

FlashRunner Cube Series

**High-Performance, Standalone In-
System Programmers**

User's Manual

Revision 2.0 — November



Copyright © 2023 SMH Technologies

DC11573

Copyright Information

Copyright © 2023 SMH Technologies Srl.

No part of this manual may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from SMH Technologies.

Disclaimer

The material contained in this document is provided “as is”, and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, SMH Technologies disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. SMH Technologies shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should SMH Technologies and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Whilst every effort has been made to ensure that programming algorithms are correct at the time of their release, it is always possible that programming problems may be encountered, especially when new devices and their associated algorithms are initially released. It is SMH Technologies’s policy to endeavor to rectify any programming issues as quickly as possible after a validated fault report is received.

It is recommended that high-volume users always validate that a sample of a devices has been programmed correctly, before programming a large batch. SMH Technologies can not be held responsible for any third party claims which arise out of the use of this programmer including ‘consequential loss’ and ‘loss of profit’.

SMH Technologies Warranty Information

SMH Technologies warrants that this product will be free from defects in materials and workmanship for a period of one (1) year from the date of shipment. If any such product proves defective during this warranty period, SMH Technologies, at its option, either will repair the defective product without charge for parts and labor, or will provide a replacement in exchange for the defective product. Parts, modules and replacement products used by SMH Technologies for warranty work may be new or reconditioned to like new performance. All replaced parts, modules and products become the property of SMH Technologies. In order to obtain service under this warranty, Customer must notify SMH Technologies of the defect before the expiration of the warranty period and make suitable arrangements for the performance of service. Customer shall be responsible for packaging and shipping the defective product to the service center designated by SMH Technologies, with shipping charges prepaid. SMH Technologies shall pay for the return of the product to Customer if the shipment is to a location within the country in which the SMH Technologies service center is located. Customer shall be responsible for paying all shipping charges, duties, taxes, and any other charges for products returned to any other locations. This warranty shall not apply to any defect, failure or damage caused by improper use or improper or inadequate maintenance and care. SMH Technologies shall not be obligated to furnish service under this warranty a) to repair damage resulting from attempts by personnel other than SMH Technologies representatives to install, repair or service the product; b) to repair damage resulting from improper use or connection to incompatible equipment; c) to repair any damage or malfunction caused by the use of non-SMH Technologies supplies; or d) to service a product that has been modified or integrated with other products when the effect of such modification or integration increases the time or difficulty of servicing the product.

THIS WARRANTY IS GIVEN BY SMH TECHNOLOGIES WITH RESPECT TO THE PRODUCT IN LIEU OF ANY OTHER WARRANTIES, EXPRESS OR IMPLIED. SMH TECHNOLOGIES AND ITS VENDORS DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SMH TECHNOLOGIES’ RESPONSIBILITY TO REPAIR OR REPLACE DEFECTIVE PRODUCTS IS THE SOLE AND EXCLUSIVE REMEDY PROVIDED TO THE CUSTOMER FOR BREACH OF THIS WARRANTY. SMH TECHNOLOGIES AND ITS VENDORS WILL NOT BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IRRESPECTIVE OF WHETHER SMH TECHNOLOGIES OR THE VENDOR HAS ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Technical Support

Please e-mail any technical support questions about this product to: support@smh-tech.com

Contents

1	FR CUBE AT A GLANCE	5
1.1	OVERVIEW	5
1.2	FEATURES	6
1.3	MODEL COMPARISON	7
1.4	PACKAGE CHECKLIST	8
1.5	CONNECTORS OVERVIEW	8
1.6	LEDs	9
1.7	PROGRAMMING DRIVERS AND LICENSES	9
2	GETTING STARTED	10
2.1	GUIDED TUTORIAL	10
2.2	MANUAL PROJECT EDITING	21
2.3	WHERE TO GO FROM HERE	21
3	COMMANDS	22
3.1	OVERVIEW	22
3.2	COMMAND SYNTAX	23
3.3	OK ANSWER	23
3.4	ERR ANSWER	23
3.5	BUSY ANSWER	23
3.6	FR CUBE TERMINAL	24
3.7	COMMAND REFERENCE	25
3.8	DATA IN/OUT COMMANDS	25
3.9	EXECUTION COMMAND	26
3.10	FILE SYSTEM COMMANDS	27
3.11	PROGRAMMING COMMANDS	28
3.12	STATUS COMMANDS	29
3.13	SYSTEM COMMANDS	30
3.14	TIME COMMANDS	31
3.15	VOLATILE MEMORY COMMANDS	32
4	STANDALONE MODE	33
4.1	OVERVIEW	33
4.2	SIGNALS	33
4.3	PROJECT ASSIGNMENT	34
5	FR CUBE API	35
5.1	OVERVIEW	35
5.2	INCLUDING THE API IN YOUR APPLICATION	35
5.3	FUNCTION REFERENCE	36
5.4	FR_CLOSECOMMPORT()	36
5.5	FR_EXECOMMAND()	36
5.6	FR_GETFRAME()	37
5.7	FR_GETLASTERRORMESSAGE()	38
5.8	FR_RECEIVEFILE()	38
5.9	FR_SENDFILE()	39
5.10	ERRORE. IL SEGNALIBRO NON È DEFINITO.
5.11	FR_SENDFRAME()	40
5.12	FR_OPENCOMMPORTA	40
5.13	40



6	FR CUBE FILE SYSTEM	41
6.1	OVERVIEW	41
6.2	FILE SYSTEM STRUCTURE	42
7	VARIABLE DATA PROGRAMMING	43
7.1	OVERVIEW	43
7.2	USAGE	43
8	POWER AND RELAYS OPTIONS	45
8.1	POWER SUPPLY OPTIONS.....	45
8.2	RELAYS.....	45
9	CONNECTORS	46
9.1	ISP CONNECTORS.....	46
9.2	GROUND DOMAINS.....	49
10	SPECIFICATIONS	50
10.1	ELECTRICAL SPECIFICATIONS	50
10.2	ISP CONNECTORS.....	51
10.3	MECHANICAL SPECIFICATIONS	51
11	PROTECTED MODE AND DATA ENCRYPTION	53
11.1	PROTECTED MODE	53
11.2	DATA ENCRYPTION	55
12	DIAGNOSTIC TEST	56
13	FAQ AND USER'S CASES	56
14	CONTACTS	58

1 FR Cube At a glance

1.1 Overview

Congratulations for purchasing a FR CUBE In-System Programmer. The FR CUBE Series of In-System Programmers are a breakthrough in the Programming industry. The programmers support a large number of devices (microcontrollers, memories, CPLDs and other programmable devices) from various manufacturers and have a compact size for easy ATE/ fixture integration. They work in standalone or connected to a host PC (RS232, LAN and USB connections are built-in), and are provided with easy-to-use software utilities



Figure 1: FlashRunner Cube

1.2 Features

- Support of microcontrollers, serial and parallel memories, CPLDs and other programmable devices
- High-speed, parallel programming
- Compact size (fixture friendly)
- Standalone operations or host controlled
- Designed for easy ATE interfacing
- Robust and reliable
- Support of several programming interfaces (JTAG, BDM, SPI, I²C, UART, etc.)
- Large built-in internal memory for projects, images, etc.
- Programmable power supply output (1.5 – 13V)
- Programmable I/O Voltage (1.6 – 5.5V)
- High-speed I/O
- USB, LAN (isolated), RS-232 (isolated) and low-level interface (isolated)
- ISP I/O relay barrier (only available on the single-site model)
- I/O protection
- Wide range power power supply (12 – 24V)

The shortest possible programming times are guaranteed due to a combination of highly optimized programming algorithms, local storage of programming data and high slew rate line driver circuitry.

1.3 Model Comparison

FR CUBE Model Comparison

Feature	FR CUBE GP02	FR CUBE GP04	FR CUBE GP08
General features			
Programming Sites	2	4	8
Power Supply	12 – 24V	12 – 24V	12 – 24V
Device Type Support	Microcontrolle, CPLDs, Serial Memories	Microcontrolle, CPLDs, Serial Memories	Microcontrolle, CPLDs, Serial Memories
Programming Protocols	UART, SPI, JTAG, I ² C, BDM, SWD, etc.	UART, SPI, JTAG, I ² C, BDM, SWD, etc.	UART, SPI, JTAG, I ² C, BDM, SWD, etc.
Relay Barrier	No	No	No
ISP Lines			
Adj. Voltage Range	1.6-5.5V	1.6-5.5V	1.6-5.5V
Adj. Voltage Resolution	100mV	100mV	100mV
Bidirectional Lines	12	24	48
Prog. Clock Out Lines	2	4	8
Progr.ble Power Supply (PPS)			
Range	1.5-15V	1.5-15V	1.5-15V
Resolution	100mV	100mV	100mV
Host Interface			
RS-232 (Isolated)	Yes	Yes	Yes
LAN (Isolated)	Yes, 100Mbit/s	Yes, 100Mbit/s	Yes, 100Mbit/s
USB	Yes, Full Speed	Yes, Full Speed	Yes, Full Speed

All the FlashRunner Cube models offer the Low-Level interface (Isolated) using the START, START_ENA[1...n], PASS/FAULT[1...n], BUSY and PRJ_SEL[0...5] signals.

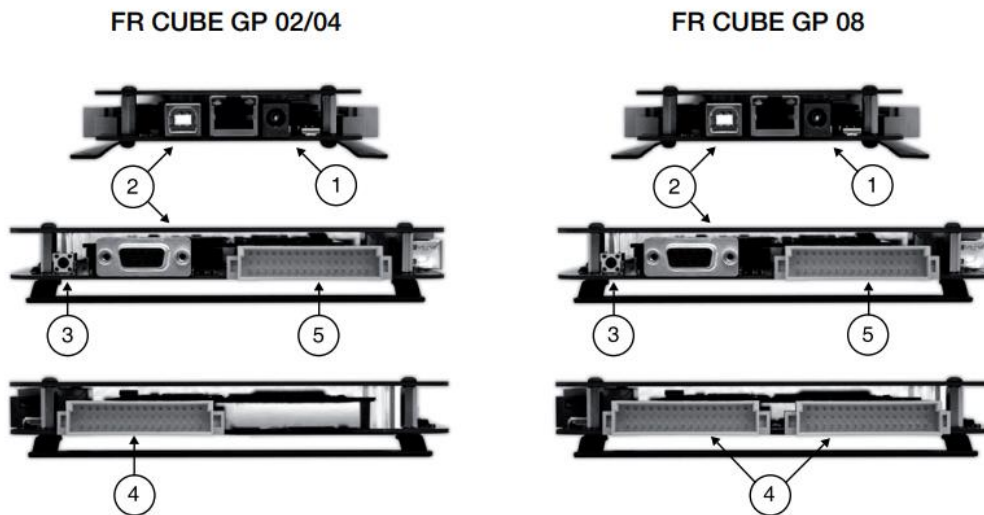
1.4 Package Checklist

The FR CUBE package includes the following items:

1. FR CUBE unit
2. 15V power supply
3. Serial and USB cabled
4. FR CUBE test board
5. 48-way, female wire-wrap DIN41612 connector
6. Software CD

1.5 Connectors overview

FR CUBE has several connectors for interfacing to a host PC, to an Automatic Test Equipment (ATE), and to the target system(s) to be programmed. The following pictures show where, depending on the model, the various connectors are located.



1. The POWER connector accepts a DC voltage between 12V and 24V.
2. The USB connector, LAN, and RS-232 connectors are used to interface the instrument to a PC.
3. The ETH RESET push button is used to reset LAN settings to their factory settings.
4. The ISP connector(s) are used to interface to the target system(s) to be programmed.
5. The LOW-LEVEL INTERFACE connector is used to interface the instrument to an ATE or other systems.

For details and pinout of the various connectors, see the “Connectors” chapter.

1.6 LEDs

The LEDs on the top cover of the instrument, from top to bottom, indicate:

1. **POWER:** the instrument is turned on.
2. **STATUS:** indicates system warnings. Normally off, blinks if the system needs user action (to retrieve detailed error information, see “Status Commands” section)
3. **BUSY:** turns on when programming (when a programming project is being executed).
4. **PASS/FAULT:** result of programming. Each programming site has a PASS/FAULT LED, which turns green if programming on that site has been successful, red otherwise.

FR CUBE GP02 LEDs	FR CUBE GP04 LEDs	FR CUBE GP08 LEDs
PWR ON <input type="checkbox"/>	PWR ON <input type="checkbox"/>	PWR ON <input type="checkbox"/>
STATUS <input type="checkbox"/>	STATUS <input type="checkbox"/>	STATUS <input type="checkbox"/>
BUSY <input type="checkbox"/>	BUSY <input type="checkbox"/>	BUSY <input type="checkbox"/>
S1 PASS <input type="checkbox"/> FAULT	S1 PASS <input type="checkbox"/> FAULT	S1 PASS <input type="checkbox"/> FAULT
S2 PASS <input type="checkbox"/> FAULT	S2 PASS <input type="checkbox"/> FAULT	S2 PASS <input type="checkbox"/> FAULT
	S3 PASS <input type="checkbox"/> FAULT	S3 PASS <input type="checkbox"/> FAULT
	S4 PASS <input type="checkbox"/> FAULT	S4 PASS <input type="checkbox"/> FAULT
		S5 PASS <input type="checkbox"/> FAULT
		S6 PASS <input type="checkbox"/> FAULT
		S7 PASS <input type="checkbox"/> FAULT
		S8 PASS <input type="checkbox"/> FAULT

1.7 Programming Drivers and Licenses

FR CUBE comes with preinstalled programming drivers (algorithms) that support common microcontrollers and memories. When you purchase a new programming driver, you are supplied with a new driver file (.wnd) and an updated license file (.wnl). The license file enables the use of all of your purchased drivers on your specific FR CUBE unit.

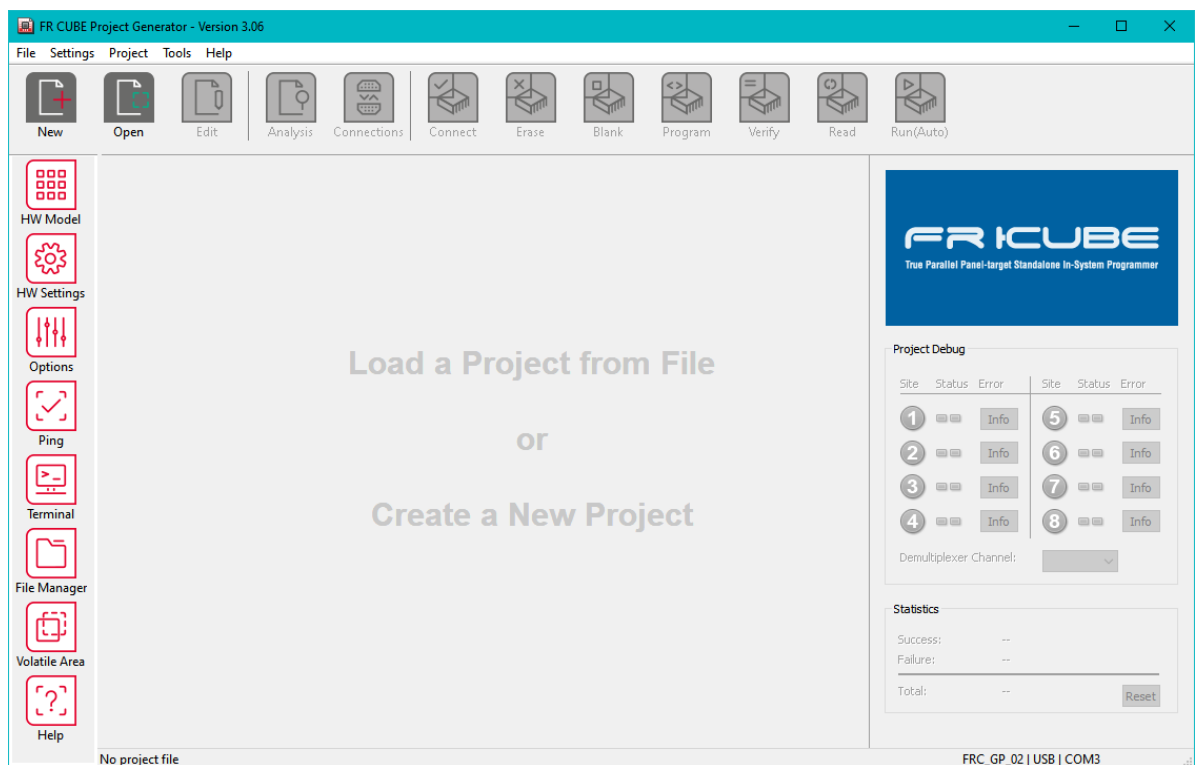
You must copy these files to the unit’s internal memory: the driver file must be copied to the unit’s **\drivers** folder, and the license file to the unit’s **\sys** folder. Please refer to “FR CUBE File System” chapter.

2 Getting Started

2.1 Guided Tutorial

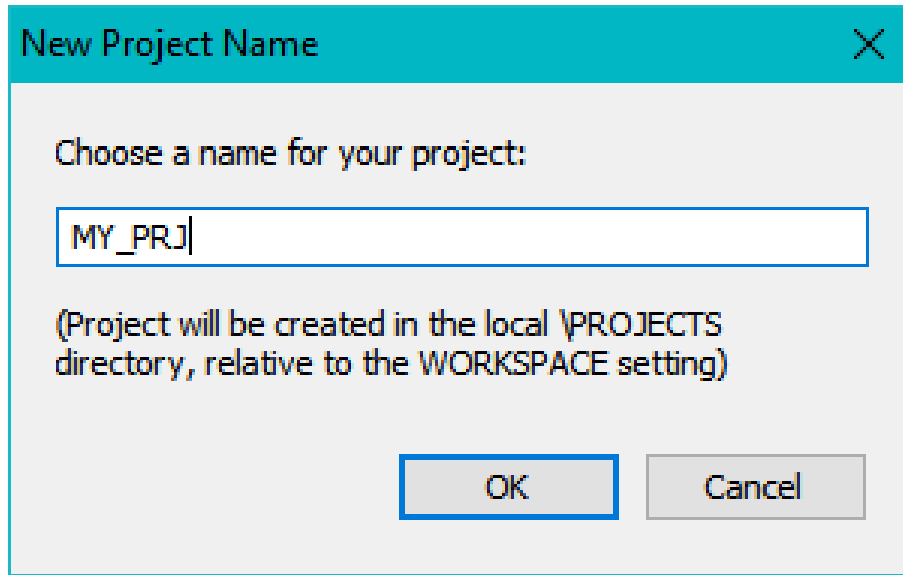
The following tutorial will guide you through the steps required to set up your FR CUBE programmer and create your first programming project.

- 1. Install Software**
Insert the Setup CD into your PC and install the FR CUBE software.
- 2. Launch the Project Generator**
Launch the Project Generator application, that is located under **Programs > SMH Technologies > FR CUBE Software > Project Generator**.



3. Create a New Project

Select File > New Project, give a name to your programming project, and then follow the Project creation Wizard steps.



New Project Name

Choose a name for your project:

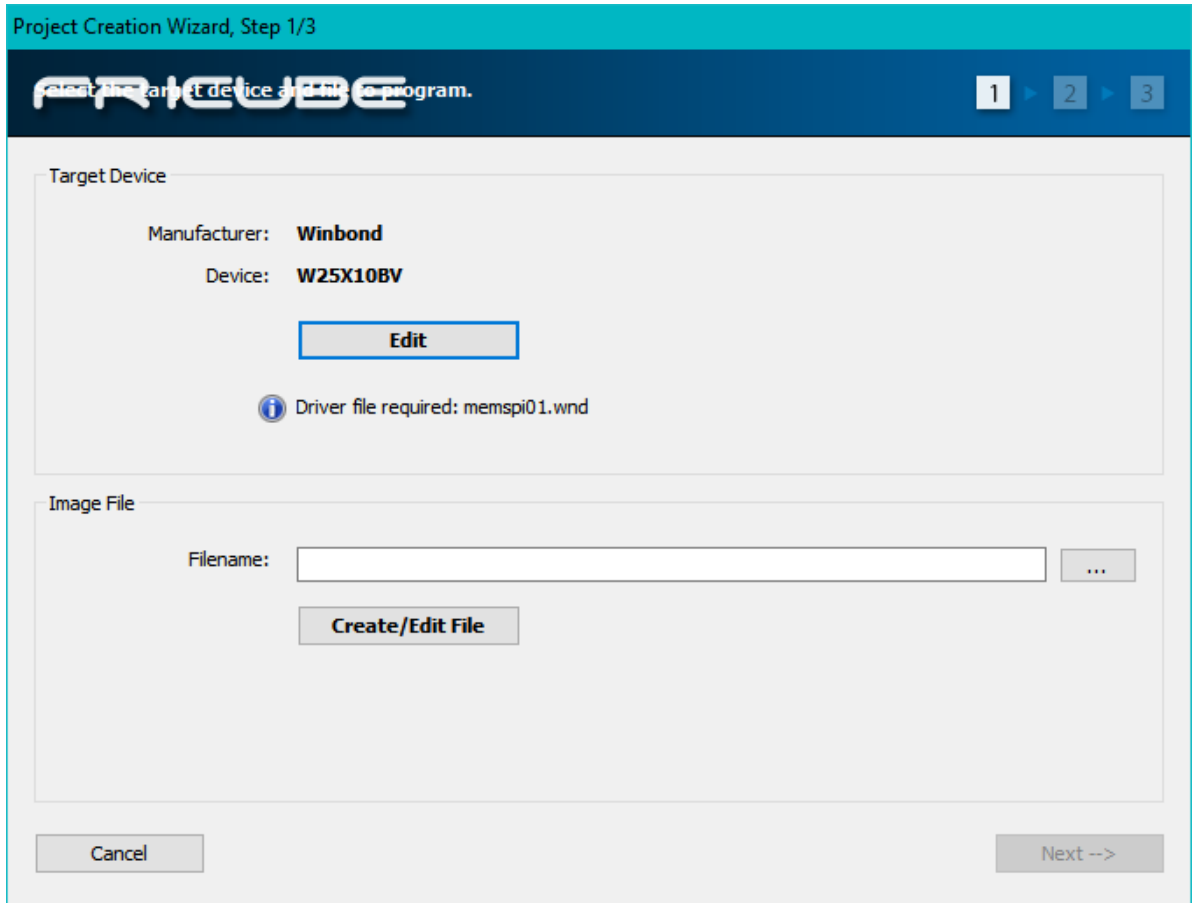
MY_PRJ

(Project will be created in the local \PROJECTS directory, relative to the WORKSPACE setting)

OK Cancel

4. Create a New Project, Step 1 of 3

In the first Wizard step, specify the target device, by clicking “Edit” button.



Project Creation Wizard, Step 1/3

1 2 3

Target Device

Manufacturer: **Winbond**

Device: **W25X10BV**

Edit

i Driver file required: memspi01.wnd

Image File

Filename: ...

Create/Edit File

Cancel **Next -->**

Next, specify the file to be programmed (image file). To create an image file, clicking the **“Create/Edit File”** button. A dedicated window will open.

Image File Creation

Input Data

Data Type	Parameters	Buffer Range
FILE	128KB.bin, BIN, 00000000-0001FFFF	00000000-0001FFFF

Buttons: Add, Edit, Remove

WriteNow! Image File Requirements for W25X10BV Device

```

Device Memory (8-bit addr):      00000000-0001FFFF
Buffer Memory (8-bit addr):     00000000-0001FFFF
  
```

Output File

Format: WriteNow! Image

Filename: C:\SMH Technologies\images\MY_IMG.wni

Fill unused locations with: HFF

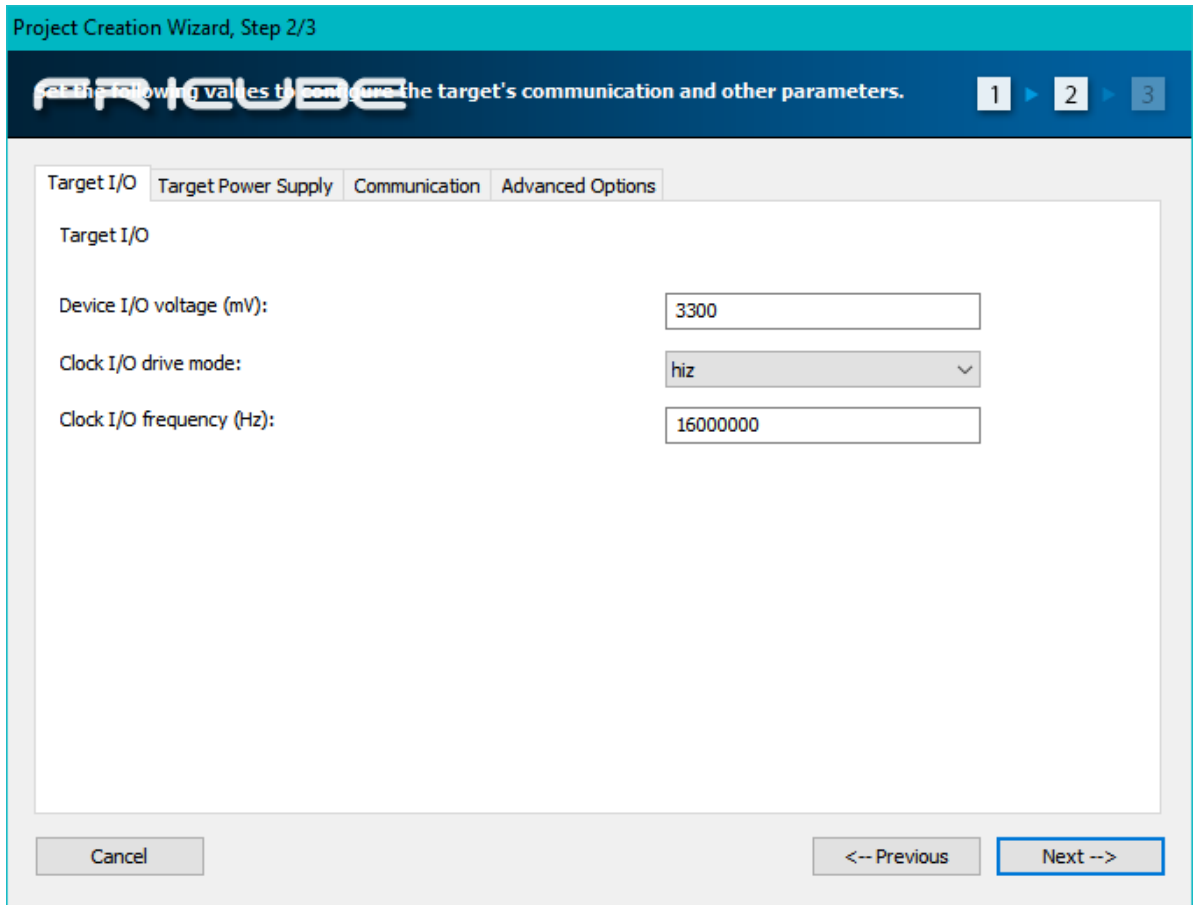
Create a conversion report

Buttons: OK, Cancel

In the Output File section, specify the output filename by clicking the “...” button. Use the **“Add”** button to compose the data that will compose the Image file. Use the “...” button to specify the name of the Image file. When done, click **“OK”** to return to the Wizard, and proceed to Step 2.

5. Create a New Project, Step 2 of 3

In this step, specify target parameters and connection values. The Wizard will automatically fill all data with typical values for the selected target device.



Project Creation Wizard, Step 2/3

Enter the following values to configure the target's communication and other parameters.

1 2 3

Target I/O Target Power Supply Communication Advanced Options

Target I/O

Device I/O voltage (mV): 3300

Clock I/O drive mode: hiz

Clock I/O frequency (Hz): 16000000

Cancel <-- Previous Next -->

The number of tabs displayed in this window depends on the selected target device; however, three tabs (“**Target I/O**”, “**Target Power Supply**” and “**Communication**”) are always present and will be briefly discussed below.

The first tab is “**Target I/O**”. The “**Device I/O voltage**” setting specifies the voltage of the ISP lines. You should check the target board schematics, or ask the board developer about this value. The allowed voltage also depends on the selected target device.

The “**Clock I/O drive mode**” setting allows you to decide how the SxL04 ISP line is driven (the x index refers to the programming site; see “ISP Connectors” on page 59). This line can be used as an auxiliary ISP line (to provide a clock to the target device), as a generic I/O line, or as a highimpedance output (no electrical driving). When used as output line (set to high or low), it could be used, for example, to disable the external watchdog circuit in the target board. When used as clock out, you can specify the output frequency in the “**Clock I/O frequency**” field. We suggest leaving this line floating (HiZ) when not used, in order to decrease electrical noise on other ISP lines.

Project Creation Wizard, Step 2/3

Set the following values to configure the target's communication and other parameters. 1 2 3

Target I/O Target Power Supply Communication Advanced Options

Target Power Supply

Target power supply voltage (mV):

Power up time (ms):

Power down time (ms):

Cancel <-- Previous Next -->

If you decide to power the target board through the FR CUBE power supply line (SxPPS), specify in the **“Target Power Supply”** tab the electrical and timing parameters of the target power supply line. FR CUBE is able to power the target board through a dedicated programmable power supply output line per site. The voltage of the programmable power supply line (**“Target power supply voltage” setting**) can be in the range 1700mV to 13000mV. Each programmable power supply line features an internal voltage limiter that cuts the voltage output in case of short circuits or overloads. The current output is limited to about 400mA.

The **“Power up time”** setting specifies the delay between the programmable power supply line turning on and the first operation on the ISP lines. The purpose of this parameter is to wait for the power supply to become stable, before starting ISP programming. This parameter is useful when large capacitors are mounted in the target board’s power line. The **“Power down time”** setting acts in similar way: it sets the delay between the programmable power supply line turning off and subsequent operations.

Project Creation Wizard, Step 2/3

FR CUBE

At the following values to ensure the target’s communication and other parameters.

1 2 3

Target I/O Target Power Supply Communication Advanced Options

Communication

Communication protocol: SPI

Bitrate (Hz): 10000000

Fast programming mode (VPP pin): disable

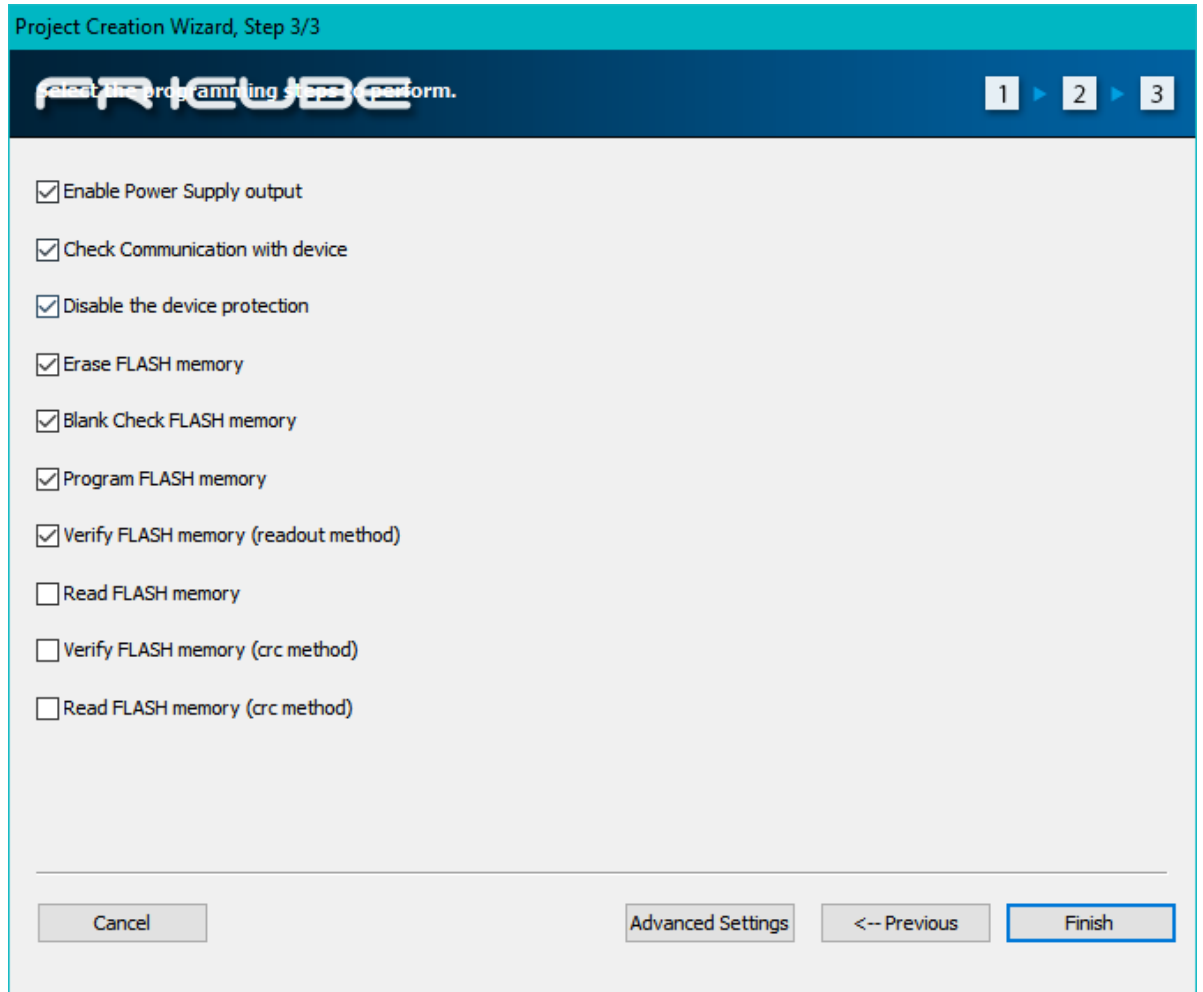
Cancel <-- Previous Next -->

The content of the **“Communication”** tab depends on the selected target device. It allows you to select the communication protocol that will be used for programming (some target devices may provide more than one communication protocol) and its related settings, usually the communication speed and other parameters. Usually, the higher the communication speed, the shorter/better the ISP cabling must be.

After carefully checking all of the parameters values, proceed to Step 3.

6. Create a New Project, Step 3 of 3

In this step you select which programming operation to perform on the target.



Project Creation Wizard, Step 3/3

1 2 3

1. Enable Power Supply output

2. Check Communication with device

3. Disable the device protection

4. Erase FLASH memory

5. Blank Check FLASH memory

6. Program FLASH memory

7. Verify FLASH memory (readout method)

8. Read FLASH memory

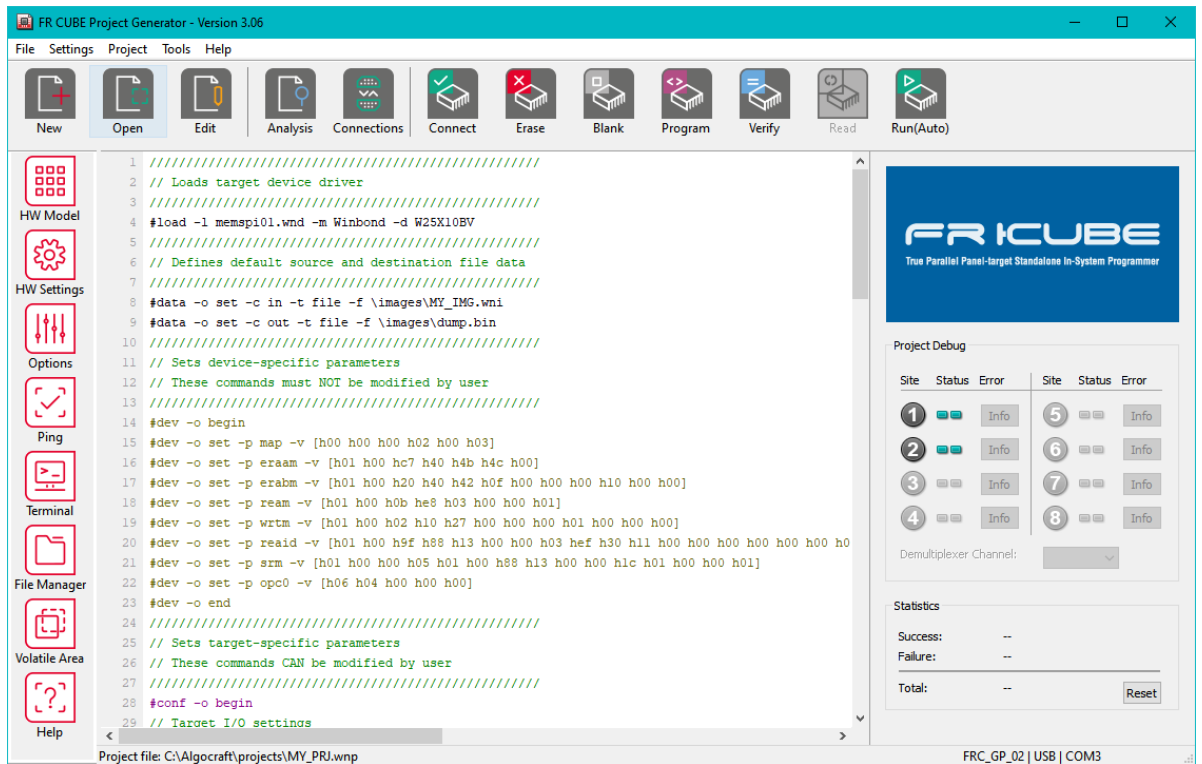
9. Verify FLASH memory (crc method)

10. Read FLASH memory (crc method)

Cancel Advanced Settings <-- Previous Finish



Click **“Finish”** to end the Wizard. At this point, a FR CUBE Programming Project will be created in the **\Projects** directory, relative to the Project Generator application location.



7. Configure your FR CUBE Instrument

Choose **Project > Select FR CUBE Model**, and specify your FR CUBE model and communication settings with the PC. Currently, FR CUBE can be connected only through a serial port. FR CUBE communicates at **115,200** bps by default. LAN and USB connections will be supported soon through a free software upgrade.

Model Selection


Model

Select a model:
FRC_GP_08

FR CUBE Gang In-System Programmer
Eight output sites

Select the external module:
NONE

No external module



Communication Settings

RS-232 Ethernet USB

Port: COM4 IP address: 192 . 168 . 1 . 153 Virtual COM Port: COM3

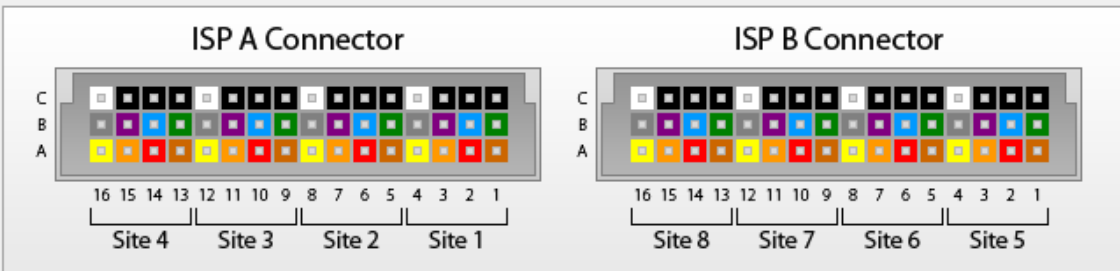
Baud rate: 115200 Port: 2101











OK Cancel

8. Connect to Target Device

Connect FR CUBE to your target system through the ISP connector(s). To view the connections for your selected target device, select **Debug > Show ISP Connections**.

FRC_GP_08 ISP Connections for Winbond W25X10BV [memspi01]



Color	Signal Name	Target Signal Name	Target Signal Description
	SxL01	#W	Write Protect (Programmer set to inactive level while programming the device)
	SxL02	CLK	Serial Clock
	SxL03	DO(IO1)	Serial Data Output
	SxL04	CLOCKOUT	
	SxL05	#S	Chip Select
	SxL06	DI(IO0)	Serial Data Input
	SxL07	#HOLD	Hold (Programmer set to inactive level while programming the device)
	SxPPS	VDD	Target VDD (Programmer can provide a power supply for the target system)
	SxGND	GND	GND
	SxRLY	RLY	Relay output

OK

9. Startup FR CUBE

Connect FR CUBE to your PC through the provided serial cable. Finally, power up FR CUBE using the provided power supply

10. Program the Target Device

Select **Debug > Run Project**. The Project file (.wnp) and Image file (.wni) will be automatically uploaded to FR CUBE and the project will be executed. Your target device(s) will be programmed.

In case of programming errors, or to change programming parameters/operations, you can relaunch the Project Wizard and review the Project settings.

2.2 Manual Project Editing

The Project file created by the Project Wizard is located, by default, in the **\Projects** directory, relative to the Project Generator application location (this location can be changed by specifying a different “workspace” path: to do so, in the Project Generator, select **Project > Edit Miscellaneous Settings** and modify the **Workspace** setting).

The generated project file is a text file and, if necessary, can be edited using any text editor. Please note, however, that once the file is modified by the user, it can be opened by the Project Generator but the Project Wizard will not be available.

2.3 Where to Go from Here

In this chapter, you have learnt how to use the Project Generator to create and execute a typical programming project. Additionally, FR CUBE can be controlled in three other ways:

1. By manually sending commands and receiving answers, using the Project Generator Terminal or any other terminal application (for more information, see “Commands” chapter);
2. By configuring the instrument so that it can work in standalone, that is without a connection to a PC (for more information, see “Standalone Mode” chapter);
3. By building your own PC software that interfaces to the instrument (for more information, see “FR CUBE API” chapter).

3 Commands

3.1 Overview

FR CUBE is a slave unit and is always awaiting for a new command incoming from the master (PC). When the programmer receives a SOF (Start Of Frame) character (**#**), indicating the start of a new command, it loads all incoming characters in a buffer until the reception of the return character (**\n**, ASCII code **h0A**).

Maximum command length is 256 characters.

After reception of the return character, the programmer interprets and executes the received command; depending on the execution of the received command the protocol will answer to the master in three different ways.

1. If the command is correctly executed, the programmer answers with an OK frame.
2. If the command execution generates errors, the programmer answers with an ERR frame.
3. If the command takes long to execute, the programmer periodically answers with a BUSY frame, until command execution is over and an OK or ERR frame is answered.

All commands and answers are case-insensitive.

3.2 Command Syntax

A FR CUBE command begins with the SOF character (**#**), followed by the command name, followed by zero or more command switches, and ends with the return character (**\n**). This is an example of a FR CUBE valid command:

```
#status -o ping{\n}
```

3.3 OK Answer

An OK answer is composed of zero or more characters, followed by the **>** character, followed by the return character (**\n**). This is an example of a FR CUBE OK answer: **pong>{\n}**

3.4 ERR Answer

An ERR answer is composed of zero or more characters (usually the hexadecimal error code), followed by the **!** character, followed by the return character (**\n**). This is an example of a FR CUBE ERR answer: **h40000103!{\n}**

3.5 BUSY Answer

A BUSY answer is sent by the programmer to the PC if a command take some time to execute. A BUSY answer is sent at most every 3 seconds. If no OK, ERR or BUSY answer is sent within 3 seconds from the last command sent to the programmer, a communication error has probably occurred.

A BUSY answer is composed of zero or more characters, followed by the ***** character, followed by the return character (**\n**).

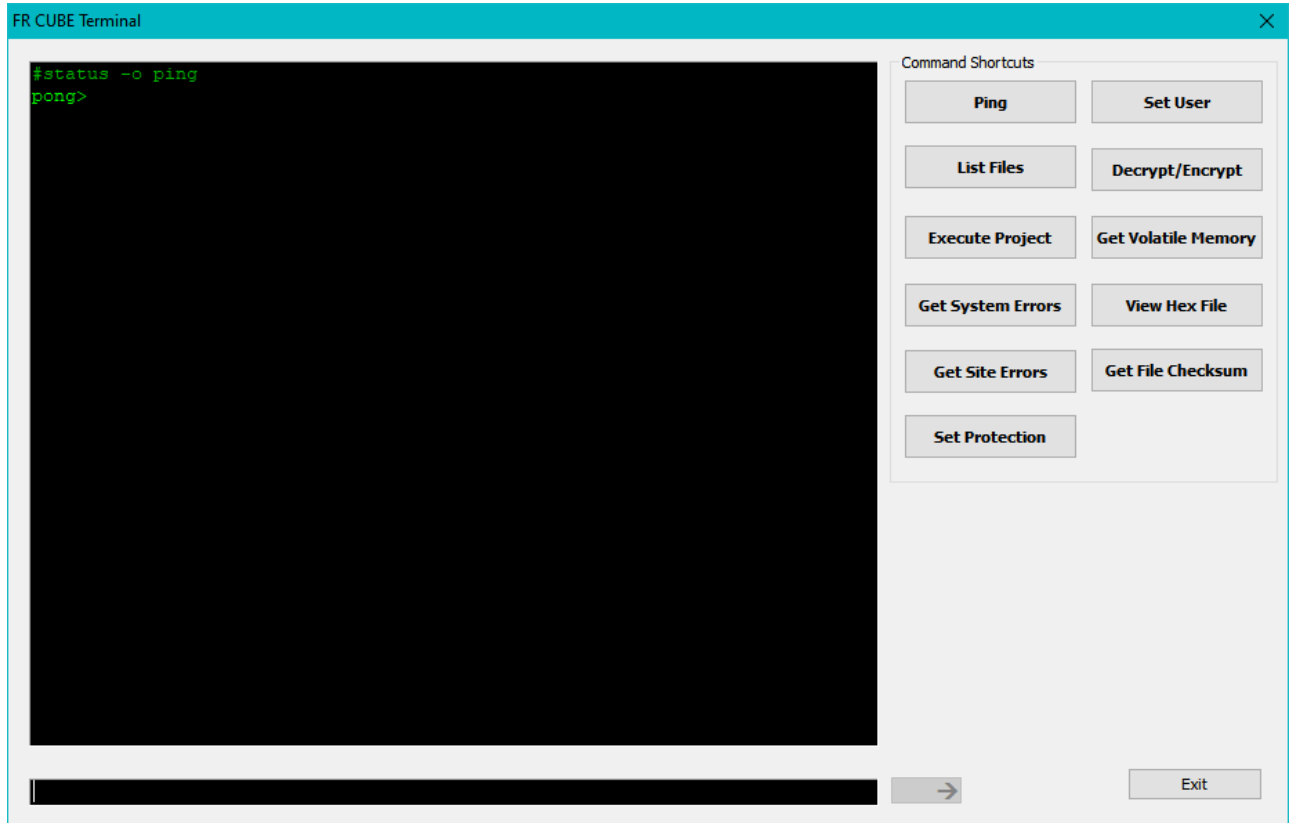
This is an example of a FR CUBE BUSY answer:

```
*{\n}
```

A valid answer always ends with two characters: >{\n}, !{\n} or *{\n}, depending on whether an OK, ERR or BUSY frame is sent to the host. Additional return characters (\n) may be present in the answer, but they don't signal the end of the answer.

3.6 FR CUBE Terminal

Commands can be sent (and answers received) using any terminal application. For your convenience, the Project Generator application includes a Terminal window that will simplify the communication with the instrument. Just select **Tools > FR CUBE Terminal** to open the Terminal window.



3.7 Command Reference

The following pages list all of the FR CUBE commands, grouped by function, together with their syntax and usage examples.

3.8 Data In/Out Commands

Syntax

```
#data -o set -c <direction> -t file -f <filename>
#data -o set -c <direction> -t volatile
```

Parameters

<direction> in or out.
<filename> Filename on the instrument's file system.

Description

Specify the source and destination of the programming data.

Examples

Sets the input image file to be programmed, and subsequently programs it:

```
#data -o set -c in -t file -f \images\myfile.wni
>
#prog -o cmd -c program -m flash -s h8000 -t h8000 -l h8000
>
```

Sets the output file to receive binary data, and subsequently reads data from the target device:

```
#data -o set -c out -t file -f \images\dump.bin
>
#prog -o cmd -c read -m flash -s h8000 -t h8000 -l h8000
>
```

3.9 Execution Command

Syntax

```
#exec -o prj -f <project> -s <sites>
```

Parameters

<project> The Project filename to execute.
 <sites> A 8 bit value indicating the programming sites to be enabled.

Description

Executes the specified Project over the specified programming sites. In case of error, a 32 bit value is returned. This value indicates whether the error is site-specific (bit 29 = 1) or system-specific (bit 29 = 0). If the error is site-specific, the 8 least significant bits (bits from 7 to 0) signal whether programming in the corresponding programming site (bit 7 = programming site 8, bit 0 = programming site 1) was successful (bit = 0) or not (bit = 1). To retrieve error messages, use the **#status -o get -p err -v <site> -l <command>**, where is 1 to 8 to retrieve a specific programming site error, or 0 to retrieve a system error. is the error detail information that is returned and can be **1, 2, 3**.

Examples

Executes the Project "myprj.wnp" on programming sites 1, 2, 3, 4:

```
#exec -o prj -f \projects\myprj.wnp -s h0f
h20000003!
```

In this case, the returned error indicates that there are site-specific errors (bit 29 = 1) and that the sites where errors occurred are sites 1 and 2. To retrieve detailed error information about site 1, for example, the following command can be sent:

```
#status -o get -p err -v 1 -l 2
h5000001,23,"Error: Timeout occurred"
>
```

The answers indicates that Project line 23 issued a **h5000001** error, and the text between quotes explains the error

3.10 File System Commands

Syntax

```
#fs -o rmdir -d <directory>
#fs -o mkdir -d <directory>
#fs -o dir -d <directory>
#fs -o del -f <filename>
#fs -o send -d <filename>
#fs -o receive -d <filename>
```

Parameters

<directory> Full path of a directory.
<filename> Full path of a filename.

Description

Allow to perform various operations on the programmer's file system.

Examples

Shows the contents of the programmer's root directory

```
#fs -o dir -d \  
2010/06/21 16:35:06 [DIR]          projects  
2010/06/21 16:35:16 [DIR]          sys  
2010/06/21 16:35:20 [DIR]          images  
2010/06/21 16:35:26 [DIR]          drivers  
>
```

3.11 Programming Commands

Syntax

```
#load -l <driver> -m <manufacturer> -d <device>
#dev -o begin
#dev -o end
#dev -o set -p <parameters> -v <value>
#conf -o begin
#conf -o end
#conf -o set -p <parameters> -v <value>
#prog -o begin
#prog -o end
#prog -o cmd -c pps -v <pps value>
#prog -o cmd -c connect
#prog -o cmd -c disconnect
#prog -o cmd -c unprotect
#prog -o cmd -c erase -m <mem type> -t <tgt addr> -l <len>
#prog -o cmd -c blankcheck -m <mem type> -t <tgt addr> -l <len>
#prog -o cmd -c program -m <mem type> -s <src addr> -t <tgt addr> -l <len>
#prog -o cmd -c verify -v <ver mode> -m <mem type> -t <tgt addr> -l <len>
#prog -o cmd -c read -m <mem type> -s <dst addr> -t <tgt addr> -l <len>
```

Parameters

<driver>	Filename of the .wnd driver.
<manufacturer>	Target device's silicon manufacturer.
<device>	Target device code.
<parameter>	Target parameter to set.
<value>	Value of the corresponding parameter.
<pps value>	on or off.
<mem type>	Target memory type.
<tgt addr>	Target start address.
<len>	Data length.
<src addr>	Source (buffer) start address.
<ver mode>	Verify mode: read or chks.
<dst addr>	Destination start address.

Description

Perform various programming settings and operations on the target device.

3.12 Status Commands

Syntax

```
#status -o ping #status -o get -p err -v <site> -l <errlevel>
```

Parameters

<site> 1 to 8 to get programming site errors.
<errlevel> Use 0 to return system errors. 1 to 3.

Description

Get instrument status or error information. When retrieving error information, one or more error lines (depending on the <errlevel> parameter) are returned. Each line begins with a 32-bit code, which codifies the following information:

Bit 31: Reserved

Bit 30: If 1, an error message in text format is available.

Bit 29: If 1, the error is programming site specific.

Bit 28: If 1, the error is driver (programming algorithm) specific.

Bit 27: If 1, the error is a system fatal error.

Bits 26 to 24: Reserved.

Bits 23 to 0: Error code. If bit 29 is 1, then bits 7 to 0 signal whether programming in the corresponding programming site (bit 7 = programming site 8, bit 0 = programming site 1) was successful (bit = 0) or not (bit = 1).

Examples

Pings the instrument to check if communication is OK:

```
#status -o ping
pong>
```

Retrieves the last generated errors, on programming site 1, with different error levels:

```
#status -o get -p err -v 1 -l 1
H50000023
>
#status -o get -p err -v 1 -l 2
H50000023,71,"Connection Error."
>
#status -o get -p err -v 1 -l 3
H50000023,71,"Connection Error.", "algo_api", 337
H10000000,71,"", "st701_cmds", 432
H10000000,71,"", "st701_entry", 287
H10000000,71,"", "st701_icc", 208
H10000001,71,"", "hal_iccl", 144
>
```

3.13 System Commands

Syntax

```
#sys -o set -p br -v <baud rate>
#sys -o get -p br
#sys -o get -p sn
#sys -o get -p ver -v <code>
#sys -o set -p lliop -s <prj sel> -f <prj filename>
#sys -o get -p lliop -s <prj sel>
```

Parameters

<baudrate> 9600, 19200, 38400, 57600, 115200, or 230400.
<code> sys or driver.
<prj sel> Project number, as selected by the PRJ_SEL[5..0] lines on the Low-Level Interface connector.
<prj filename> Project file associated to setting.

Description

Set or get instrument's internal parameters.

Examples

Sets a new serial baud rate:

```
#sys -o set -p br -v 115200
>
```

Retrieves the instrument's serial number:

```
#sys -o get -p sn
00100>
```

Associates the project test.wnp to the project number 1:

```
#sys -o set -p lliop -s 1 -f \projects\test.wnp
>
```

3.14 Time commands

Syntax

```
#time -o set -p date -d <date>
#time -o set -p time -d <time>
#time -o get -p date
#time -o get -p time
```

Parameters

<date> A date in the format yyyy/mm/dd.
<time> A time in the format hh:mm:ss.

Description

Set or get the instrument's date and time. Once set, the date and time are maintained even when the instrument is powered off.

Examples

Sets the date/time to June 1st, 2023, at noon:

```
#time -o set -p date -d 2023/06/01
>
#time -o set -p time -d 12:00:00
>
```

Retrieves the instrument's date and time:

```
#time -o get -p date
2023/06/01>
#time -o get -p time
12:02:05>
```

3.15 Volatile Memory Commands

Syntax

```
#volatile -o write -s <site> -a <start address> - l <len> -d <data>  
#volatile -o read -s <site> -a <start address> - l <len>
```

Parameters

<site> Programming site. 1 to 8 to set specific site data, 0 to set the same data for all sites.

<start address> Volatile memory starting address.

<len> Data length.

<data> A data array.

Description

Read and write data from/to the instrument's volatile memory.

Examples

Uses the volatile memory on site 1 to store the target board's MAC address:

```
#volatile -o write -s 1 -a h0 -l 6 -d [h00 h90 h96 h90 h48 h85]  
>
```

Retrieves data from site 1 volatile memory:

```
#volatile -o read -s 1 -a h0 -l 6 1, [h00 h90 h96 h90 h48 h85]  
>
```


4 Standalone Mode

4.1 Overview

FR CUBE can work with no connection to a PC (standalone mode). In standalone mode, the instrument is controlled through a low-level connection interface.

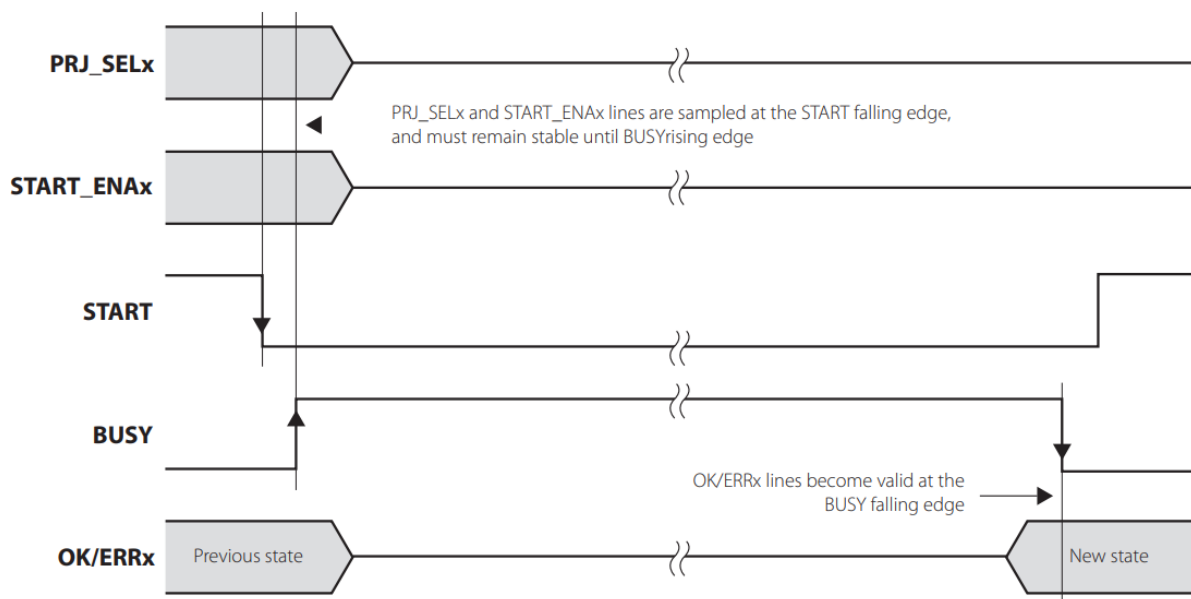
4.2 Signals

Signals needed to control the instrument in standalone mode are located in the “Low-Level Interface” connector (see “Connectors” chapter for the connector pinout on the various FR CUBE models) and are explained below.

Signal level is 0-5V. All lines are isolated (referenced to GND)

PRJ_SELx lines (input):	Define which project to execute (see “Project Assignment” later on this chapter).
START_ENAx lines (input):	Select which programming site(s) to enable. Active low.
START line (input):	Executes the project specified by PRJ_SELx lines on the programming site(s) enabled by START_ENAx lines. Active low.
BUSY line (output):	Indicates that a project is being executed. Active high.
PASS/FAULTx lines (output):	Valid at the end of the project execution (when BUSY is low). Indicate, for each programming site(s), the success state of the programming project. (OK = high, ERR = low).

The following diagram illustrates the timing for the Low-Level Interface signals.



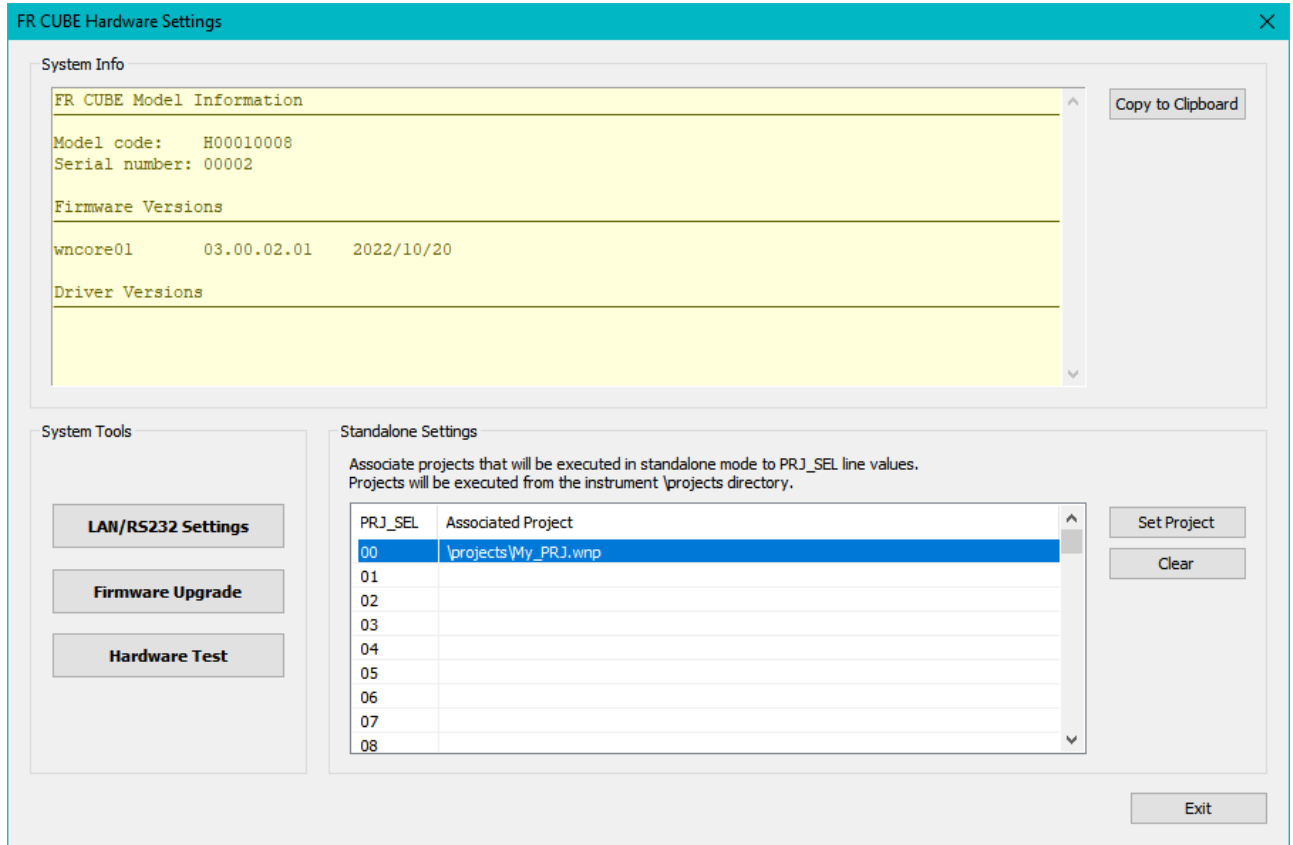
Low-Level Interface Signals Timing

UNIVERSAL PRODUCTION IN-SYSTEM PROGRAMMING

4.3 Project Assignment

Before working in standalone mode, you must associate PRJ_SELx lines to a Project filename to execute.

To do so, in the FR CUBE Project Generator application select **Project > Hardware Settings**. In the window that will appear, associate PRJ_SEL values to project names by clicking the “**Set Project**” button for each PRJ_SEL configuration you wish you setup.



5 FR CUBE API

5.1 Overview

You can build your own PC software that interfaces to the instrument, by using the provided FR CUBE Application Programming Interface (API). The FR CUBE API consists of a series of functions, contained in the **fr_comm** DLL, which allow you to set up and control the programmer.

The **fr_comm** DLL is located in the **Developer** folder, relative to the FR CUBE software installation path. In the same folder you can find the source code of sample applications, in various programming languages, that use the **fr_comm** DLL.

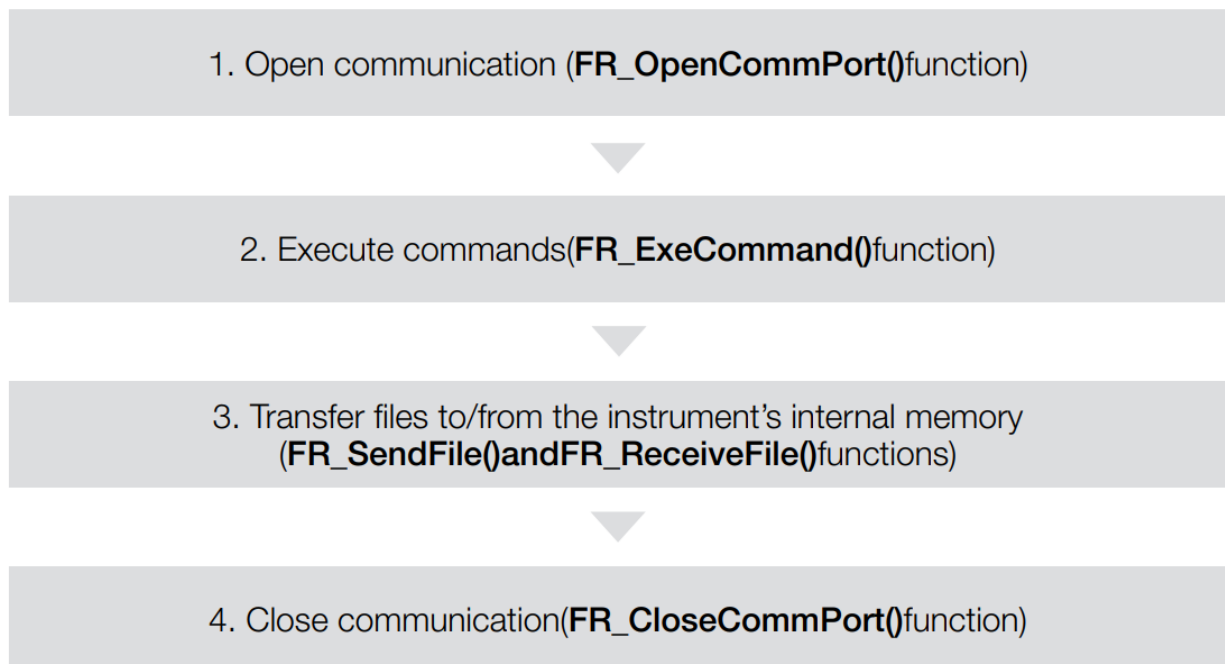
Additionally, a command line application (**fr_cmds.exe**) is provided, which reads a programming command from the stdin, sends the command to the instrument, and writes the command answer on the stdout.

5.2 Including the API in Your Application

To use the FR CUBE API, you must:

- Include the “**fr_comm.lib**” and “**fr_comm.h**” files in your application project (only needed for Visual C++ projects);
- Copy the “**fr_comm.dll**” file in the same folder of your application executable (this file must also be redistributed with your application).

The typical program flow for interfacing with FR CUBE is the following:



5.3 Function Reference

API functions are listed and explained alphabetically in the following pages.

5.4 FR_CloseCommPort()

Prototype

```
FR_COMM_ERR WINAPI FR_CloseCommPortA (FR_COMM_HANDLE handle);
```

```
FR_COMM_ERR WINAPI FR_CloseCommPortW (FR_COMM_HANDLE handle);
```

Description

Closes the communication channel with the instrument.

Return Value

0 The function call was successful.
!=0 The function call was unsuccessful.
Call the **FR_GetLastErrorMessage()** function to get error information.

Parameters

Handle Communication handle returned by the **FR_openCommPort()** function.

5.5 FR_ExeCommand()

Prototype

```
FR_COMM_ERR WINAPI FR_ExeCommandA (FR_COMM_HANDLE handle, const char *command,  
char *answer, unsigned long maxlen, unsigned long timeout_ms, FR_ANSWER_TYPE *type);
```

```
FR_COMM_ERR WINAPI FR_ExeCommandW (FR_COMM_HANDLE handle, const wchar_t *command,  
wchar_t *answer, unsigned long maxlen, unsigned long timeout_ms, FR_ANSWER_TYPE *type);
```

Description

Executes a FR CUBE command. This function automatically sends a command to the instrument and returns the answer read back from the instrument. This function combines the **FR_SendFrame()** and **FR_GetFrame()** function in a single call.

Return Value

0 The function call was successful.
!=0 The function call was unsuccessful.
Call the **FR_GetLastErrorMessage()** function to get error information.

Parameters

handle Communication handle returned by the **FR_OpenCommPort()** function.

command A valid FR CUBE command.

answer The answer read back from the instrument in response to the command sent.

maxlen Maximum length, in characters, of the answer buffer.

timeout_ms Time (in milliseconds) before the function times out.

type Type of answer received: can be:
FR_ANSWER_ACK (an OK frame was received);
FR_ANSWER_NACK (an ERR frame was received);
FR_ANSWER_TOUT (command timed out before an answer could be received).

5.6 FR_GetFrame()

Prototype

FR_COMM_ERR WINAPI FR_GetFrameA (FR_COMM_HANDLE handle, char *answer, unsigned long maxlen, unsigned long timeout_ms);

FR_COMM_ERR WINAPI FR_GetFrameW (FR_COMM_HANDLE handle, wchar_t *answer, unsigned long maxlen, unsigned long timeout_ms);

Description

Reads the answer to the command sent by the **FR_SendFrame()** function.

Return Value

0 The function call was successful.

!=0 The function call was unsuccessful.
Call the **FR_GetLastErrorMessage()** function to get error information.

Parameters

Handle Communication handle returned by the FR_OpenCommPort() function.

Answer The answer read back from the instrument in response to the command sent.

Maxlen Maximum length, in characters, of the answer buffer.

timeout_ms Time (in milliseconds) before the function times out. Û

5.7 FR_GetLastErrorMessage()

Prototype

```
void WINAPI FR_GetLastErrorMessageA (char *error_msg, unsigned long string_len);
```

```
void WINAPI FR_GetLastErrorMessageW (wchar_t *error_msg, unsigned long string_len);
```

Description

Returns a string containing the last FR CUBE error message.

Parameters

error_msg	The string that will receive the error message.
msg_len	Length, in characters, of the error message buffer.

5.8 FR_ReceiveFile()

Prototype

```
FR_COMM_ERR WINAPI FR_ReceiveFileA (FR_COMM_HANDLE handle, const char *protocol, const char *src_filename, const char *dst_path, bool force_transfer, FR_FileTransferProgressProc progress);
```

```
FR_COMM_ERR WINAPI FR_ReceiveFileW (FR_COMM_HANDLE handle, const wchar_t *protocol, const wchar_t *src_filename, const wchar_t *dst_path, bool force_transfer, FR_FileTransferProgressProc progress);
```

Description

Receives a file from the instrument's internal memory and saves it to the PC.

Return Value

0	The function call was successful.
!=0	The function call was unsuccessful. Call the FR_GetLastErrorMessage() function to get error information.

Parameters

handle	Communication handle returned by the FR_OpenCommPort() function.
protocol	Transfer protocol. Must be "ymodem" .
src_filename	The full filename, including path, of the remote file.
dst_path	The PC path where to store the file.
force_transfer	If TRUE , file transfer will be executed even if a file with the same name and CRC exists on the PC; if FALSE , file transfer will be executed only if necessary.
progress	Address of a callback function that will receive progress information, or 0 if not used.

5.9 FR_SendFile()

5.10

Prototype

```
FR_COMM_ERR WINAPI FR_SendFileA (FR_COMM_HANDLE handle, const char *protocol, const char
*src_filename, const char *dst_path, bool force_transfer, FR_FileTransferProgressProc progress);
```

```
FR_COMM_ERR WINAPI FR_SendFileW (FR_COMM_HANDLE handle, const wchar_t *protocol, const
wchar_t *src_filename, const wchar_t *dst_path, bool force_transfer, FR_FileTransferProgressProc
progress);
```

Description

Sends a file to the instrument's internal memory

Return Value

0 The function call was successful.
 !=0 The function call was unsuccessful.
 Call the **FR_GetLastErrorMessage()** function to get error information.

Parameters

handle	Communication handle returned by the FR_OpenCommPort() function.
protocol	Transfer protocol. Must be "ymodem" .
src_filename	The full filename, including path, of the remote file.
dst_path	The PC path where to store the file.
force_transfer	If TRUE , file transfer will be executed even if a file with the same name and CRC exists on the PC; if FALSE , file transfer will be executed only if necessary.
progress	Address of a callback function that will receive progress information, or 0 if not used.

5.11 FR_sendFrame()

Prototype

FR_COMM_ERR WINAPI FR_SendFrameA (FR_COMM_HANDLE handle, const char *command);

FR_COMM_ERR WINAPI FR_SendFrameW (FR_COMM_HANDLE handle, const wchar_t *command);

Description

Sends a command to the instrument. Use the **FR_GetFrame()** function to retrieve the answer.

Return Value

0 The function call was successful.

!=0 The function call was unsuccessful.

Call the **FR_GetLastErrorMessage()** function to get error information.

Parameters

handle Communication handle returned by the FR_OpenCommPort() function.

command A valid FR CUBE command.

5.12 FR_OpenCommPortA

5.13

Prototype

FR_COMM_HANDLE WINAPI FR_OpenCommPortA (const char *com_port, const char *com_settings);

FR_COMM_HANDLE WINAPI FR_OpenCommPortW (const wchar_t *com_port, const wchar_t *com_settings);

Description

Opens a RS-232, Ethernet or USB communication channel with the instrument.

Return Value

0 The function call was successful.

!=0 The function call was unsuccessful.

Call the **FR_GetLastErrorMessage()** function to get error information.

Parameters

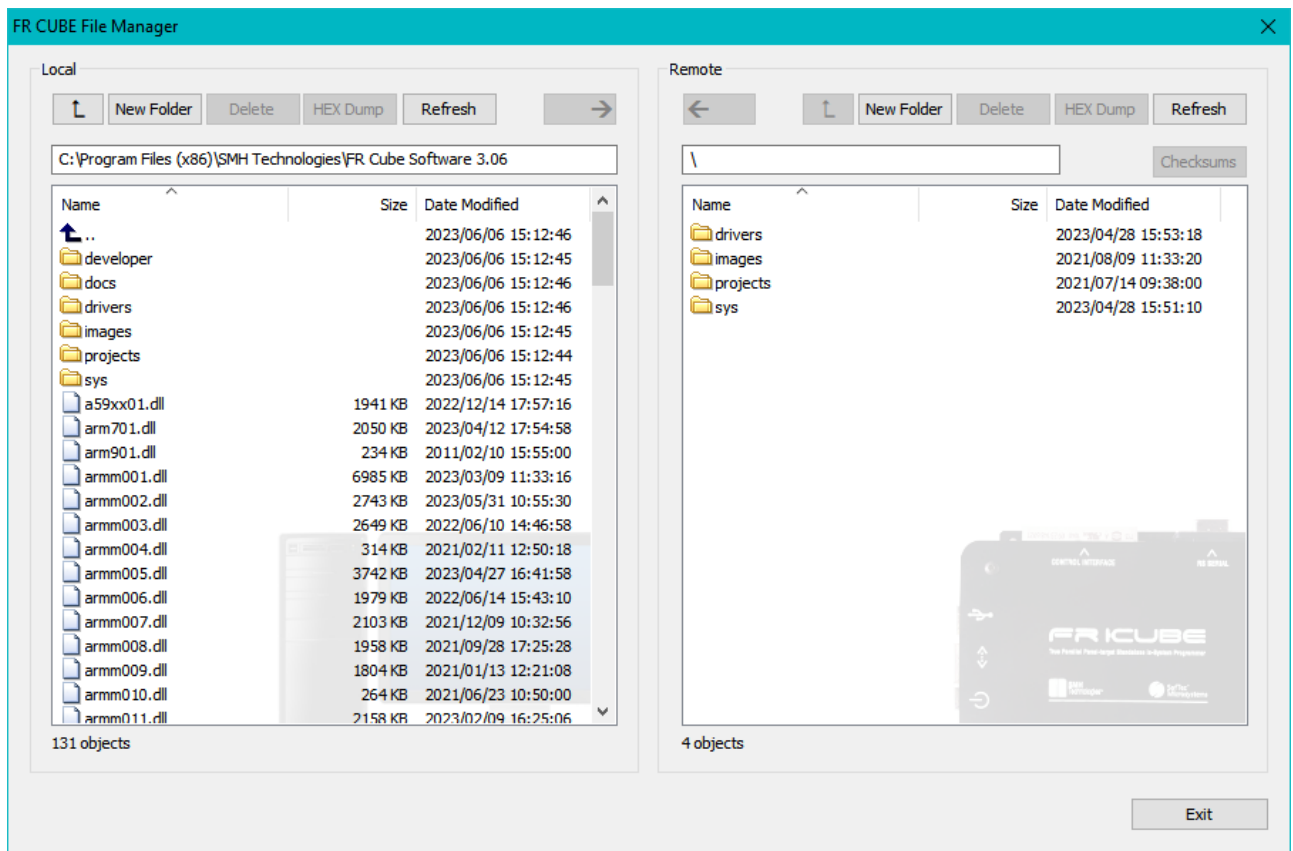
com_port Communication port. Can be "COM", "LAN" or "USB".

com_settings RS-232 settings for "COM" port (e.g.: "COM1,115200"); Ethernet settings for "LAN" port (e.g.: "192.168.1.100:2101"); Empty string for "USB" port.

6 FR CUBE File System

6.1 Overview

FR CUBE has a large, built-in non-volatile memory, used to store the various files required by the instrument: programming projects, image files, etc. This memory is organized by a file system. You can explore the FR CUBE files either by using a Terminal application and sending file-system related commands, or (more simply) by using the File Manager window of the Project Generator application. The File Manager window allows you to easily see the instrument file structure and transfer files with the PC. To open the File Manager, choose **Tools > FR CUBE File Manager** from the Project Generator menu.



6.2 File System Structure

The files required by the instrument are organized in various folders, as explained below:

- \drivers folder: contains programming algorithms (.wnd files). These files are provided by SMH Technologies.
- \sys folder: contains systems files, such as programming licenses, firmware files, etc. These files are provided by SMH Technologies.
- \project folder: contains programming projects (.prj files). You create programming projects using the Project Generator application.
- \images folder: contains FR CUBE image files to be programmed to the target (.wni files). FR CUBE image files contain all the information needed to program a target device memory. These files are created by the Project Generator application.

You can create additional folders, but the four folders listed above must always be present on the FR CUBE file system and must not be removed. Additionally, do not remove or rename the contents of the \SYS folder.

7 Variable Data Programming

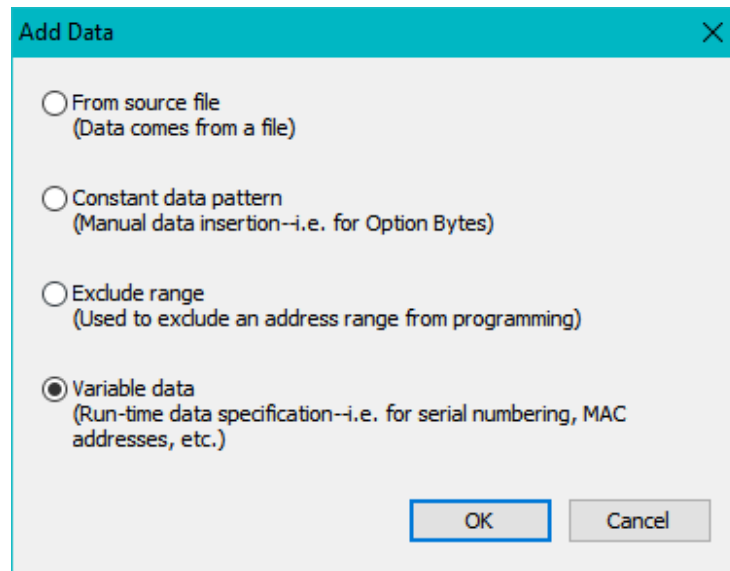
7.1 Overview

FR CUBE has built-in, dedicated memory banks for each programming site. This memory can be used to temporarily store variable data that will be written to the target device during programming. This is useful for serial numbering and for any other variable data that needs to be written to the target device at programming time.

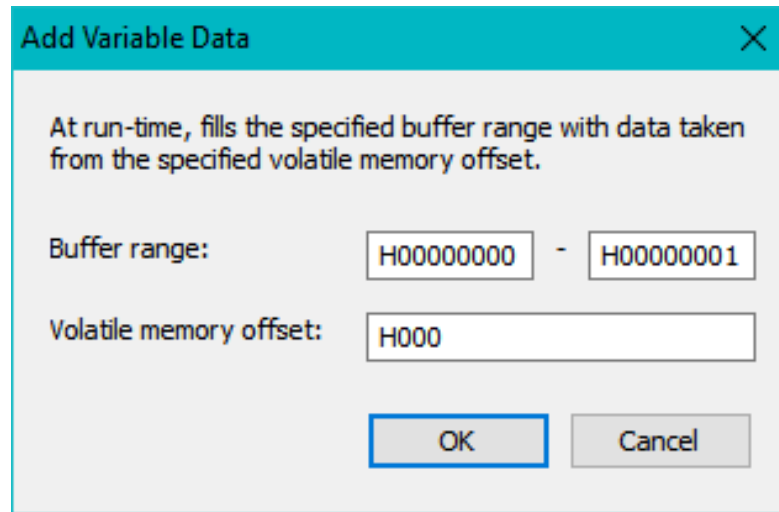
7.2 Usage

To implement variable data programming:

1. Use the Project Creation wizard of the Project Generator application to create your programming project. When creating the FR CUBE Image file, add a variable data record to the output file, as shown below.
- 2.



3. You will be asked for the target device address range to be programmed and the offset of the memory bank that will contain the variable data.



Add Variable Data [X]

At run-time, fills the specified buffer range with data taken from the specified volatile memory offset.

Buffer range: -

Volatile memory offset:

4. Proceed to the end of the Project Creation wizard. Your programming project is now ready to accept variable data.
5. Before executing the project, you must apply the variable data to each of the programming sites. To do so, send the `#volatile -o write` command (for more information, see “Volatile Memory Commands” chapter).

Alternatively, you can skip steps 1 to 3, but you must manually edit your programming project by inserting an appropriate `#data -o set -c out -t volatile` command and subsequent appropriate programming commands (for more information, see “Data In/Out Commands” chapter).

8 Power and Relays Options

8.1 Power Supply Options

FR CUBE can be powered in two ways:

1. With the provided power supply (which supplies 15V DC);
2. By providing a power supply to the PWR pin of the Low-Level Interface connector (see “Low-Level Interface Connector” chapter).

8.2 Relays

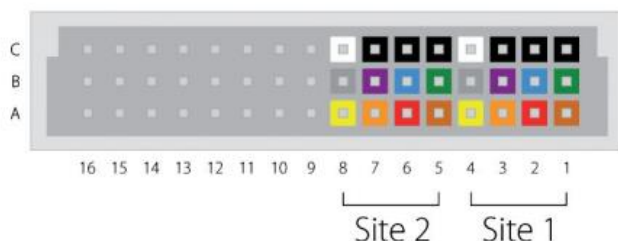
On all FR CUBE models, a special signal (SxRLY) is present (on the “ISP” connector), on every programming site. If the programming site is enabled, this signal is driven to 0V when a **#relay –o close** command is executed, and driven to 5.5V when a the **#relay –o open** command is executed). This is useful for driving an external relay barrier.

9 Connectors

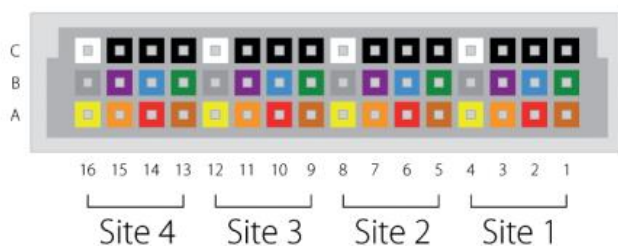
9.1 ISP Connectors

FR CUBE GP02 PASS and FR CUBE GP04 PASS models have one ISP connector; the FR CUBE GP08 PASS model has two ISP connectors.

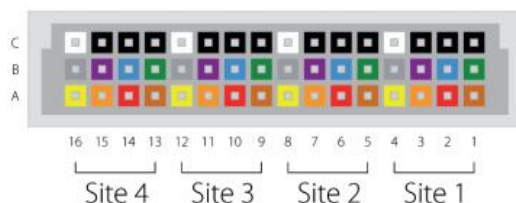
FRC_GP_02 ISP Connector



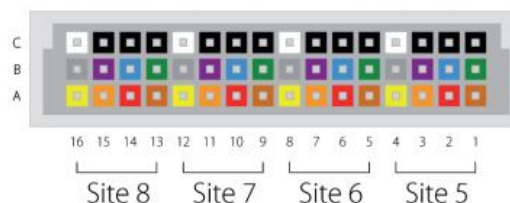
FRC_GP_04 ISP Connector



ISP Connector A



ISP Connector B





ISP Signal Definitions

Color	ISP Connector A Sites		ISP Connector B Sites				FR Signal Name	Target Signal Description		
	S1	S2	S3	S4	S5	S6	S7	S8		
	A1	A5	A9	A13	A1	A5	A9	A13	SxL01	#W - Write Protect (programmer set to inactive level while programming the device)
	A2	A6	A10	A14	A2	A6	A10	A14	SxL02	CLK - Serial Clock
	A3	A7	A11	A15	A3	A7	A11	A15	SxL03	DO(IO1) - Serial Data Output
	A4	A8	A12	A16	A4	A8	A12	A16	SxL04	CLOCKOUT
	B1	B5	B9	B13	B1	B5	B9	B13	SxL05	#S - Chip select
	B2	B6	B10	B14	B2	B6	B10	B14	SxL06	DI(IO0) - Serial Data Input
	B3	B7	B11	B15	B3	B7	B11	B15	SxL07	#HOLD - Hold (programmer set to inactive level while programming the device)
	B4	B8	B12	B16	B4	B8	B12	B16	SxPPS	VDD - Target VDD (Programmer can provide a power supply for the target system)
	C1	C5	C9	C13	C1	C5	C9	C13	SxRLY	GND - GND
	C2	C6	C10	C14	C2	C6	C10	C14	SxGND	RLY - Relay Output
	C3	C7	C11	C15	C3	C7	C11	C15		

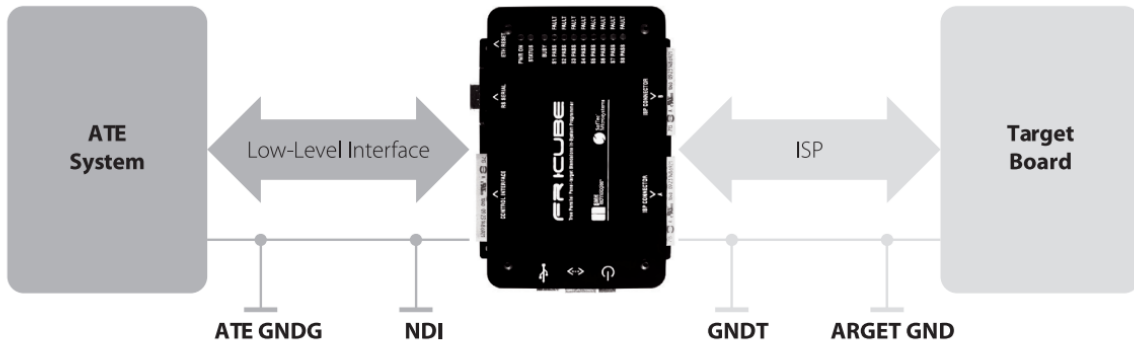
Low-Level Interface Signals

Signal	Description	FR CUBE GP02 Pin	FR CUBE GP04 Pin	FR CUBE GP08 Pin
PWR	Input Power Supply (12-24V)	A5/B5	A5/B5	A5/B5
GND	Power Supply Ground	C5	C5	C5
GNDI	Low-Level Interface Ground	A10/B12/ C15/C16	A10/B12/ C15/C16	A10/B12/ C15/C16
TX_RS232	RS-232 TX (Output)	A16	A16	A16
RX_RS232	RS-232 RX (Input)	B16	B16	B16
PRJ_SEL0	Project Selector 0 (Input, internal pull-up)	B10	B10	B10
PRJ_SEL1	Project Selector 1 (Input, internal pull-up)	C10	C10	C10
PRJ_SEL2	Project Selector 2 (Input, internal pull-up)	A11	A11	A11
PRJ_SEL3	Project Selector 3 (Input, internal pull-up)	B11	B11	B11
PRJ_SEL4	Project Selector 4 (Input, internal pull-up)	C11	C11	C11
PRJ_SEL5	Project Selector 5 (Input, internal pull-up)	A12	A12	A12
START	Project Start (Input, internal pull-up)	A7	A7	A7
START_ENA1	Site 1 Project Start Enable (Input, internal pull-up)	B7	B7	B7
START_ENA2	Site 2 Project Start Enable (Input, internal pull-up)	C7	C7	C7
START_ENA3	Site 3 Project Start Enable (Input, internal pull-up)	-	A8	A8
START_ENA4	Site 4 Project Start Enable (Input, internal pull-up)	-	B8	B8
START_ENA5	Site 5 Project Start Enable (Input, internal pull-up)	-	-	C8
START_ENA6	Site 6 Project Start Enable (Input, internal pull-up)	-	-	A9
START_ENA7	Site 7 Project Start Enable (Input, internal pull-up)	-	-	B9
START_ENA8	Site 8 Project Start Enable (Input, internal pull-up)	-	-	C9
BUSY	Busy (Output, push-pull)	C12	C12	C12
PASS/FAULT1	S1 PASS/FAULT (Output, push-pull)	A13	A13	A13
PASS/FAULT2	S2 PASS/FAULT (Output, push-pull)	B13	B13	B13
PASS/FAULT3	S3 PASS/FAULT (Output, push-pull)	-	C13	C13
PASS/FAULT4	S4 PASS/FAULT (Output, push-pull)	-	A14	A14
PASS/FAULT5	S5 PASS/FAULT (Output, push-pull)	-	-	B14
PASS/FAULT6	S6 PASS/FAULT (Output, push-pull)	-	-	C14
PASS/FAULT7	S7 PASS/FAULT (Output, push-pull)	-	-	A15
PASS/FAULT8	S8 PASS/FAULT (Output, push-pull)	-	-	B15

All low-level interface lines are isolated from system GND (and are referenced to GNDI), except for the PWR line, which is referenced to GND.

9.2 Ground Domains

The following diagram illustrates the two ground domains of the programmer.



ATE and Target Ground Domains

In order to avoid undesired current paths between the programmer and the target board, we suggest to use a power supply with a floating output (ground not referenced to the Earth potential).

10 Specifications

10.1 Electrical Specifications

Feature	Value
Maximum Ratings	
Power supply voltage	30V
ISP SxL0[1..7] voltage	-0.7-6.5V
ISP SxL0[1..7] current	±60mA
ISP SxPPS voltage	-0.7-18V
ISP SxPPS current ^(*)	380mA
ISP SxRLY voltage	-1.0-30V
Low level interface PRJ_SELx, START, START_ENAx, BUSY, PASS/FAULTx voltage	-0.7-6.0V
Operating Ranges	
Power supply voltage	12-24V
ISP SxL0[1..7] voltage	0-5.5V
ISP SxPPS voltage	1.5-15V
ISP SxPPS current	300mA
ISP SxRLY voltage	0-28V
Low level interface PRJ_SELx, START, START_ENAx, BUSY, PASS/FAULTx voltage	0-5.0V
Physical and Environmental	
Operating conditions	0-40°C, 90% humidity max (without condensation)
Storage conditions	-10-60°C, 90% humidity max (without condensation)
EMC (EMI/EMS)	CE, FCC

(*) Current limited, recovers automatically after fault condition is removed.

10.2 ISP Connectors

ISP and Low-Level Interface connectors are DIN48 male connectors. We suggest using the following compatible female connectors.

For wire wrapping:

DIN41612 connector, 3 rows, 48 pins, 180° female, C style

Manufacturer: Conec

Manufacturer Part Number: 122A10619X

Catalog Part Number: Mouser 706-122A10619X

For soldering:

DIN41612 connector, 3 rows, 48 pins, female, R/A C style

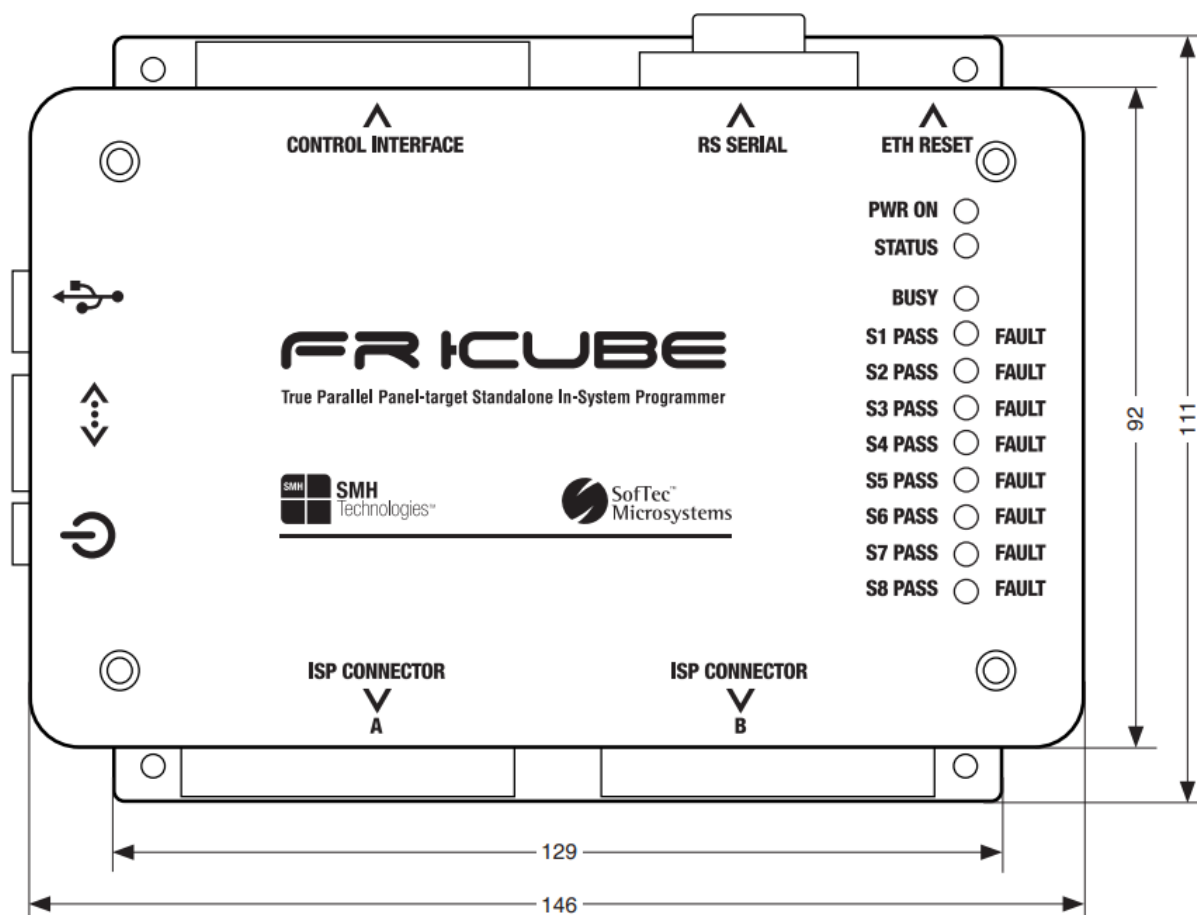
Manufacturer: FCI

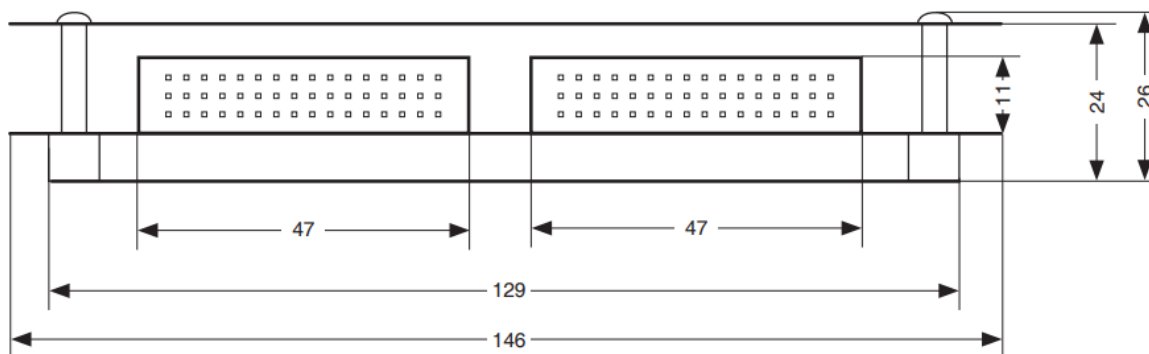
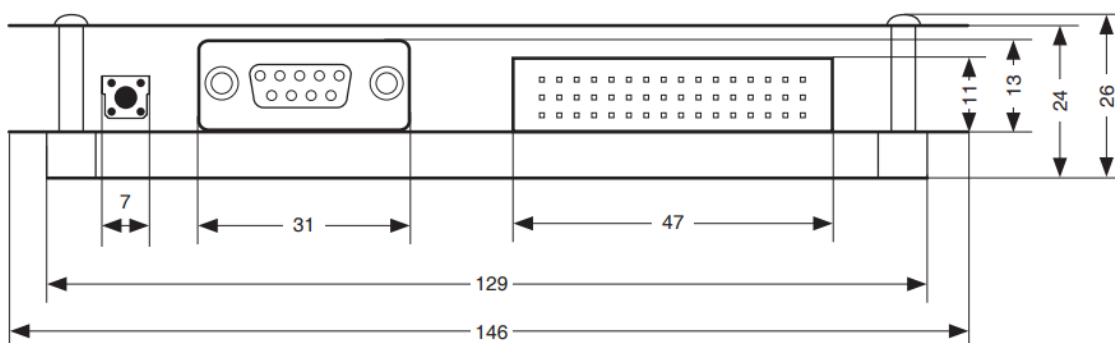
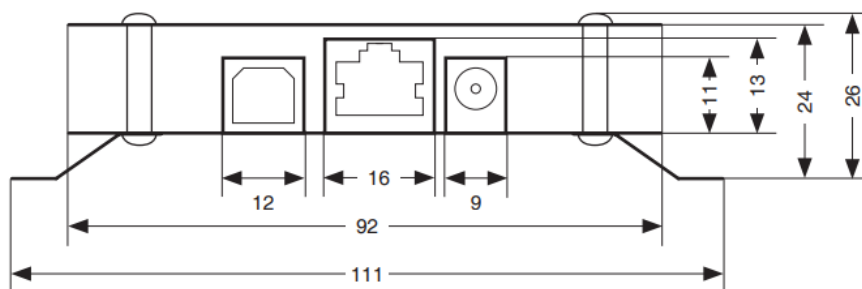
Manufacturer Part Number: 86093488613755E1LF

Catalog Part Number: Mouser 649-8693488637E1L

10.3 Mechanical Specifications

The following drawings detail the mechanical dimensions of the various FR CUBE models.





11 Protected Mode and Data Encryption

11.1 Protected Mode

A security feature has been created to safeguard the intellectual property of the embedded firmware code. This is done by securing the programmer with a special private password. This password will encrypt all the files that are sent or received from the main computer. It is important to choose the programmer lock carefully. If the password is lost, it will no longer be possible to write data to the FlashRunner Cube memory. The protected state is referred to as follows:

- Protection mode disable -> Level 00
- Protection mode enable -> Level 02

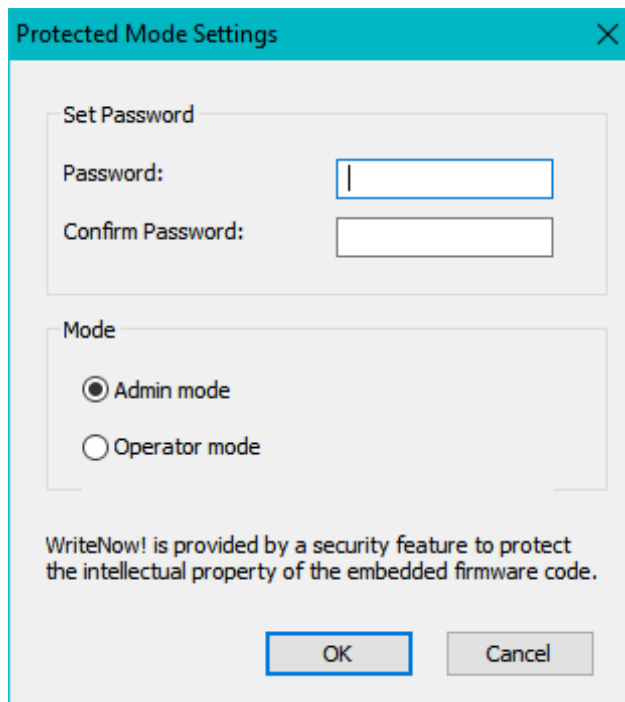
When the protection is active, two user modes are available:

- Operator mode
- Admin mode

In operator mode, the user can only transfer encrypted data to/from the programmer.

In admin mode, the user can transfer any file to/from the programmer. Only in admin mode is it allowed to change the protection level. The operator mode will be reactivated starting from the next reboot.

In order to get access the security settings, select "**Tools->Protected Mode**" and the following window will appear:





It is possible to reach the same result using FlashRunner Cube commands. Below are the commands used to activate the protected mode:

```
#sys -o set -p protection --password SMH --enable yes  
>
```

Now the current protection level is 02 and the user mode is operator. To enter in admin mode use the following command:

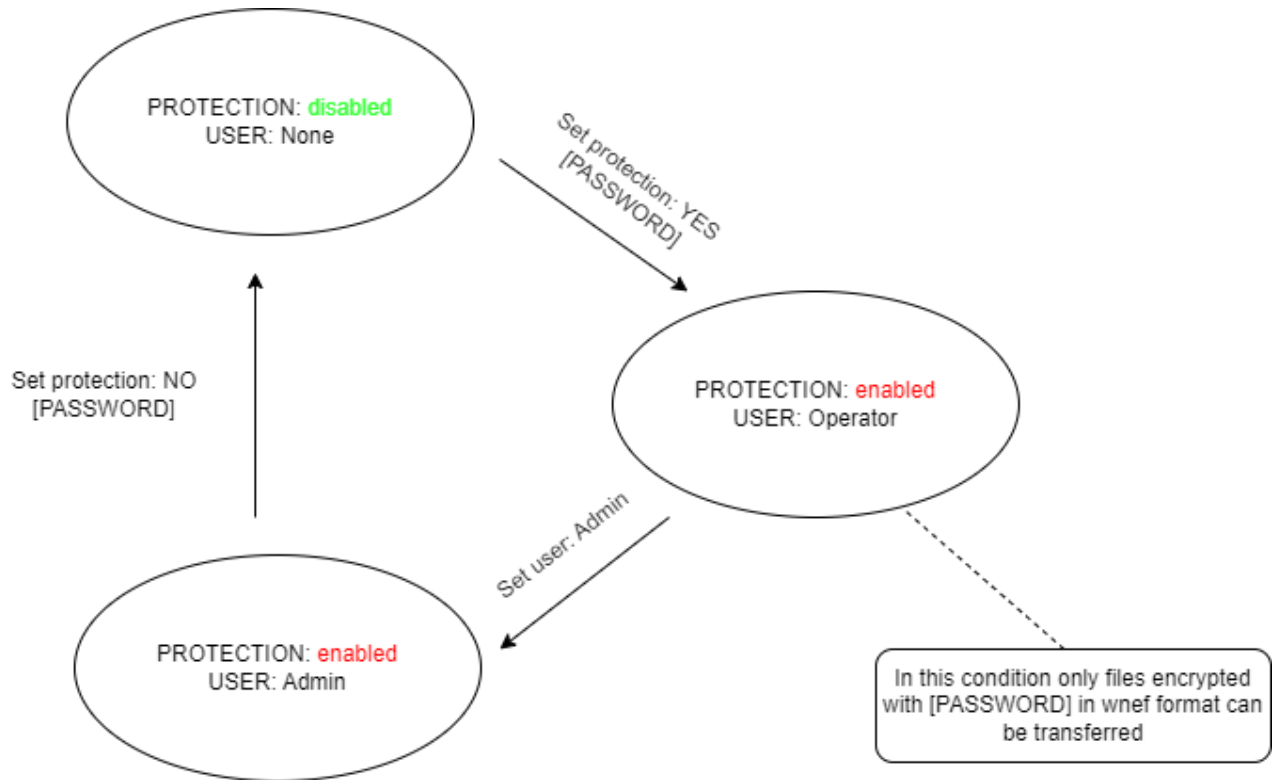
```
#sys -o set -p user --name admin --password SMH  
>
```

(Optional) Make sure that the password matched by checking the current user mode:

```
#sys -o get -p user Admin  
>
```

Now that you have the admin rights, send the following command if you want to disable the protected mode:

```
#sys -o set -p protection --password SMH --enable no
>
```



11.2 Data Encryption

Encrypting the files can be done through the **wn_sec.exe** command line tool, located in the **developer\C++** directory, as follows:

Encrypt your source file with the same password used for locking the FlashRunner Cube;

```
wn_sec.exe -o -enc -if ".\myprj.wnp" -of ".\myprj.wnp.wnef" -password SMH
```

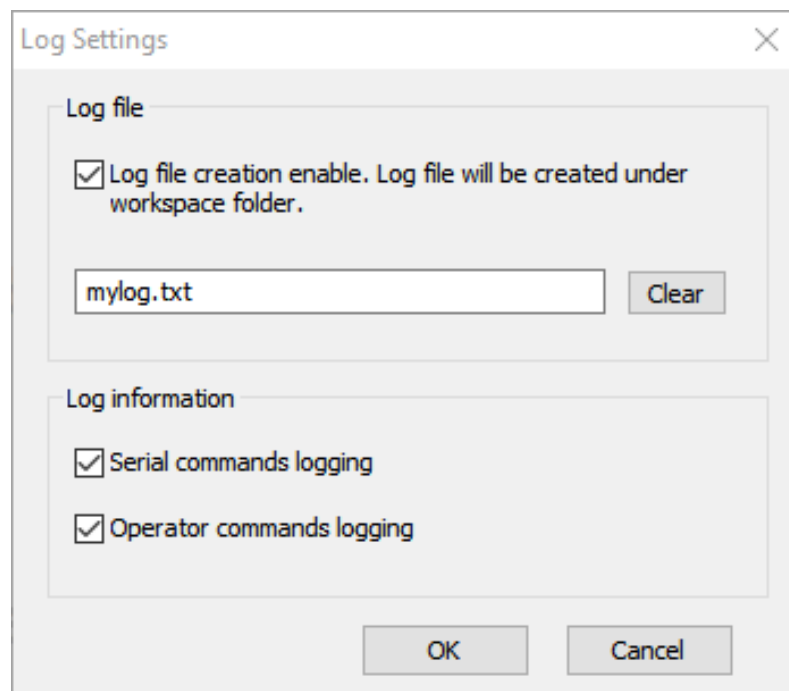
```
wn_sec.exe -o -enc -if ".\myimage.wni" -of ".\myimage.wni.wnef" -password SMH
```

12 Diagnostic Test

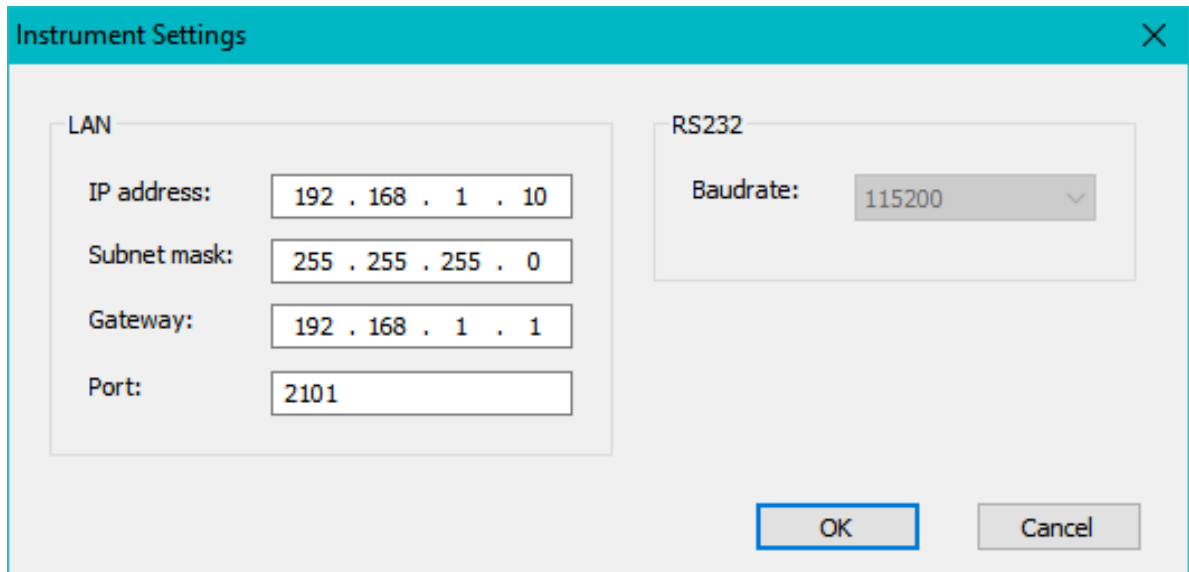
FlashRunner Cube has the possibility to self test its hardware structure. This means that user can verify if there is something wrong on the FlashRunner Cube hardware. In order to correctly perform the diagnostic test select “**Hardware Settings > Hardware Test**”.

13 FAQ and User’s cases

1. Has the FlashRunner Cube a log file to trace the operations performed?
Yes, starting from FR Cube Project Generator (v3.06), the user has the possibility to enable and save the log file. In order to enable the log file the user has to allow it clicking on “**Tools > Log Window > Show Log Window**”. FR Cube offers also the possibility to save the log file in .txt format clicking Log Settings and allowing the Log file creation.

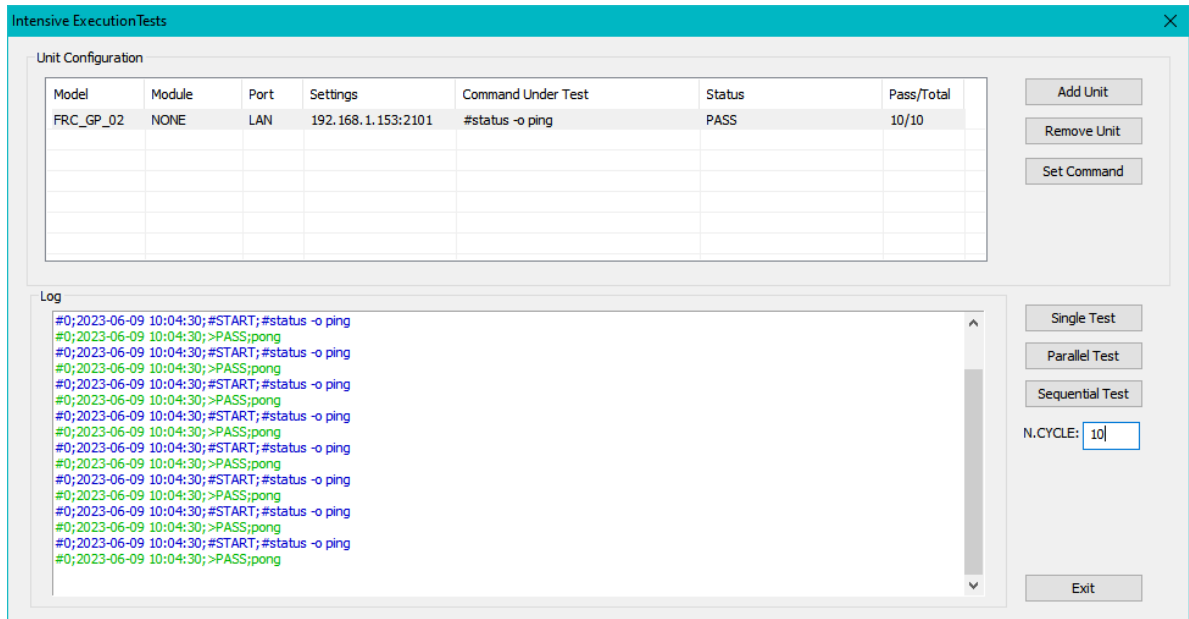


2. How is possible to change the IP address of the FlashRunner Cube?
Using the FR CUBE Project Generator it is possible to change the IP address of the FlashRunner Cube by clicking on “**HW Settings > LAN/RS232 Settings**”.



The screenshot shows the 'Instrument Settings' dialog box. It is divided into two main sections: LAN and RS232. The LAN section contains four input fields: IP address (192 . 168 . 1 . 10), Subnet mask (255 . 255 . 255 . 0), Gateway (192 . 168 . 1 . 1), and Port (2101). The RS232 section contains a Baudrate dropdown menu set to 115200. At the bottom right, there are 'OK' and 'Cancel' buttons.

3. Is possible to perform a stress test using the FlashRunner Cube?
Yes, it is possible to perform a programming stress test from the FR Cube Project Generator 3.05. Clicking on “**Tools > Intensive Execution Tests**”

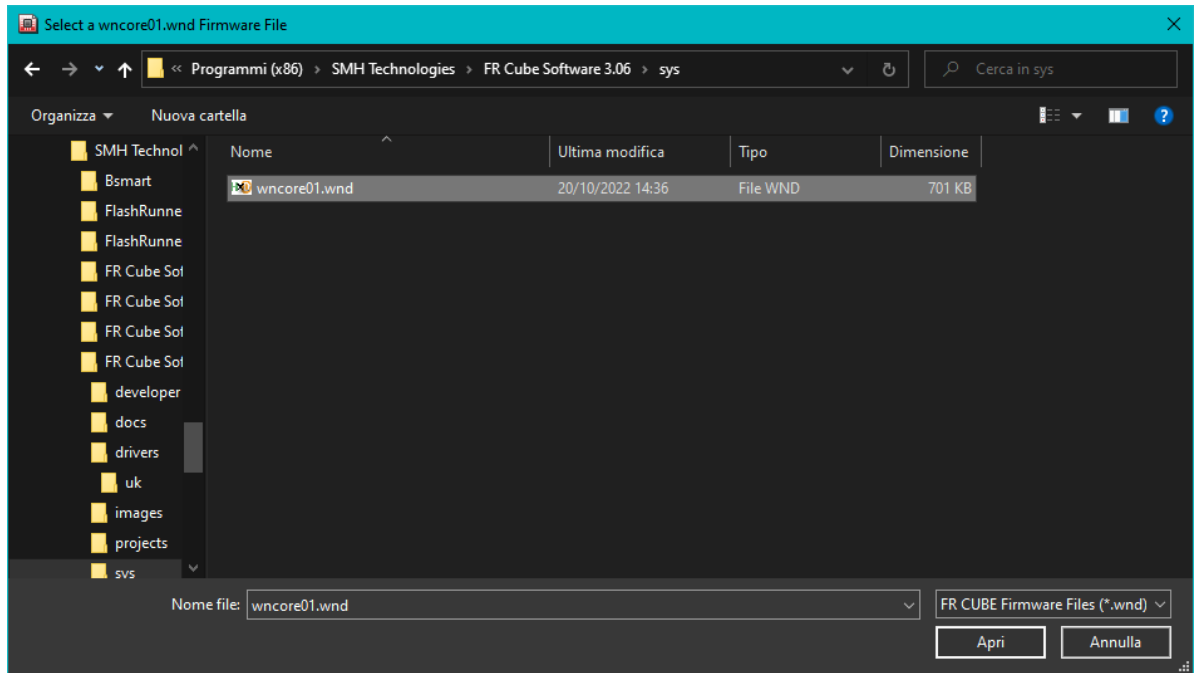


The screenshot shows the 'Intensive Execution Tests' dialog box. It features a 'Unit Configuration' table with the following data:

Model	Module	Port	Settings	Command Under Test	Status	Pass/Total
FRC_GP_02	NONE	LAN	192.168.1.153:2101	#status -o ping	PASS	10/10

Below the table is a 'Log' section displaying a series of test results, each starting with '#0;2023-06-09 10:04:30;#START;#status -o ping' and ending with '>PASS;pong'. To the right of the log are several control buttons: 'Add Unit', 'Remove Unit', 'Set Command', 'Single Test', 'Parallel Test', 'Sequential Test', 'N.CYCLE:' (set to 10), and 'Exit'.

4. How to update the WNcore of the FlashRunner Cube?
In order to start the updating process of the FlashRunner Cube the user has to click on “HW Settings > Firmware Upgrade” and then select the wncore01.wnd file.



14 Contacts

Our support team is at your disposal. If you have any troubles or questions please write an email at support@smh-tech.com