**

device - connected device ID (integer value)

path - file path (related to current active folder)

- return

Nothing

- **example**

```
SaveConfiguration(0, ".\\configuration\\cfg0.dsc")
LoadConfiguration(0, """.\configuration\cfg0.dsc""")
Output:
Saving configuration from .\configuration\cfg0.dsc file
Loading configuration from .\configuration\cfg0.dsc file
```

SaveConfiguration

Saves configuration into the specified file.

- **syntax**

SaveConfiguration(*device, path*)

- **parameters**

device - connected device ID (integer value)

path - file path (related to current active folder)

- return

Nothing

- example

```
SaveConfiguration(0, ".\\configuration\\cfg0.dsc")
LoadConfiguration(0, """.\configuration\cfg0.dsc""")
Output:
Saving configuration from .\configuration\cfg0.dsc file
Loading configuration from .\configuration\cfg0.dsc file
```

3.8.5 Device commands

The following functions can be used to send commands to the STSPIN family devices.
A detailed description of each command is available on the device datasheets.

In all command examples, it is assumed that the verbose script execution mode is enabled.

Table 16 shows the supported commands per device.

Table 16. Device commands summary

Command	L6470	L6472	L6474	L6480	L6482	PowerSTEP01	L6206	L6208
NOP	X	X	X	X	X	X	X	X
SetParam	X	X	X	X	X	X	X	X
GetParam	X	X	X	X	X	X	X	X
Run	X	X	X	X	X	X	X	X
StepClock	X	X	X	X	X	X		
Move	X	X	X	X	X	X	X	X
GoTo	X	X	X	X	X	X		
GoTo_DIR	X	X	X	X	X	X		
GoUntil	X	X	X	X	X	X		
ReleaseSW	X	X	X	X	X	X	X	X
GoHome	X	X	X	X	X	X		
ResetPos	X	X	X	X	X	X		
ResetDevice	X	X	X	X	X	X		
SoftStop	X	X	X	X	X	X	X	X
HardStop	X	X	X	X	X	X	X	X
SoftHiZ	X	X	X	X	X	X	X	X
HardHiZ	X	X	X	X	X	X	X	X
GetStatus	X	X	X	X	X	X	X	X
BUSY	X	X	X	X	X	X	X	X
FLAG	X	X	X	X	X	X	X	X
RESET	X	X	X	X	X	X	X	X
Enable	X	X	X	X	X	X	X	X
Disable	X	X	X	X	X	X	X	X

NOP

Do nothing command.

- **syntax**

NOP(*device*)

- **Parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

- **return**

nothing

- **example**

```
NOP(0)
```

Output:

```
NOP - device: 0
```

SetParam

This command sets the *param* register of the selected device at the specified value. The *param* value can be both the mnemonic name of the register in the string format or the register address in the integer format.

Some registers can be written when the motor is stopped or in the high impedance state only or cannot be written at all (read-only registers).

Please, refer to the related paragraph in the device datasheet for a detailed description of the registers.

- **syntax**

SetParam(*device, param, value*)

- **parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

param - address of device register (integer value) or mnemonic name (string value)

value - register value (integer value)

- **return**

nothing

- example

```
SetParam(0,0x0D,1000)
SetParam(1,"ACC",0x1A)
Output:
SetParam - device: 0 param: 13 set value: 0X0003E8
SetParam - device: 0 param: ACC set value: 0X00001A
```

GetParam

This command returns the current value of the *param* register of the device. The *param* value can be both the mnemonic name of the register in the string format or the register address in the integer format.

When the verbose script execution mode is enabled, the register value is also printed in the string version (formatted value).

- **syntax**

GetParam(*device, param*)

- **parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

param - address of device register (integer value) or mnemonic name (string value)

- **return**

Register value

- **example**

```
value = GetParam(0,0x15)
print "Full Step Speed = %d" % value
print GetParam(0,"MARK")
Output:
GetParam - device: 0 param: 21
Returned value: 596.093 Step/s
Full Step Speed = 39
GetParam - device: 0 param: MARK
Returned value: 1 Step
1
```

Run

This *device* makes the motor reaching the target *speed* in the direction *dir*.

The *dir* value can be provided in the formats shown in *Table 17*:

Table 17. 'dir' value formats

Parameter format	Forward direction	Backward direction
string	FW, fw, Forward, forward, CW, cw, Clockwise, clockwise	BW, bw, Backward, backward, ccw, CCW, Counterclockwise, counterclockwise
integer	> 0	<= 0
boolean	True	False

The *speed* value can be provided as integer or formatted string. When it is passed in the string format, it should be a decimal number optionally followed by the “S/s” or “Step/s” suffix.

- **syntax**

Run(device, dir, speed)

- **parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

dir - target direction (boolean, integer or formatted string)

speed - target speed (integer value or formatted string)

- **return**

nothing

- **example**

```
Run(0,True,10000)
Run(1,"bw","100.5 S/s")
Output:
Run - device: 0 dir: True speed: 10000
Run - device: 1 dir: bw speed: 100.5 S/s
```

StepClock

This command switches the *device* in the step clock mode and imposes the direction *dir*.

The *dir* value can be provided in the formats shown in *Table 17*.

- **syntax**

StepClock (*device, dir*)

- **parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

dir - target direction (boolean, integer or formatted string)

- **return**

nothing

- **example**

```
StepClock(0,False)
Output:
StepClock - device: 0 dir: False
```

Move

This command makes the *device* moving the motor of *nstep* (micro)steps in the direction *dir*.

The *dir* value can be provided in the formats shown in *Table 17*.

- **syntax**

Move(*device, dir, nstep*)

- **parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

dir - target direction (boolean, integer or formatted string)

nstep - number of steps (integer value)

- **return**

Nothing

- **example**

```
Move(0,1,0x12345)
Output:
Move - device: 0 dir: 1 n.step: 0X012345
```

GoTo

This command makes the *device* reaching the *pos* absolute position through the shortest path.

- **syntax**

Move(*device, pos*)

- **parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

pos - target absolute position (integer value)

- **return**

Nothing

- **example**

```
GoTo(0,98765)
Output:
GoTo - device: 0 abs.pos.: 0X0181CD
```

GoTo_DIR

The GoTo_DIR command makes the *device* reaching the *pos* absolute position imposing the direction *dir*.

The *dir* value can be provided in the formats shown in *Table 17*.

- **syntax**

GoTo_DIR(*device, dir, pos*)

- **parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

dir - target direction (boolean, integer or formatted string)

pos - target absolute position (integer value)

- **return**

Nothing

- **example**

```
GoTo_DIR(0,"ccw",123456)
Output:
GoTo_DIR - device: dir: ccw target pos.: 0X01E240
```

GoUntil

The *GoUntil* command makes the *device* reaching the target *speed* imposing a direction *dir.* When the SW pin input is forced low (switch turn-on event) the ABS_POS register is reset to zero (home position) or its current value is copied into the MARK register according to *act* value; then the device performs a SoftStop command. If the SW_MODE bit of the CONFIG register is low, the HardStop command is performed instead.

The *dir* value can be provided in the formats shown in *Table 17*.

The *act* value can be provided in the formats shown in *Table 18*.

Table 18. 'act' value formats

Parameter format	Sets ABS_POS to HOME	Stores ABS_POS in MARK
string	home, Home, HOME	mark, Mark, MARK
integer	> 0	<= 0
boolean	True	False

The *speed* value can be provided as an integer or formatted string. When it is passed in the string format, it should be a decimal number optionally followed by the “S/s” or “Step/s” suffix.

- **syntax**

GoUntil(*device, act, dir, speed*)

- **parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

act - action (boolean, integer or formatted string)

dir - target direction (boolean, integer or formatted string)

speed - target speed (integer value)

- **return**

Nothing

- **example**

```
GoUntil(0,"home","bw",128)
GoUntil(0,"MARK","FW",2000)
Output:
GoUntil - device: 0 act: home dir: bw spd: 128
GoUntil - device: 0 act: MARK dir: FW spd: 2000
```

ReleaseSW

The ReleaseSW command makes the *device* moving at minimum speed imposing a direction *dir*. When the SW pin input is forced high externally or by an internal pull-up resistor (switch turn-off event), the ABS_POS register is reset to zero (home position) or its current value is copied into the MARK register according to *act* value; then the *device* performs a HardStop command.

The *dir* value can be provided in the formats shown in *Table 17*.

The *act* value can be provided in the formats shown in *Table 18*.

If the minimum speed value is lesser than 5 step/s or low speed optimization is enabled, the motion is performed at 5 step/s.

- **syntax**

ReleaseSW(*device, act, dir*)

- **parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

act - action (boolean, integer or formatted string)

dir - target direction (boolean, integer or formatted string)

- **return**

Nothing

- **example**

```
ReleaseSW(0,True,False)
ReleaseSW(0,0,1)
```

Output:

```
ReleaseSW - device: 0 act: True dir: False
ReleaseSW - device: 0 act: 0 dir: 1
```

GoHome

The GoHome command makes the *device* reaching the home position (zero value of ABS_POS register) through the shortest path.

- **syntax**

GoHome(*device*)

- **parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

- **return**

Nothing

- **example**

```
GoHome(0)
```

output:

```
GoHome - device: 0
```

ResetPos

The command sets the *device* ABS_POS register to zero (home position).

- **syntax**

ResetPos(*device*)

- **parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

- **return**

Nothing

- **example**

```
ResetPos(0)
Output:
ResetPos - device: 0
```

ResetDevice

The ResetDevice command resets the *device* to the power-up conditions.

Warning: It is not recommended to reset the device when output bridges are not in high the impedance state. Performing HardHiZ or SoftHiZ commands before resetting the device is recommended.

- **Syntax**

ResetDevice(*device*)

- **Parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

- **return**

Nothing

- **example**

```
ResetDevice(1)
Output:
ResetDevice - device: 1
```

SoftStop

The SoftStop command makes the *device* to immediately decelerate to zero speed and stop the motor.

- **syntax**

SoftStop (*device*)

- **parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

- **return**

Nothing

- **example**

```
SoftStop(0)
Output:
SoftStop - device: 0
```

HardStop

The HardStop command makes the *device* to immediately stop the motor with infinite deceleration.

- **syntax**

HardStop (*device*)

- **parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

- **return**

Nothing

- **example**

```
HardStop(0)
Output:
HardStop - device: 0
```

SoftHiZ

The SoftHiz command makes the *device* decelerating to zero speed and then disabling the power bridges.

- **syntax**

SoftHiz (*device*)

- **Parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

- **return**

Nothing

- **Example**

```
SoftHiZ(0)
Output:
SoftHiZ - device: 0
```

HardHiZ

The HardHiz command makes the *device* to immediately disable the power bridges.

- **syntax**

HardHiz (*device*)

- **Parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

- **return**

Nothing

- **Example**

```
HardHiZ(0)
Output:
HardHiZ - device: 0
```

Enable

This command enables the selected power bridge.

- **Syntax**

Enable(*device, target*)

- **Parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

target - bridge index (integer value: 0 the first bridge, 1 the second one if existing)

- **example**

```
Enable(0, 0)
Output:
Enable - device: 0 target : 0
```

Disable

This command disables the selected power bridge.

- **Syntax**

Disable(*device, target*)

- **Parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

target - bridge index (integer value: 0 the first bridge, 1 the second one if existing)

- **example**

```
Disable(0, 0)
Output:
Disable - device: 0 target: 0
```

GetStatus

This command returns the *device* STATUS register value and clears the status bit flags.

When the verbose script execution mode is enabled, the STATUS value is also printed in the string version (formatted value).

- **Syntax**

GetStatus(*device*)

- **Parameters**

device - device index (integer value: 0 the first device, 1 the second one, ...)

- **return**

The STATUS register value

- **example**

```
GetStatus(0)
Output:
GetStatus - device: 0
High Impedance state,
SW status: Open,
Motor status: Constant speed in Clockwise direction
```

BUSY

This command returns the status of the BUSY line.

- **Syntax**

BUSY()

- **Parameters**

None

- **Return**

True if the BUSY line of the chain is low, False otherwise

- **Example**

```
if(BUSY()):
    print "Device is busy!"
Output:
BUSY - True
Device is busy!
```

FLAG

This command returns the status of the FLAG line.

- **Syntax**

FLAG()

- **Parameters**

None

- **Return**

True if the FLAG line of the chain is low, False otherwise

- **Example**

```
FLAG()
SetParam(0, "SPEED", 0) # Try to write a read-only register
if(FLAG()):
    print "Alarm!"
Output:
FLAG - False
SetParam - device: 0 param: SPEED set value: 0X000000
FLAG - True
Alarm!
```

RESET

This command returns the RESET line status when it is called without parameter.

It forces the RESET line at a specified status when it is called with a line status parameter.

Warning: It is not recommended to reset the device when the output bridges are not in the high impedance state. Performing HardHiZ or SoftHiZ commands before resetting the device is recommended.

- **Syntax**

RESET()

RESET(*value*)

- **Parameters**

value - True: force reset (RESET line low), False: release reset (RESET line is high)

- **Return**

True if RESET line of the chain is low, False otherwise

- **Example**

```
RESET(True)
print RESET()
RESET(False)
print RESET()
```

Output:

```
RESET writing @ True
RESET reading
True
RESET writing @ False
RESET reading
False
```

Appendix A Firmware update

A.1 STEVAL-PCC009V2

A.1.1 Automatic update using the IBUUI updater

1. Start the IBUUI updater tool (by default in **Start menu > All programs > STMicroelectronics > SPINFamily Evaluation Tool**)
2. Click on the “Start” button.
3. Follow the instructions provided by the tool.

If the automatic firmware update fails, follow the alternative procedure described in *Section A.1.2*.

A.1.2 Manual update using the ST-LINK/V2 programmer

If the “IBUUI updater” software does not work as expected, the firmware can be updated using the ST-LINK/V2 programmer or an equivalent tool.

1. Install the STM32 ST-LINK utility software from: www.st.com/web/catalog/tools/FM147/SC1887/PF258168#.
2. Run the software.
3. Connect the STEVAL-PCC009V2 to the PC through the USB cable and to the ST-LINK/V2 through the 20 poles connector.
4. Open the firmware file (**fwpsin_pcc009v2_XXXX.hex**). The file is located in the “Firmware” folder in the installation folder of the SPINFamily evaluation tool. By default a shortcut to the folder can be found in **Start menu > All programs > STMicroelectronics > SPINFamily Evaluation Tool**.
5. Click on the “Connect to the target” button.
6. Click on the “Program verify” button.
7. Wait until the end of the process.
8. Close the application and reset the board.

A.2 Discovery boards

A.2.1 Update through DFU

1. Download and install the latest version of the DfuSe software from: www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF257916.
2. Open the BOOT jumper on the board (see the board user manual to locate the jumper position).
3. Connect the board to the PC through the USB cable.
4. Wait for the board is properly identified by the OS as “STM Device in DFU mode”.
5. Run the DfuSe demonstration software (by default **Start menu > All programs > STMicroelectronics > DfuSe**).
6. An “STM Device in DFU Mode” must be present in the list of the “Available DFU Devices”. Else, it means that your board is not correctly configured or not connected to the PC.
7. In the “Upgrade or Verify Action” group, click on the “Choose…” button.
8. Select the **fwpspin_discoverykit_XXXX.dfu file**. The file is located in the “Firmware” folder in the installation folder of the SPINFamily evaluation tool. By default a shortcut to the folder can be found in **Start menu > All programs > STMicroelectronics > SPINFamily Evaluation Tool**.
9. Click on the “Upgrade” button.
10. Wait until the end of the process.
11. Close the BOOT jumper.
12. Close the application and reset the board.

A.2.2 Update through ST-LINK/V2 programmer

If the firmware update through the DFU does not work as expected, the firmware can be updated using the ST-LINK/V2 programmer or an equivalent tool.

1. Install the STM32 ST-LINK utility software from: www.st.com/web/catalog/tools/FM147/SC1887/PF258168#.
2. Run the software.
3. Connect the board to the PC through the USB cable and to the ST-LINK/V2 through the 20 poles connector.
4. Open the firmware file (**fwpsin_discoverykit_XXXX.hex**). The file is located in the “Firmware” folder in the installation folder of the SPINFamily evaluation tool. By default a shortcut to the folder can be found in **Start menu > All programs > STMicroelectronics > SPINFamily Evaluation Tool**.
5. Click on the “Connect to the target” button.
6. Click on the “Program verify” button.
7. Wait until the end of the process.
8. Close the application and reset the board.

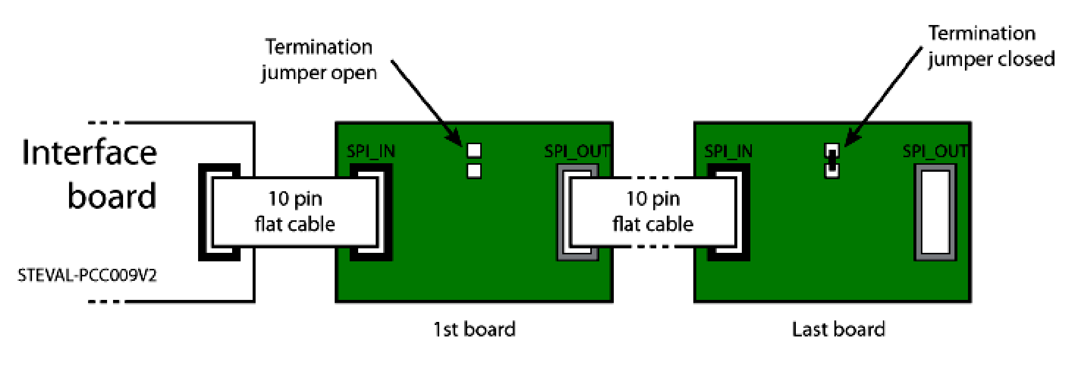
A.3 Nucleo platform

1. Verify that the embedded ST-LINK firmware is updated to the V2-1 revision.
2. Install the STLINK/V2-1 drivers from: www.st.com/web/catalog/tools/FM147/SC1887/PF260217.
3. Connect the Nucleo development board to the PC through the USB cable.
4. The board is detected as a mass storage device named “NUCLEO”.
5. Look for the firmware file (**guifw-STM32XXXX-nucleo.bin**) fitting with the Nucleo development board. The file is located in the “Firmware” folder in the installation folder of the SPINFamily evaluation tool. By default a shortcut to the folder can be found in **Start menu > All programs > STMicroelectronics > SPINFamily Evaluation Tool**.
6. Drag-and-drop the file into the “NUCLEO” device.
7. Reset the board.

Appendix B Daisy chain configuration (STEVAL-PCC009V2)

More demonstration boards can be connected in the daisy chain mode. This way you can control up to eight motors using a single communication board (STEVAL-PCC009V2 only).

Figure 41. Daisy chain example



To drive two or more boards in daisy chain configuration:

1. Plug the interface board to the PC through the USB cable
2. If requested, install interface board drivers
3. Connect the interface board 10-pin connector to the SPI_IN connector of the first demonstration board
4. Open the termination jumper of the demonstration board
5. Connect the SPI_OUT connector of the previous demonstration board to the SPI_IN connector of the next one
6. Repeat point *4.* and *5.* for all the others boards of the chain except for the last one.
7. Check the termination jumpers of all the demonstration boards: all the jumpers except the last one should be opened.

When the chain configuration is set, you can connect the interface board to the PC as usual.

Information about the termination jumper and the SPI connectors can be found in the application note of the specific demonstration board.

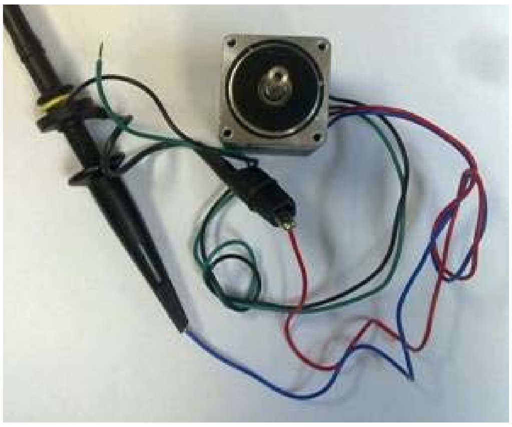
Warning: Increasing the number of the devices connected in the chain could degrade SPI communication performances. If communication issues are found, try to reduce SPI clock speed.

Appendix C Motor electric constant (Ke) measurement

A motor electric constant is the coefficient that relates the motor speed to the BEMF amplitude. This value is usually not present on stepper motor datasheets, but it can be easily measured by means of an oscilloscope.

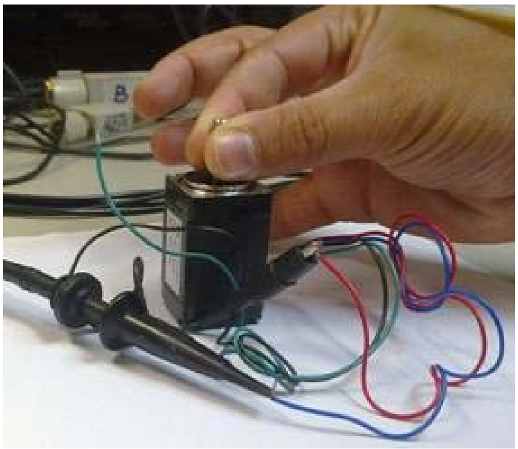
1. First of all connect one of the motor phases to an oscilloscope channel.

Figure 42. Stepper motor



2. Set the oscilloscope trigger value to the rising or falling edge of the channel and set the threshold value close to zero (few mV above or below zero).
3. Quickly turn the motor shaft (you can also do it by hand).

Figure 43. Ke measurement



4. Set oscilloscope time and voltage scales in order to display a sinewave during the motor rotation.
5. Turn the motor until a “good” sinewave is obtained: a good sinewave keeps its amplitude constant for at least 2 or 3 cycles.

This operation might require some attempts.

Figure 44. Ke measurement - bad waveform

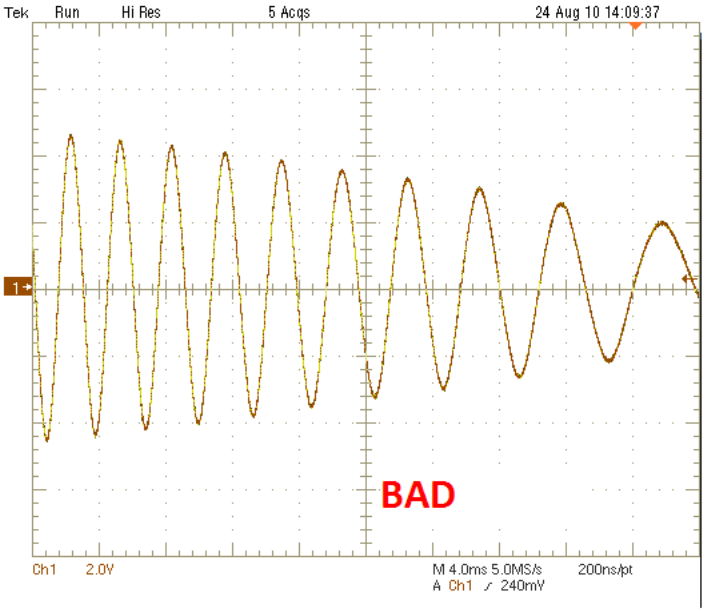
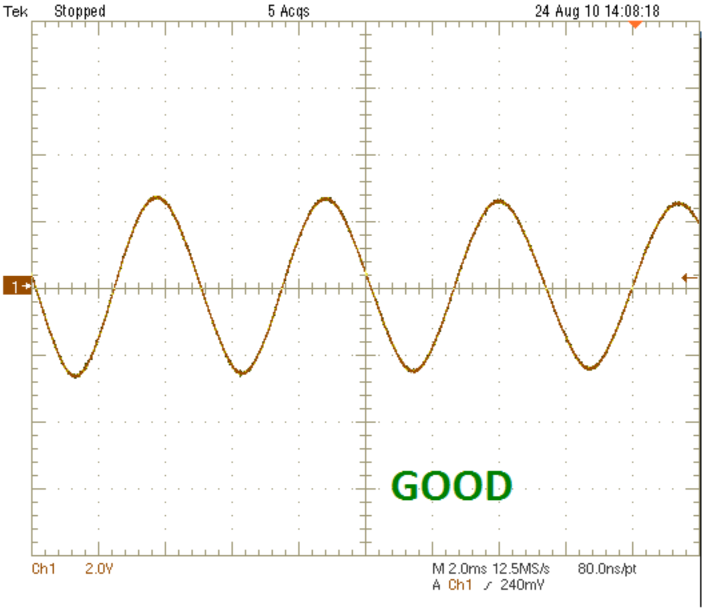
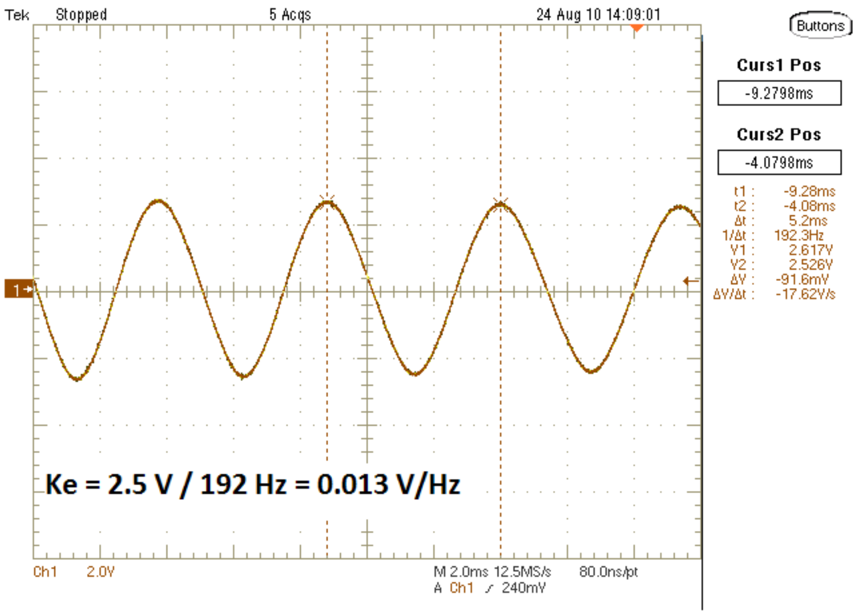


Figure 45. Ke measurement - good waveform



Measure the peak to the frequency ratio of the “good” sinewave. The resulting value is the motor electric constant expressed in V/Hz.

Figure 46. Ke measurement - result



Ke = 2.5 V / 192 Hz = 0.013 V/Hz

Appendix D Python introduction

D.1 Python basics

Python is a complete programming language that allows developing full working applications. The STSPIN family evaluation tool scripting feature makes available the power of this language allowing users to write complex scripts with low effort.

Following a brief description of Python language basics, you can find more information at: www.python.org.

D.2 Program syntax

Python syntax is based on following simple rules:

- Empty lines and lines filled with spaces are ignored
- All identifiers must start with a letter (**A to Z** or **a to z**) or an underscore (_)
- No punctuation characters (e.g. **@, $,** and **%**) should be used within identifiers (underscore excluded)
- Identifiers are case sensitive (**Dog** is different from **dog**)
- All characters following the '#' symbol are considered as comments and then ignored by the code interpreter
- Statements are grouped by an indentation (tab or space indention are both accepted, but spaces are preferred) (see *Section D.2.1*)
- Single statements end with a carriage return
- A variable value is assigned by the equal sign '='
- Variables are defined through value assignments
- Strings can be enclosed in single quotes (') or double quotes (“) (see *Section D.3.2*)
- An escape character in string definitions is '\' (see *Section D.3.2*)
- Lists are defined enclosing indices values between square brackets ([…]) (see *Section D.3.3*)
- List indices are enclosed between square brackets ([…]) (see *Section D.3.3*)
- Indices support slice notation through the colon ':'
- Numbers can be written in the hexadecimal form using the '0x' prefix (see *Section D.3.1*)

Following an example of syntax of a Python program:

```
#I am a comment
A = 1     #variable A is defined and assigned
B = 0x64  # this is 100 in hex format
# 3D = 122    Not a valid identifier! It starts with a number!
# _r4? = 34   Not a valid identifier! You cannot use punctuation chars (?)!

s = "#I'm not a comment! I'm a string!" # String in double quotes
s2 = 'I am a string too!'               # String in single quotes

while B > A :
    # 'while' loop statements are grouped by indentation...
    A = A + 1 #New values assigned to A and B
    B = B - A
    if A == 3 : print "You can put inline single statement groups as this 
one."
    if A == 5 :
    # end of while loop

seq = [2, 4, 6, 8, 10, 12] # This is a List
upseq = seq[3:]            # This is equal to [8, 10, 12]
downseq = seq[:3]          # This is equal to [2, 4, 6]
midseq = seq[2:4]          # This is equal to [6, 8]


     # Empty lines are ignored

print "end"
#This is the end of the program
```

D.2.1 Lines and indentation

The Python language indicates blocks of the code by the line indentation, which is rigidly enforced.

The programmer can specify a variable number of spaces in the indentation, but all statements within the block must be indented by the same number of spaces.

Example

```
# Wrong code (indentation error)
if (x > 0):
    # Block 1
    print "True result"
  a = 1 # wrong indentation: it should be aligned to previous command
else:
  # Block 2
  print "False result"
  a = 2

# Correct code
if (x > 0):
    # Block 1
    print "True result"
    a = 1
else:
  # Block 2
  print "False result"
  a = 2
```

Statements in the Python end with a new line. You can use the line continuation character (\) to specify a multiline command.

Example

```
sum = value_a + \
      value_b + \
      value_c
```

You can use the semicolon (;) to insert multiple statements on the single line.

Example

```
x = 1; y = 2; print “Answer:”, x + y #output--> Answer: 3
```

D.2.2 Comments

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the physical line end are part of the comment, and the Python interpreter ignores them.

Example

```
#I'm a comment
str = "#I'm not a comment! I'm a string!" # And I'm a comment too
```

D.2.3 Variables

Variables are nothing but reserved memory locations to store values. This means that when a variable is created, some space in the memory is reserved. Based on the data type (see *Section D.3*) of a variable, the interpreter allocates the memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, integers, decimals, or characters in these variables can be stored.

Variable identifiers must start with a letter or an underscore and cannot include punctuation and special characters but underscore. Python is a case-sensitive language thus an **A** variable is different from **a**.

To define a variable a value should be assigned to it through the assignment operator '=' (see *Section D.4.1 on page 85*).

Example

```
counter = 100    # integer assignment
miles = 1000.0   # floating point
name = "Nemo"    # string
```

D.2.4 Reserved words

The following list shows the reserved words in Python and in the xSpin device GUI environment. These reserved words may not be used as constants or variables or any other identifier names.

Reserved words in Python and in the xSpin device GUI environment

_IBUface
and
Assert
break
BUSY
CheckAssembly
Class
ConfirmPopup
continue
Def
Del
Delay
elif
else
Except
Exec
fh
Finally
FLAG
for
From
GetNdevices
GetParam
GetStatus
Global
GoHome
GoTo
GoTo_DIR
GoUntil
HardHiZ
HardStop
if
import
in
is
lambda
LoadConfiguration
Move
NOP
not
or
pass
print
raise
RelaseSW
RESET
ResetDevice
ResetPos
return
Run
SaveConfiguration
SetParam
SoftHiZ
SoftStop
StepClock
SysInit
try
Verbose
while
with
yield

D.3 Python types

D.3.1 Numbers

Python supports different types of numbers:

- int (signed integers)
- long (long integers [can also be represented in octal and hexadecimal])
- float (floating point real values)
- complex (complex numbers)

Here are some examples:

Examples of number types in Python

int	long	float	complex
10	123456789L	0.0	3.14j
100	-0x1ABCDL	15.20	45.j
-786	0133L	-21.9	9.322e-36j
080	0xABCDEFABCDEFABCDEFl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

D.3.2 Strings

Strings are formally groups of characters. A string is declared using the equal sign '=' followed by the string characters enclosed in single quotes or double quotes. Triple quotes (“”” or ''') can be used to span the string across multiple lines.

If a single quote or a double quote character needs to be included (according to the enclosing method) into the string, the escape character **'\'** needs to be added before it.

Example

```
#string declaration
strdq = "I'm a string saying \"Hello!\""
strsq = 'I\'m a string saying "Hello!"'
# The two strings are formally the same: I'm a string saying "Hello!"

#multiline string
paragraph = """This is a
multiline string."""
```

Strings can be concatenated using the + operator and repeated using the * operator. Two literal (not variable) strings can be concatenated omitting the + operator.

Example

```
str1 = "Hello"
str2 = str1 + "World" # str2 = "HelloWorld"
str3 = 3*str1         # str3 = "HelloHelloHello"
str4 = "See" 'You'    # str4 = "SeeYou"
```

Characters can be extracted from a string using indexing. String characters are enumerated left to right starting from 0. Negative indexes are also allowed to start counting from the right (-1 is the last character, -2 is the last but one and so on).

String characters cannot be changed.

Example

```
word = "STMicroelectronics"
print word[4]       # print 'c'
print word[:3]      # print 'STM'
print word[7:]      # print 'electronics'
print word[-11:-3] # print 'electron'

# word[2] = 'X' is not allowed!!! String characters cannot be changed this way.
```

D.3.3 Lists

Lists are groups of values identified by an index. A list is declared using the equal sign '=' followed by the list items list separated by commas and enclosed between square brackets (**[…]**). The list items can be of different types. The list declarations do not need to use the line continuation character (see *Section D.2.1 on page 80*).

Example

```
#List declaration
seq = [2, 3, "alpha", 'beta']
```

The list items can be extracted and manipulated using indices. The Index 0 is assigned to the first item of the list, 1 to the second and so on. Negative indexes are also allowed to start counting from the right (-1 is the last item of the list, -2 is the last but one and so on).

Example

```
A = seq[0]   # A = 3
B = seq[2]   # B = "alpha"
C = seq[-1] # C = 'beta'
```

Like strings, lists can be sliced, concatenated and copied (see *Section D.3.2*).

Example

```
A = seq[:2]    # A = [2, 3]
B = seq[2:]    # B = ["alpha", 'beta']
C = seq[1:-1]  # C = [3, "alpha"]
D = seq[1:3]   # D = [3, "alpha"]

E = 3 * D + 2 * A   # E = [3, "alpha", 3, "alpha" , 3, "alpha", 2, 3, 2, 3]

copy = seq[:] # copy is a shallow copy of seq
```

Unlike strings, list items can be individually changed. Slice assignments are also possible.

Example

```
seq[1] = 100 + seq[1]      # seq = [102, 3, ["alpha", 'beta']
seq[1:3] = [103, 104]      # seq = [102, 103, 104, 'beta']
seq[0:2] = []              # seq = [104, 'beta']
seq[1:1] = ["alpha", 105] # seq = [104, "alpha", 105, 'beta']
seq[:0] = [103, "gamma"]  # seq = [103, "gamma", 104, "alpha", 105, 'beta']

seq[:] = []                # seq = [] (empty)
```

Table 19. Summary table

Operator	Description
seq[i] = t	Replace item i of seq list with t
seq[i:j] = st	Replace items from i to j of seq list with st
seq[i:j] = []	Delete items from i to j of seq list
seq[i:i] = [x]	Insert x items before item i of seq list
seq[0:0] = [x]	Insert x items at the beginning of seq list
seq[len(seq): len(seq)] = [x]	Insert x items at the end of seq list

D.4 Python statements, functions and operators

D.4.1 Operators

Table 20 assumes a = 5, b = 10, s1 = ”Hello” and s2 = ”World”:

Table 20. Examples with operators

Operator	Description	Examples
+	Addition and concatenation.	a + b ---> 15; s1 + s2 ---> “HelloWorld”
-	Subtraction.	a - b ---> -5
*	Multiplication and repetition.	a * b ---> 50; s1 * a ---> “HelloHelloHelloHelloHello”
/	Division.	b / a ---> 2
%	Modulus - divides the left hand operand by the right hand operand and returns the remainder.	b % a ---> 0
**	Exponent - performs the exponential (power) calculation on the operators.	b**a ---> 100,000
//	Floor Division - the division of the operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 is equal to 4 and 9.0//2.0 is equal to 4.0
==	Checks if the values of the two operands are equal or not; if yes then the condition becomes True.	(a == b) is not true

Table 20. Examples with operators (continued)

Operator	Description	Examples
!=	Checks if the values of the two operands are equal or not; if the values are not equal then the condition becomes True.	(a != b) is true
<>	Checks if the values of the two operands are equal or not; if the values are not equal then the condition becomes True. It is similar to != operator.	(a <> b) is true
>	Checks if the value of the left operand is greater than the value of the right operand; if yes then the condition becomes True.	(a > b) is not true
<	Checks if the value of the left operand is lower than the value of the right operand; if yes then the condition becomes True.	(a < b) is true
>=	Checks if the value of the left operand is greater than or equal to the value of the right operand; if yes then the condition becomes True.	(a >= b) is not true
<=	Checks if the value of the left operand is lower than or equal to the value of the right operand; if yes then the condition becomes True.	(a <= b) is true
=	Simple assignment operator.	c = a + b will assign the value of a + b to c
+=	Add AND assignment operator.	c += a is equivalent to c = c + a
-=	Subtract AND assignment operator.	c -= a is equivalent to c = c - a
*=	Multiply AND assignment operator.	c *= a is equivalent to c = c * a
/=	Divide AND assignment operator.	c /= a is equivalent to c = c / a
%=	Modulus AND assignment operator.	c %= a is equivalent to c = c % a
**=	Exponent AND assignment operator.	c **= a is equivalent to c = c ** a
//=	Floor Division and assigns a value.	c //= a is equivalent to c = c // a
&	Binary AND Operator.	(0xAC & 0x0F) ---> 0x0C (0000 1100)
|	Binary OR Operator.	(0x30 | 0x0D) ---> 0x3D (0011 1101)
^	Binary XOR Operator.	(0xAA ^ 0xF0) ---> 0x5A (0101 1010)
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits (bitwise not).	(~0x3F) ---> 0xC0 (1100 0000)
<<	Binary Left Shift Operator.	0x13 << 2 ---> 0x4C (0100 1100)
>>	Binary Right Shift Operator.	0x13 >> 2 ---> 0x04 (0000 0100)
and	Logical AND operator.	True and True is true
or	Logical OR Operator.	False or True is true
not	Logical NOT Operator.	not True is false
in	True if it finds a variable in the specified list and false otherwise.	“cat” in [“cat”, “dog”, “cow”] is True
not in	True if it not finds a variable in the specified list and false otherwise.	“cat” not in [“cat”, “dog”, “cow”] is False

D.4.2 The print statement

The print statement prints on the output window the result of the expression evaluation followed by a carriage return. More than one expression result can be sent to the output if a list of expressions, separated by commas, is added after the print keyword. A space is added between each single expression result.

If a comma is written at the end of the statement, no carriage return is added.

Syntax

```
print expression [, expression, ...][,]
```

Example

```
l = "A"
n = 10
print 'Current Letter :', l
print 'Current Number:', n
print 'Letter', l,
print 'and number', n

#output:
#Current Letter : A
#Current Number: 10
#Letter A and number 10
```

D.4.3 The if-elif-else statement

The **if-elif-else** statement is used to conditionally execute the part of the code according to the expression results.

The code after the **if** keyword is executed if the *expression* result is True; the program execution exits the statement when **elif** keyword, else the keyword or the end of statement are reached.

If the *expression* result is False the program execution jumps to the next **elif** keyword and checks the related logic expression; hereafter the code is executed as **elif** is an **if** keyword.

If no **elif** keyword is present the program execution jumps to the **else** optional keyword and executes the following statements; the program exits the statement otherwise.

Syntax

```
if expression:
   statement(s)
[elif] expression2:
   statement(s)
[elif] expression3:
   statement(s)
[else:]
   statement(s)
```

D.4.4 The while statement

The **while** statement is used to repeat the code execution according to an expression result.

The program checks the *expression* result: if it is True the statements following the **while** keyword are executed; the program exits the statement otherwise. When the end of the statement is reached, the program execution returns to the **while** keyword and the *expression* result is checked again.

If an optional **else** keyword is put at the end of the statement, its code is executed before exiting the statement.

Syntax

```
while expression:
   statement(s)
[else:]
   statements(s)
```

Example

```
num = 0
while num < 5:
   num += 1
   print "Iteration", num

#output:
#Iteration 1
#Iteration 2
#Iteration 3
#Iteration 4
#Iteration 5
```

D.4.5 The for statement

The **for** statement repeats the code execution for each element of a list.

At first statement iteration *itering_var* is forced equal to the first element of the sequence and then the statement code is executed. When the end of the statement is reached, the program execution returns to the **for** keyword and the *itering_var* value is set equal to the next *sequence* element one. If the end of the *sequence* is reached, the program exits the statement.

If an optional **else** keyword is put at the end of the statement, its code is executed before exiting the statement.

Syntax

```
for iterating_var in sequence:
   statements(s)
[else:]
   statements(s)
```

Example

```
for animal in ['cat', 'dog', 'cow']:
   print animal

#output:
#cat
#dog
#cow
```

D.4.6 The break statement

The **break** statement forces the program execution to exit the current statement. The **break** statement is commonly used to exit from a loop statement when an external condition occurs. The **break** statement can be used in both **while** (see *Section D.4.4*) and **for** (see *Section D.4.5*) statements. The **break** statement DOES NOT cause the **else** code of the loop execution.

Example

```
for letter in 'STmicrolectronics':
   if letter == 'c': break
   print 'Current Letter :', letter
else
   print "You'll not see this message"
print "See you!"

#output:
#Current Letter : S
#Current Letter : T
#Current Letter : m
#Current Letter : i
#See you!
```

D.4.7 The continue statement

The **continue** statement rejects all the remaining statements in the current iteration and moves the program execution back to the top of the loop. The **continue** statement can be used in both **while** (see *Section D.4.4*) and **for** (see *Section D.4.5*) statements.

Example

```
var = 10
while var > 0:
   var = var -1
   if var > 5: continue
   print 'Current value :', var #This code is skipped until var <= 5
print "All done!"

#output:
#Current value : 5
#Current value : 4
#Current value : 3
#Current value : 2
#Current value : 1
#Current value : 0
#All done!
```

D.4.8 The pass statement:

The **pass** statement has to be used when a statement is required syntactically but no operation has to be executed. The **pass** statement is a *null* operation; nothing happens when it is executed.

Example

```
for letter in 'Python':
   if letter == 'h':
      pass
      print 'This is pass block'
   print 'Current Letter :', letter
```

D.4.9 The len function

The **len** function returns the number of elements in a list or a string. This function, combined with the **range** (see *Section D.4.10*), can be used to iterating over the indices of a list.

Example

```
seq = [0, 1, 2, 3, 4]
print len(seq) # 5 is printed to output

spelling = "STMicroelectronics"
for index in range(len(spelling)):
   print spelling[index], "-",

#output:
#S-T-M-i-c-r-o-e-l-e-c-t-r-o-n-i-c-s-
```

D.4.10 The range function

The **range** function generates a list containing an arithmetic progression. This function, combined with the **len** (see *Section D.4.9*), can be used to iterate over the indices of a list.

Example

```
seq = range(5) # seq = [0, 1, 2, 3, 4]

spelling = "STMicroelectronics"
for index in range(len(spelling)):
   print spelling[index], "-",

#output:
#S-T-M-i-c-r-o-e-l-e-c-t-r-o-n-i-c-s-
```

D.4.11 The Infinite Loops

You must pay attention to not include never ending loops in your script sources. An infinite loop can be interrupted by means of the “**Script>Stop script**” menu item. The command interrupts the script execution and sets all the connected devices in the high impedance state [HardHiZ command (see *Section 3.8.5 on page 60*)].

Revision history

Table 21. Document revision history

Date	Revision	Changes
03-May-2016	1	Initial release.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

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

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

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

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

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

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