UG10029

DEMO-BYL1-EVB software user guide

Rev. 1 — 1 April 2022 User guide

Document information

Information	Content
Keywords	BYLMP GUI, FlexGUI
Abstract	This user guide introduces the NXP BYLMP GUI, a GUI based on the common software platform called FlexGUI.

Revision	Date	Document Changes
1	20220401	Initial release.

1 Introduction

This document introduces **NXP BYLMP GUI**. It is based on common software (SW) platform called **FlexGUI**.

The user should follow this workflow:

1. Set up hardware. Become familiar with target devices and kits.
2. Set up the software.
   a. Download the SW package from website or other source.
   b. Program the MCU with provided firmware, using the USB Multilink.
   c. Install Java dependency on your OS.
   d. Install the GUI or use portable version.
3. Learn how to work with launcher, workspace, script editor, and register map.
4. Learn how to use tabs and options specific for FS85[5] and PF502x[6], [7] devices.
5. Learn how to use tabs and options specific for BYLMP system solution.

Recommendation:

Refer to related hardware user guides and data sheets for details on target devices and evaluation kits. The target devices and evaluation kits are not within the scope of this document. Links to references are found in Section 8.

What is FlexGUI?

FlexGUI is a universal SW framework for development of GUI applications for remote evaluation and configuration of target embedded devices controlled by MCU using standard peripherals like SPI, I^2C, IO, ADC, TMR, and so forth.

This universal framework provides a common user interface for various types of analog devices (for example SBCs, PMICs, GDs, and XSs). It consists of workspace, script editor, register map, and optionally other generic tabs/components, which can be easily customized per device type.

Each device family is supported by custom extension of this framework which handles following specifics.

- Definition of communication between MCU and device.
  - Definition of data frame formats for supported buses.
- Definition of device models and their revisions.
  - Definition of control arguments (adjusting differences in the control logic per device models).
  - Definition of automatically generated scripts into script editor.
  - Definition of register map (register sets, register groups, registers).
  - Definition of required MCU resources (pins, buses, converters, timers, and so forth).
  - Definition of bus routing rules for available register sets or registers.
  - Definition of feature tabs (visualization of device features in terms of configuration and its feedback).
  - Definition of feature sets (initialization behavior, loaded tabs).
- Definition of related kits (evaluation boards).
  - List of supported MCUs.
  - List of supported devices (with constraints on allowed models and revisions).
  - Configuration and wiring (pin mux) of used MCU resources for enumerated devices.

The firmware provided runs on the target MCU, then executes commands from the GUI and translates them into actions on MCU peripherals such as SPI, I^2C, IO, ADC, and TMR. After completion, it sends a response with the result and optionally additional data back to the GUI.

2 Hardware preparation

This hardware preparation section briefly describes how to set up the supported evaluation kits for use with the software GUI.

Note: *This section is not meant to be an exhaustive description for the target devices or the evaluation kits used. For a deep dive into the technical details, refer to related user guides, data sheets, schematics, and other device-related deliverables.*

2.1 XVALEYEQ4ESEVB / KL25Z

The XVALEYEQ4ESEVB / KL25Z kit serves as an NXP reference design for a system solution demonstrating a combination of power management products. It uses an on board MCU with USB interface for the communication with the PC / GUI SW.

2.1.1 System block diagram

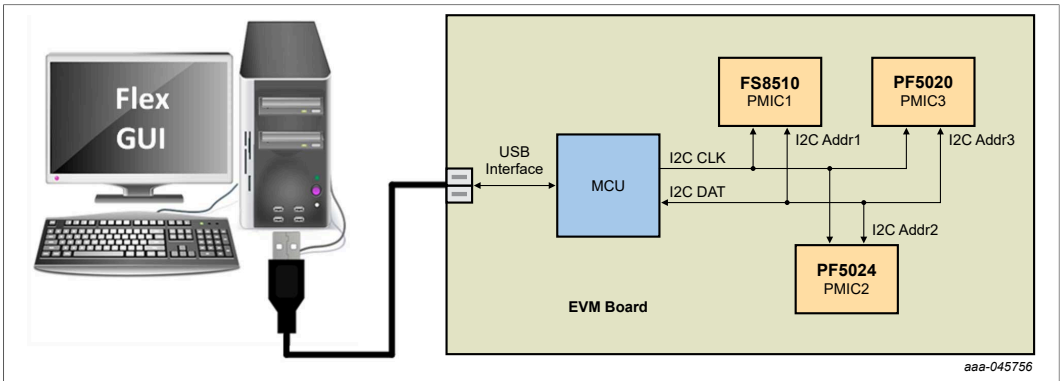


Figure 1. System block diagram

2.1.2 Supported MCUs and devices

Table 1 provides a list of supported MCUs and their interfaces.

Table 1. Supported MCUs

Target MCU	External MCU	Connection	Programming	Notes
KL25Z128VLK4	No	USB HID	USB Multilink	All soldered.

The supported devices are:

- FS8510
- PF5020
- PF5024

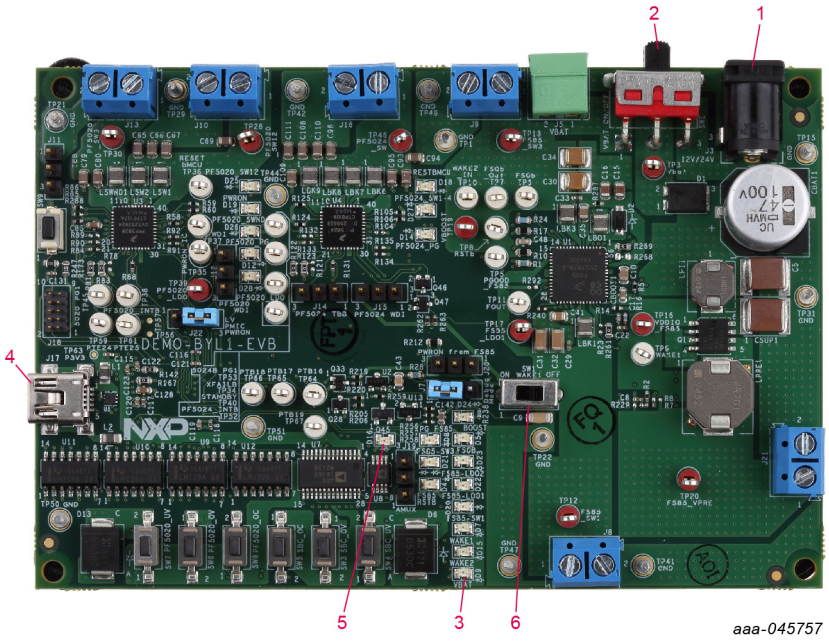


Figure 2. XVALEYEQ4ESEVB

1. Power supply connector
2. Power switch
3. VBAT LED
4. USB connector
5. USB LED
6. SW1 (WAKE1) switch

2.1.3 Connecting to the kit

This section describes required hardware settings of the kit in order to realize connection with the GUI.

1. Connect the **USB cable** to the PC and the **USB port [4]** on the kit.
2. The **LED [5]** should be ON which signalizes USB connected.
3. Provide external VIN 12 V on available **power supply connectors [1]**.
4. Turn on the kit using **switch [2]** into right position depending on used power supply connector.
5. The **VBAT LED [3]** should be ON which signalizes connected power supply.
6. Turn on **SW1 [6]** to power up SBC and optionally PMICs (depends on J20 and J22 positions).

3 Software preparation

The software preparation section describes how to download the SW package, install Java on your computer. and run the application.

3.1 Downloading the SW package

To download the latest version of the GUI SW:

1. Visit the NXP website for BYLMP EVB[1] / Software Downloads and look for the GUI SW.
2. Download the SW package as a (.zip) file.
3. Unzip the downloaded file and check to see that the folder contains the files listed in Table 2.

Table 2. Content of the SW package

Folder Name	Folder Contents
GUI	This directory contains GUI application.
NXP_BYLMP_GUI-[version].exe	Executable installer of GUI application for Windows.
MCU	This directory contains firmware for KL25Z MCU.
flexgui-fw-kl25z-usb-hid-BYLMP-[version].bin/.gdbi/.hex/.map/.s19	Firmware binaries (.bin, .s19, and so forth).
flexgui-fw-kl25z-usb-hid-BYLMP-[version]_USBMULTILINK.bat	GDB programming script for USB Multilink Universal.
LICENSE_FW.txt	License file for MCU firmware.
UG	This directory contains user guide for GUI application.
ChangeLog.txt	Diff list for provided releases.
README.txt	Package content and brief instructions.

3.2 Programming the KL25Z onboard MCU

This section describes how to program provided firmware on MCU with use of USB Multilink.

Note: *Releases and versions of target tools referenced in this section may become outdated in near future. Therefore NXP does not guarantee that the combination of updated releases and versions of target tools work together out of the box. Direct distribution of these tools with our GUI solution is not possible because of licensing restrictions.*

3.2.1 Creating root folder for GDB tools

In order to program the MCU via Multilink, collect, and prepare several tools.

Create common folder called **'Multilink_GDB'**, where these tools are gathered. There are no constraints on the location of this folder, the folder may be placed in a user directory if desired or other locations of choice.

3.2.2 Installing USB multilink system drivers

Download the latest drivers from the PEmicro's USB Multilink Universal webpage.

Figure 3. Downloading PEmicro USB multi-link

Follow these steps to install the drivers:

1. Run **'multilink_universal_install.exe'**, click through the setup and wait for completion.
2. Connect BYLMP EVB[1] micro-USB port to USB port on your PC in order to provide a power source for the MCU.
3. Connect USB Multilink to BYLMP EVB[1] via **[PORT G – Mini-10 (Kinetis)].**
4. Connect USB Multilink to USB port on your PC and wait for OS to install it.
5. Open **Device Manager** and determine whether the program successfully installed as expected as shown in Figure 4.

Figure 4. Driver installation completed.

After successfully installing the program, proceed with the preparation of required tools.

3.2.3 Getting GNU Arm GDB

Download latest package for Win32 (or any other arch) from the GNU Arm Embedded Toolchain webpage[4].

Follow these steps to install GNU Arm GDB.

1. Run **'gcc-arm-none-eabi-9-2020-q2-update-win32.exe'**, click through the setup and wait for completion.
2. Locate the file named **'arm-none-eabi-gdb.exe'** by navigating to **<path-to-installation-folder>/GNU Arm Embedded Toolchain/9 2020-q2-update/bin** folder as shown in Figure 5.

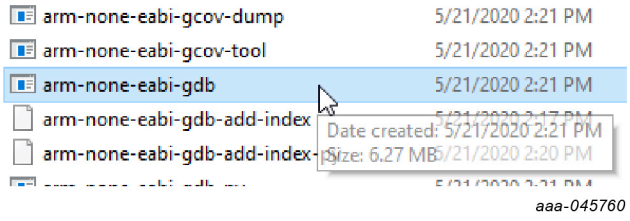


Figure 5. Locating the file arm-none-eabi-gdb.exe

3. Copy the **'arm-none-eabi-gdb.exe'** file to your 'Multilink_GDB' root folder and subfolder called **'client'**.

3.2.4 Getting PEmicro's GDB Server for Arm

Download latest package from the PEmicro's GDB Server Plug-In for Eclipse-based ARM IDEs webpage[3].

A login prompt or a prompt to a direct download link via email appears. Choose either.

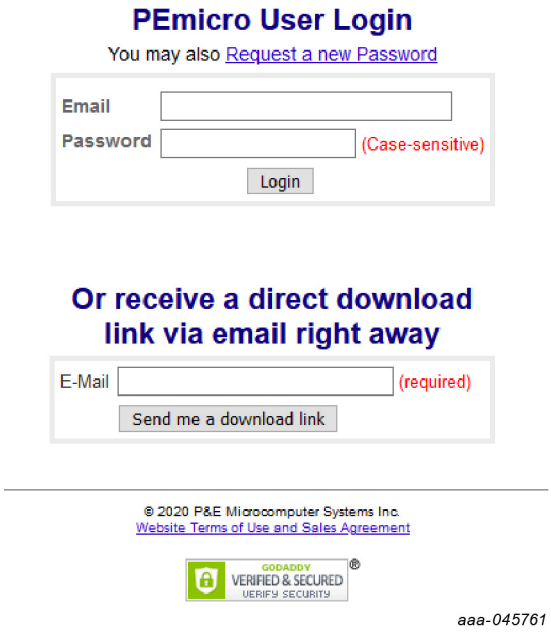


Figure 6. PEmicro user login

Follow these steps to obtain PEMicro's GDB Server[3] from the downloaded archive.

1. Unpack the downloaded zip archive called **'com.pemicro.debug.gdbjtag.pne.updatesite-4.5.8-SNAPSHOT.zip'**.
2. Go to the plugins subdirectory of unpacked file.
3. Extract content of the **'com.pemicro.debug.gdbjtag.pne.expansion_4.5.8.202003301732.jar'** file using a compression tool of your choice.
4. Navigate to the win32 subdirectory of the extracted file to view the directory structure shown in Figure 7.

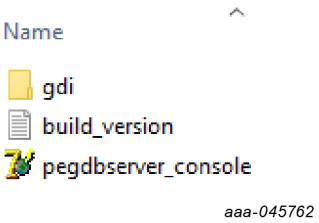


Figure 7. Win32 extracted file subdirectory

5. Create folder named **'server_win32'** in your 'Multilink_GDB' root folder.
6. Copy and paste the files files shown in Figure 7 to the **'server_win32'** in your 'Multilink_GDB' root folder.

3.2.5 Programming procedure

Follow these steps to program the MCU with provided firmware.

1. Copy content of the MCU folder (see Table 2) to your 'Multilink_GDB' root folder. The copied content of the MCU folder should have similar directory structure as shown in Figure 8, without the docs and drivers folders.

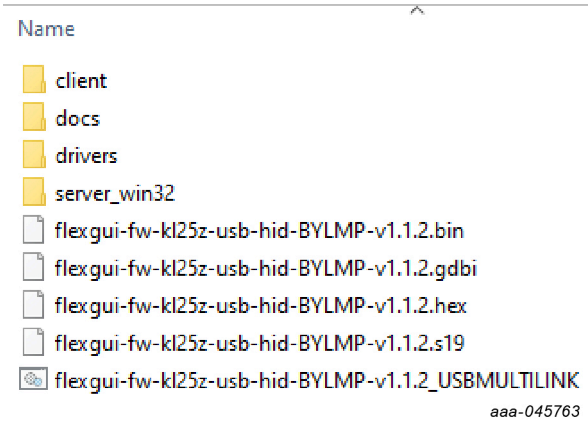


Figure 8. Multilink_GDB directory structure

2. Run the script file called **'flexgui-fw-kl25z-usb-hid-BYLMP-v1.1.2_USBMULTILINK.bat'** and wait for completion.
3. A console output displays as shown in Figure 9 indicating 'Programmed' and 'Checksum Verification Successful'.

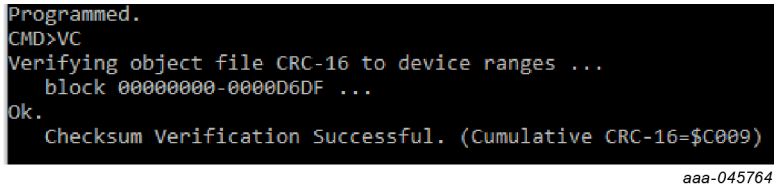


Figure 9. Console output

4. The console output verifies that the programming of the MCU was successful.

3.3 Running the GUI (Windows OS Only)

Follow these steps to install the GUI on your OS.

1. Double-click the installer located in the GUI folder, see Figure 10.

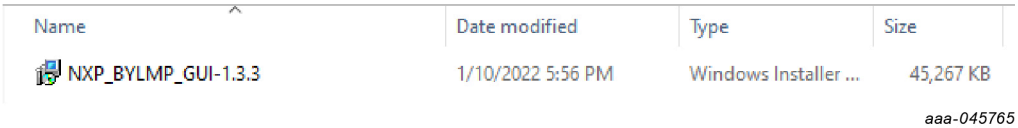


Figure 10. GUI installer

2. Accept license agreement (Figure 11) and click the Next button.

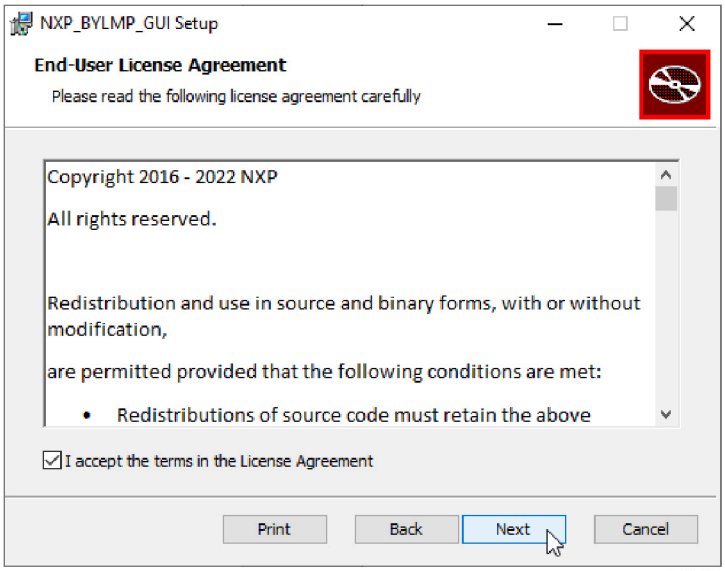


Figure 11. End-User license agreement

3. Select a destination folder (Figure 12) and click the Next button.

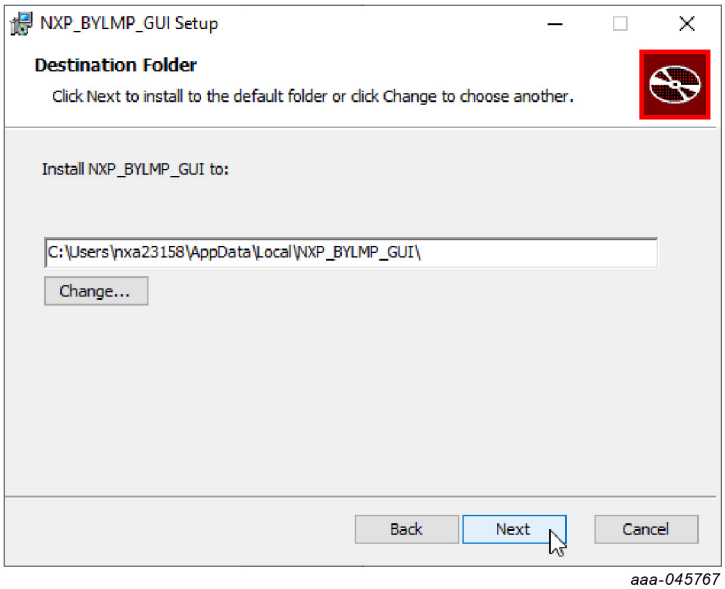


Figure 12. Destination folder selection

4. Confirm the installation (Figure 13). Wait for the installation to complete.

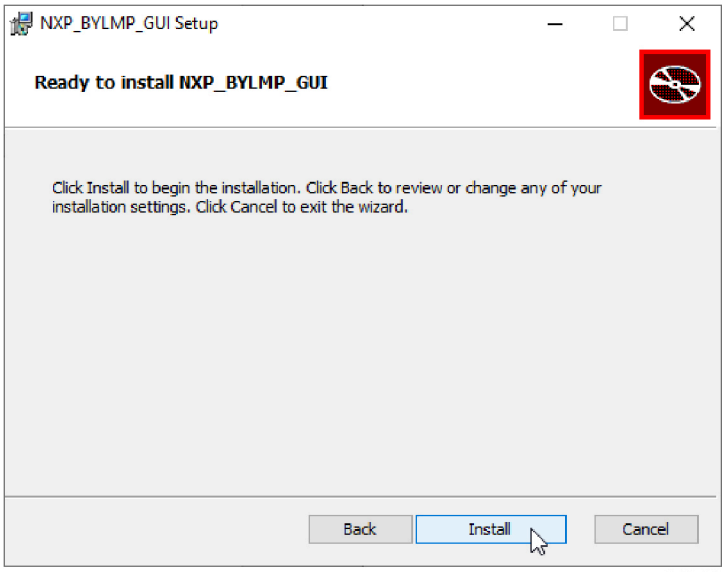


Figure 13. Confirm installation

5. Launch the GUI via the desktop icon or start menu shortcut called **NXP_BYLMP_GUI** and wait for the GUI launcher to appear. The GUI displays a splash screen shown in Figure 14.

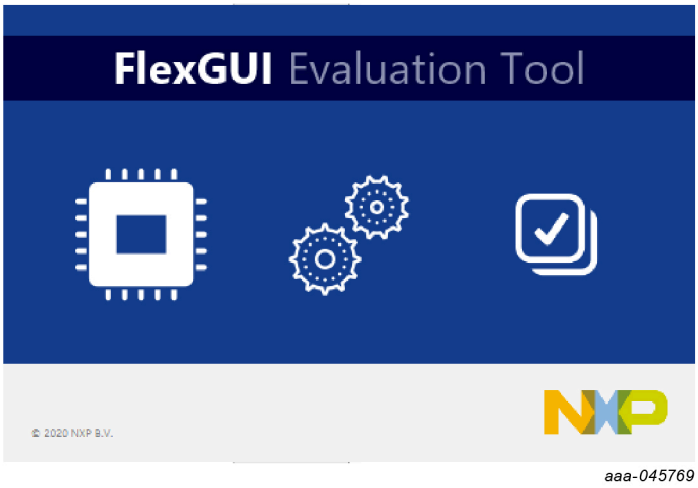


Figure 14. GUI splash screen

4 Using the software

This section describes how to work with the GUI application, namely how to select a kit for use and how to navigate through the application environment. It uses the FS8510 as the reference and the steps are either similar or the same for the PF5020 and PF5024.

4.1 Application launcher

The application provides a dedicated kit selection window that is launched after start, see Figure 15.

After the splash screen presents and is dismissed, the application launcher provides a dedicated kit selection window, a window launched after start, see Figure 15.

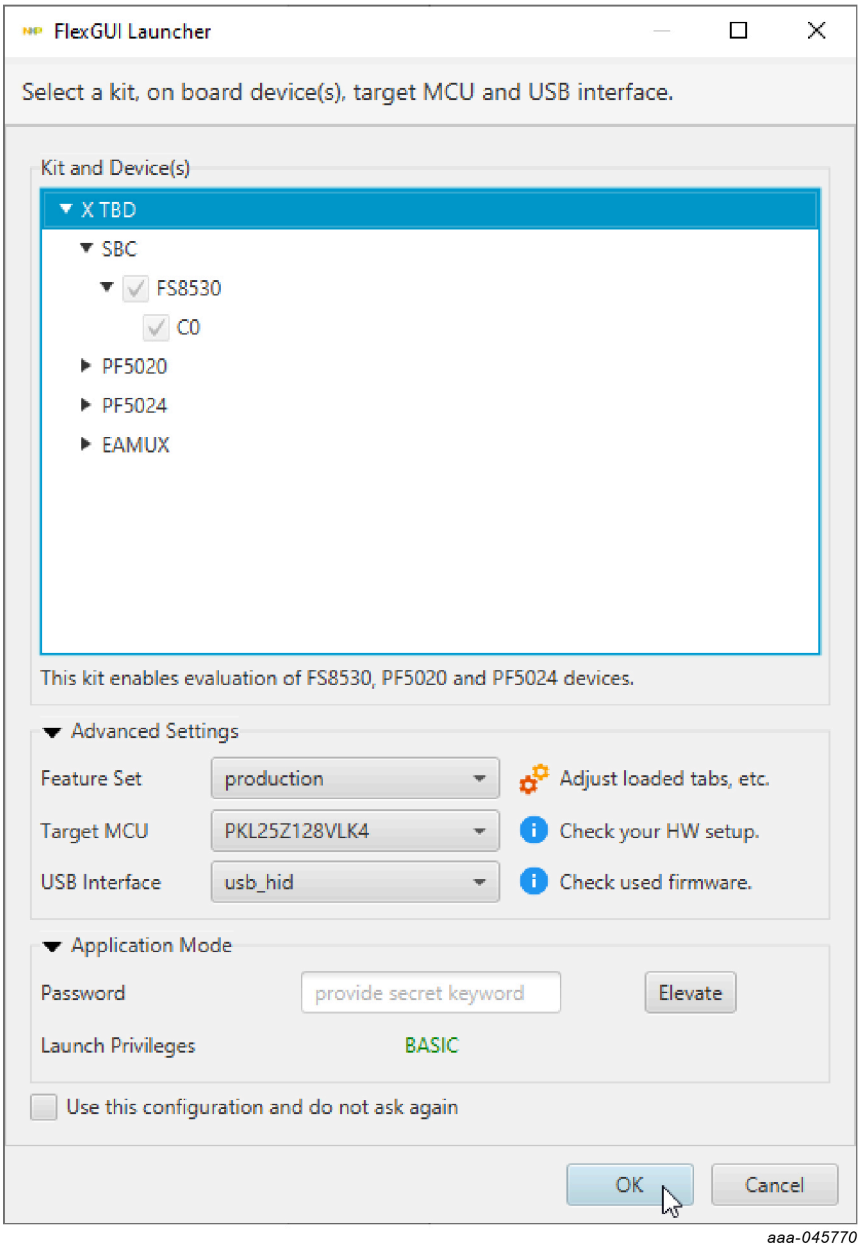


Figure 15. Application launcher

After making selections, selecting the OK button downloads the selected application workspace. For details on each section of the application launcher window, see Section 4.1.1 "Kit selection", Section 4.1.2 "Advanced settings", Section 4.1.3 "Application mode", and Section 4.1.4 "Default configuration".

4.1.1 Kit selection

It encompasses an area for selecting a kit, a device/s present on the kit and their models and revisions. The kit and device selection area of the application launcher allow users to view and select a kit, the devices included in the kit and view their models and revisions.

Depending on the the kit/device selection, the values in the advanced settings section change to support the kit or device selected, see Section 4.1.2.

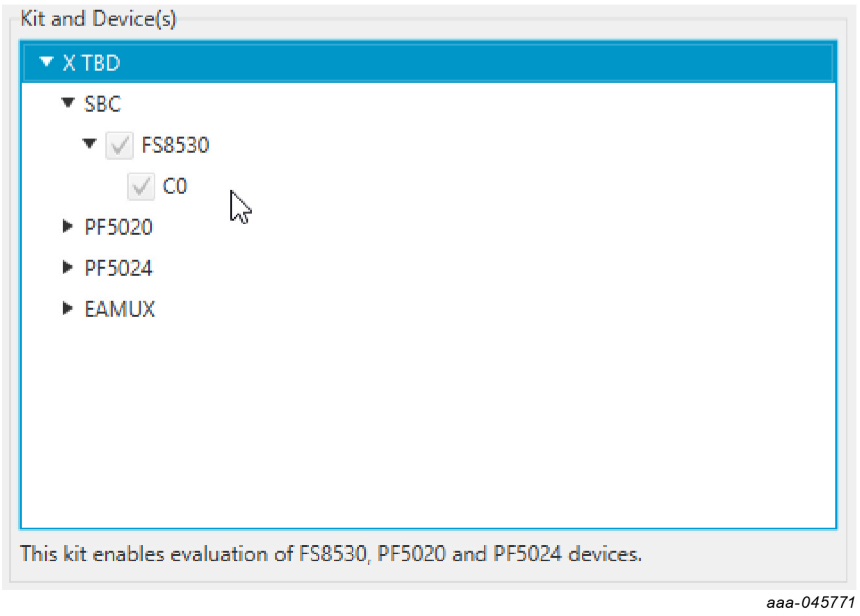


Figure 16. Kit and device selection

CAUTION: *It is important for user to match this selection of device models and revisions with their physical counterparts, otherwise GUI might display less or more features and also user might experience unexpected behavior.*

There are usually two types of kits.

- **Soldered** – Without professional equipment, chips soldered to boards cannot be changed out by the user. For soldered kits, the GUI disables any device model and revision selection for the kit in launcher.
- **Socketed** – With socketed kits, the user can easily swap different models and revisions of chips. For socket kits, the GUI enables the device model and revision selection for the kit in launcher. For the DEMO-BYL1-EVB kit, socketed kits are not proposed, and only soldered kits are available.

4.1.2 Advanced settings

This configuration is for adjusting advanced options. The default values are suitable for the most use cases. See Figure 17.

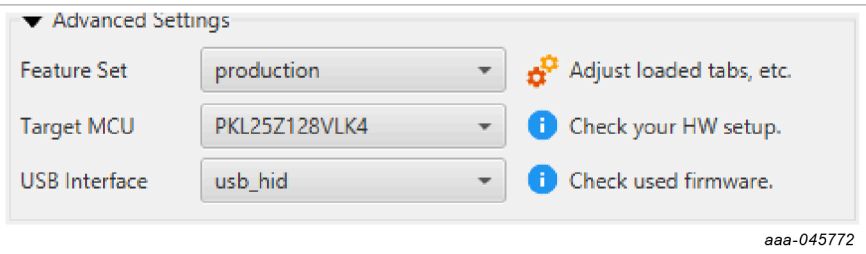


Figure 17. Advanced settings.

The feature set represents loaded tabs, default device modes (including polling), and other arguments for default GUI behavior.

Target MCU is self-explanatory, the kit might have an embedded MCU or it can use external shield boards.

CAUTION: *The target MCU must match the physical MCU counterpart, otherwise the GUI is unable to attach MCU resources.*

4.1.3 Application mode

This configuration is for differentiation between basic and advanced (internal) version of the GUI. The purpose of basic version is to hide configuration options and features which are not available to the mass market.

The application mode section of the application launcher supports two modes, the basic (Figure 18) and advanced (internal) (Figure 19) versions of the GUI. Basic is the default setting. The basic version hides configuration features and options that are not available to the mass market.

To access the hidden features and options of Advanced mode, click the Elevate button and enter the correct password.

To return to Basic mode after using Advanced mode, click the Drop button to return to basic launch privileges.

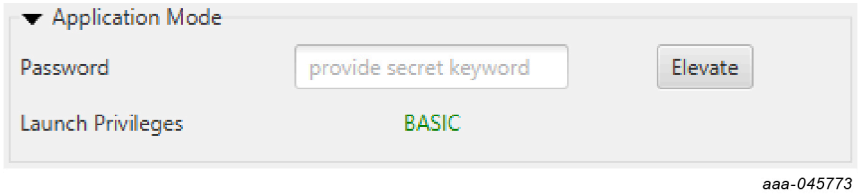


Figure 18. Basic mode

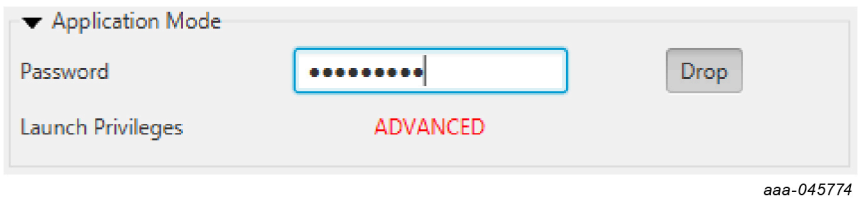


Figure 19. Accessing advanced mode

4.1.4 Default configuration

The default configuration option (Figure 20) on the application launcher allows the application to start without the application launcher. Checking "Use this configuration and do not ask again" stores the selection and opens the application with the selected default settings. User may disable this feature in the global settings, see sections Section 4.3.1 and Section 4.4.1.

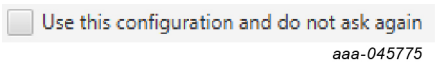


Figure 20. Default configuration option

4.2 Online and Offline configuration

The application allows users to work in either an online or offline operational state.

- **Online** – The kit is connected and all user configuration choices are immediately applied to the physical devices on the kit.
- **Offline** – The kit is not connected and all user configuration choices are stored locally. The offline configuration can be saved for use later.

Important: *This duality allows the GUI to serve as an alternative to OTP (one-time programming) configuration tools. These tools are usually in the form of a spreadsheet or dedicated applications that lack the ability to communicate with the kit.*

4.2.1 Configuration export and import

The current configuration can be exported into the script editor, see Figure 21. See sections Section 5.3 and Section 6.3 for details on available scripts.

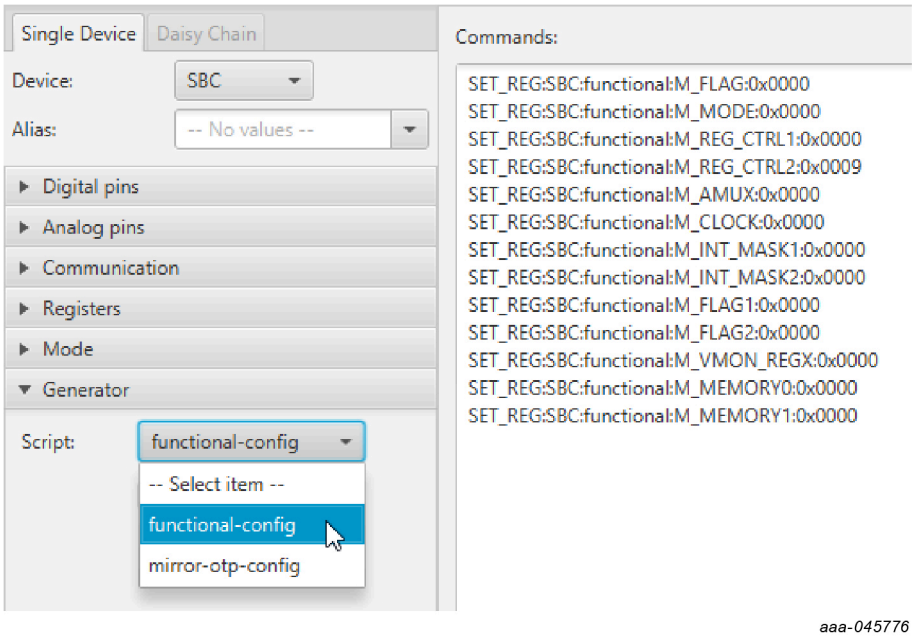


Figure 21. Configuration export

When the 'Append' option is selected (see Figure 22), the generated script is appended to the existing content in commands window.

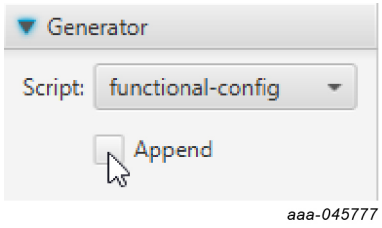


Figure 22. Append option on the generator panel

The generated script can be modified, exported/imported directly to and from the script editor. See Section 4.5 "Script editor tab" for details.

4.2.2 Configuration synchronization with the MCU

Because the user can modify configuration settings in the offline mode, addressing how to handle the offline modified settings when the GUI connects to the MCU, needs to be addressed.

There are thee options that the GUI has to address these changes.

- Attempt to synchronize configuration choices with the physical device.
- Wipe out configuration choices and restore defaults.
- Do nothing. **Enabled by default.**

The default behavior can be altered on demand for custom application builds and selected products.

4.3 Workspace layout

The GUI uses a standard application layout divided into several working areas, see Figure 23.

- Menu (see Section 4.3.1) and Toolbar (see Section 4.3.2)
- Request Log (see Section 4.3.3)
- Device Control Panel (see Section 4.3.4)
- Device Tabs (see TBD) and Script Editor (see Section 4.5)
- Configuration and evaluation tabs (see Section 4.3.5)
- Status bar (see Section 4.3.6)

The areas are described in following subsections.

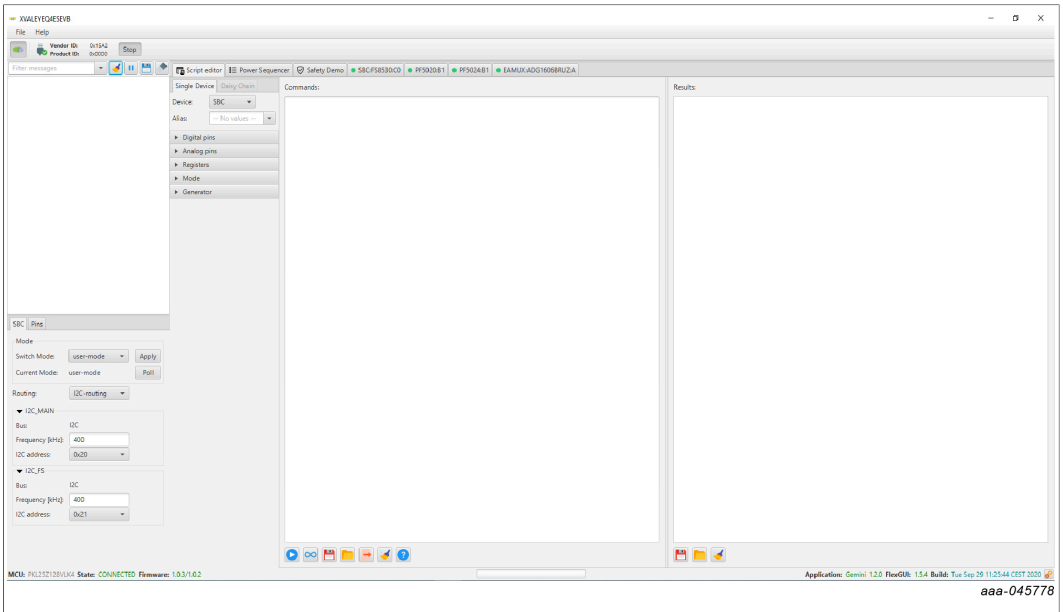


Figure 23. Application workspace

4.3.1 Menu

The application menu (Figure 24) provides access to additional windows/dialogs.

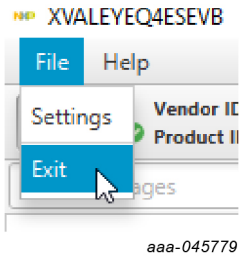


Figure 24. Application menu

The “File” menu currently has only two options. “Settings”, which enables users to customize the GUI behavior and “Exit”.

4.3.2 Main toolbar

The main application toolbar provides access to communication settings. It enables users to control a USB Human Interface Device (HID) connection.

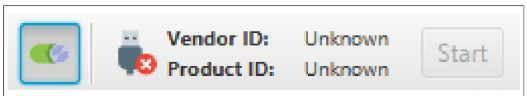


Figure 25. No MCU detected

Figure 26. MCU detected with valid vendor and product ids

Once the GUI detects an MCU with a valid **Vendor ID** and **Product ID**, the user may press the Start button to begin the connection. See Figure 26.

Note: *When selected, the button to the left of "Vendor ID:" and Product ID:" hides or displays the toolbar panel in order to save space.*

4.3.3 Request log

The request log section of the application (Figure 27) documents processed application requests.

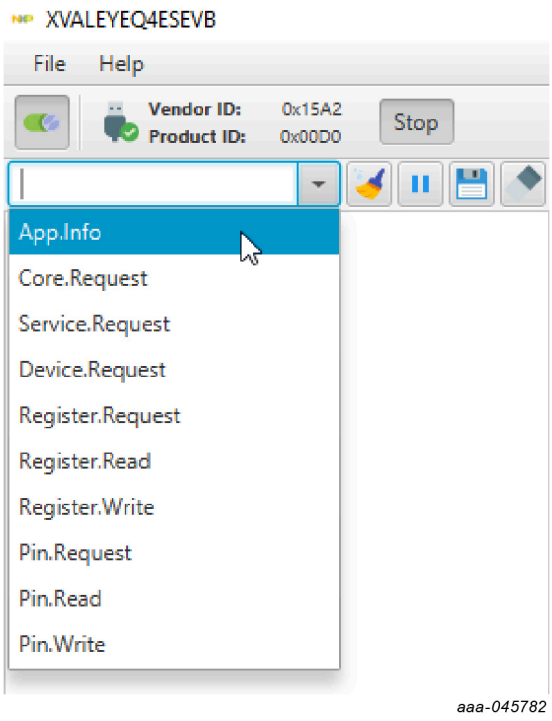


Figure 27. Request log section

There are dedicated types of read/write requests for analog/digital pins and device registers.

- For register read and write requests, the message format is:
  Request Number > Device Name [Register Name:Register Address] [R/W]: Value
- For analog/digital pin read and write requests, the message format is:
  Request Number > Device Name [Pin Name] R/W: Value
- For register requests, users may change log level to FINE to view sent and received data frames. See Section 4.4.2 for details on log levels. The requester request format is:
  Request Number > in/out [0x Hex Value]|[0b Binary Value]

Additionally the following controls are available to modify the behavior of notification area.

- **Clean Message Log** – cleans all logs in the window.
- **Pause/Resume Log Processing** – pauses/resumes adding of new log messages into the area
- **Save Log** – saves logged messages into file.
- **Fast Logs Filtering** – enables user to search for logs based on pre-defined categories or user's input.

In Figure 28, the left image shows only read and write requests while the right image provides details about data frames sent and received. To view more detail, open Settings/ Logs and set the Log Level = FINE).

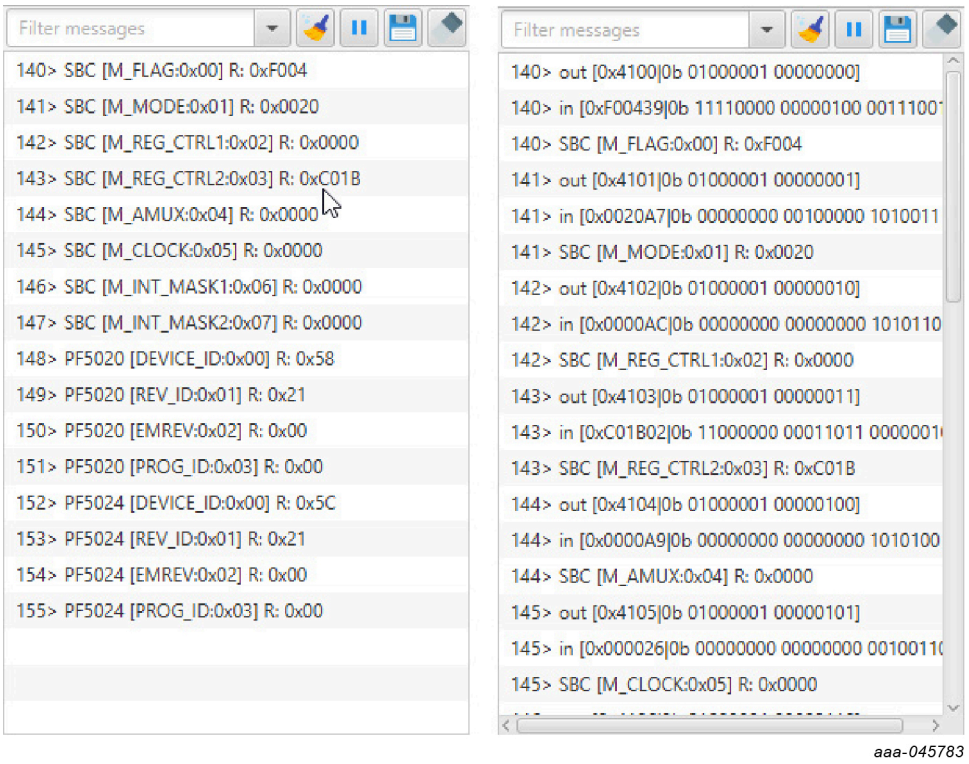


Figure 28. Request log

4.3.4 Device control panel

Located below the request log area of the application, the device control panel (Figure 29) enables users to work with device modes, change routing, and communication properties as well as monitor the input and output pins.

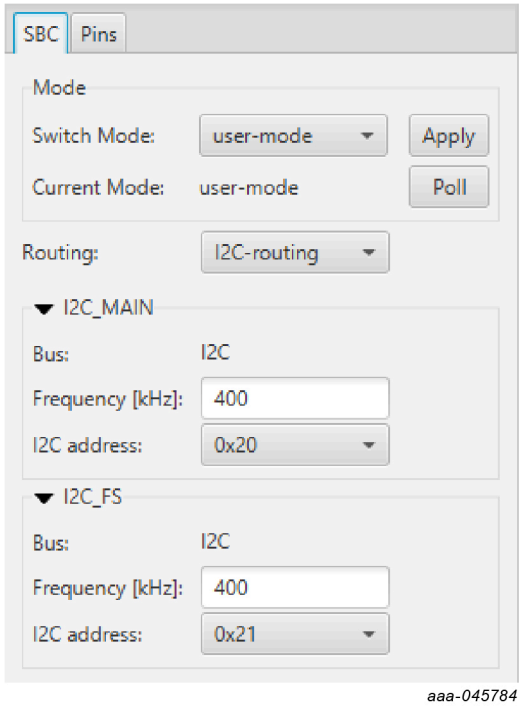


Figure 29. Device control panel

If a kit contains two or more devices, the device control panel contains configuration for last selected device tab (Figure 30).

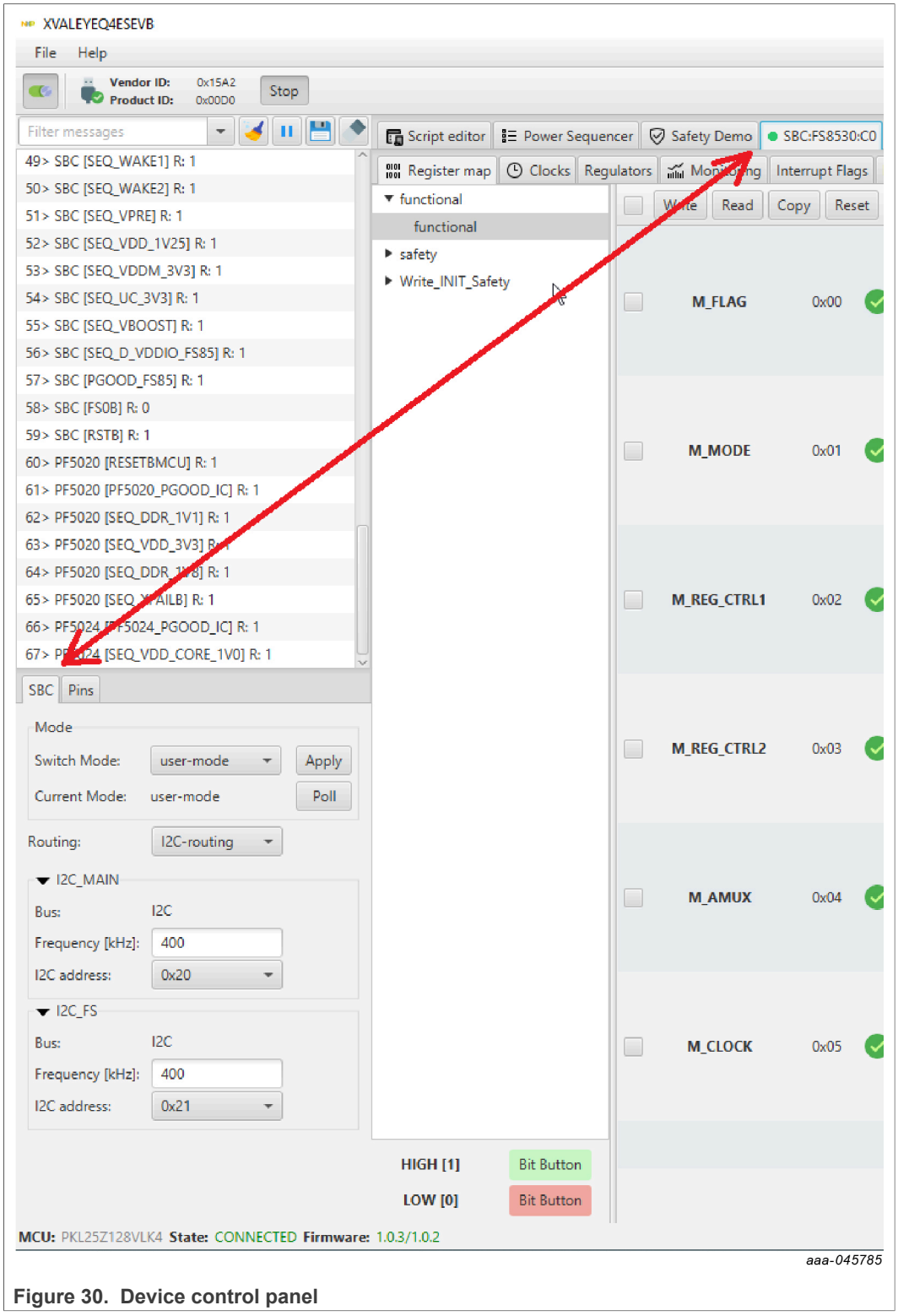


Figure 30. Device control panel

Switch mode (Figure 31) can be used to select the next device mode and must be confirmed by selecting the Apply button. Current mode indicates the current mode in effect. The Poll button is a toggle that enables the validation of the mechanism. It periodically checks whether the device continues to fulfill the conditions for the selected

mode. If not, the GUI informs the user and disables validation. If validation is enabled, the user can be sure that the current mode is still active.

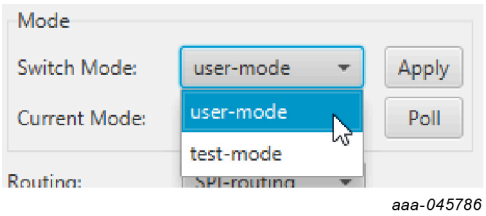


Figure 31. Switch mode

Device modes are used as wrappers for actions on pins and registers required to get given device into desired state. The GUI is tightly coupled with these and properly reacts on device mode change events, for example, some configuration items might be available only for specific device modes.

Next important setting is routing. It basically instructs the GUI, how to realize communication between MCU and device. Currently there is supported only SPI and I^2C, which can be further configured.

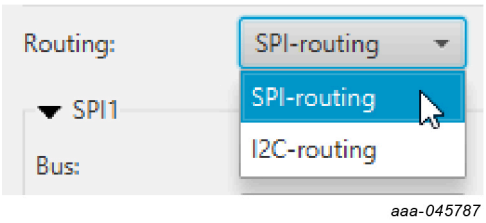


Figure 32. Routing

In case of SPI, only the frequency can be changed. See

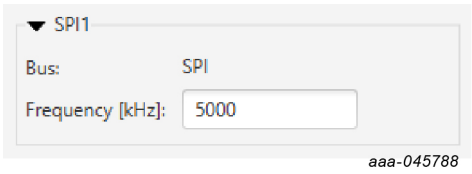


Figure 33. SPI bus options

In case of I^2C, selecting the device address is also possible.

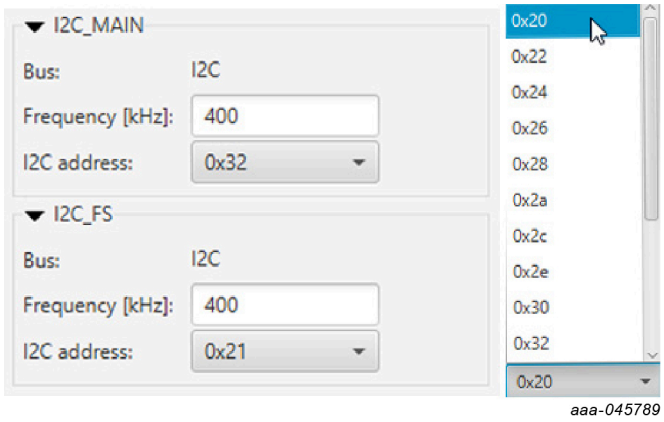


Figure 34. I²C device address selection

On the Pins tab, the other device control panel tab shown in Figure 29, users can work directly with the input and output pins of the device.

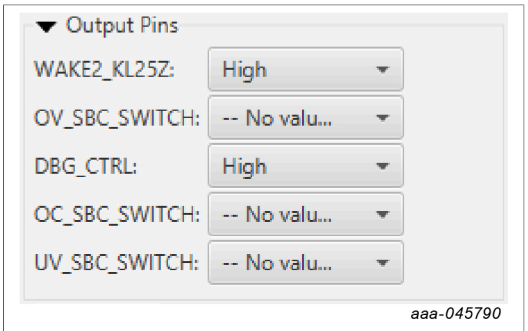


Figure 35. Output pins

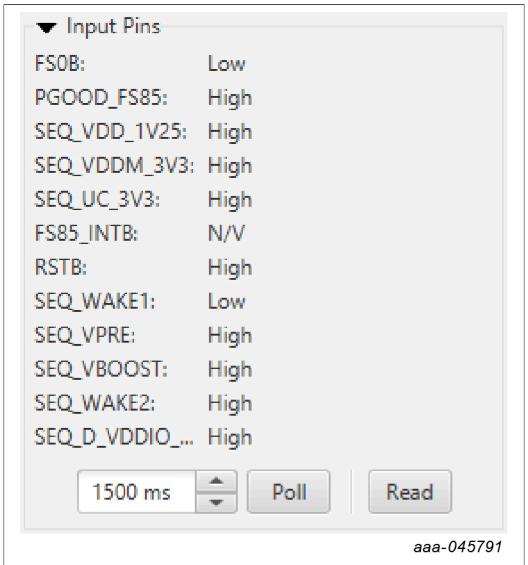


Figure 36. Input pins

For input pins, the user can enable polling for cyclic reading of the pin value with a given time period.

4.3.5 Configuration and Evaluation tabs

The configuration and evaluation tabs area of the interface enables user interaction with with all loaded devices. Each device has its own dedicated tab with several subtabs.

- **Kit Tab** - This type of tab interfaces multiple devices at the same time, it is suitable for system solution configuration an evaluation via additional subtabs.
  - Script editor, Power Sequencer, and Safety Demo
- **Device Tab** – This type of tab interfaces only to a single device at the same time, it is suitable for a specific chip configuration and evaluation via additional subtabs.
  - SBC, PF5020, PF5024, and EAMUX
  - Subtabs contains the following tabs.
    - **Register map** – enables the user to work with device on low-level abstraction of its features/functionality, see Section 4.6 for details on this common tab.
    - **Feature tabs** – tabs/components enable the user to work with device on high-level abstraction of its features/functionality, these are specific for each loaded device.

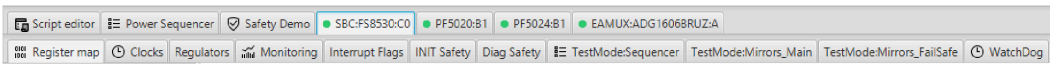


Figure 37. Configuration and evaluation tabs

Each device tab has its own indicator bullet to inform the user of the state of the device. The states are:

- OFFLINE – The device is offline and is not accessible as indicated by a red bullet.
- BOOTING – The device is booting and is accessible as indicated by a red bullet.
- ONLINE – The device is online, accessible, and responsive as indicated by a green bullet.
- BUSY – The device is busy and processing some requests as indicated by a yellow bullet.
- ERROR – The device is in an error state. The device is not behaving as expected or communication with the device is faulty, indicated by a red bullet.

4.3.6 Status bar

The status bar, located at the bottom of Figure 23, provides a standard overview of application's conditions. The status bar has three areas, left middle, and right.

On the left side of the status bar, the target MCU of the kit is shown along with the state of the connection and the firmware version (loaded on connect event). See Figure 38.

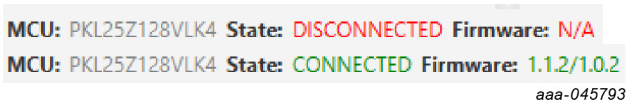


Figure 38. Target MCU status

In the middle of the status bar, the progress bar informs the user about the relative number of processed requests. See Figure 39.

Figure 39. Progress bar

On the right, the status bar displays the application version, FlexGUI platform version and build time. The Icon following the Build status displays a dialog with list of platform modules and versions.

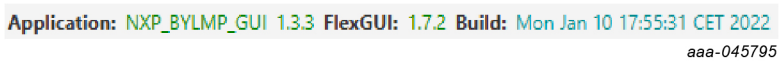


Figure 40. Application status

4.4 Workspace settings

The GUI enables users to customize the GUI behavior for individual preferences. Several areas of the GUI can be configured as described in these sections.

4.4.1 Kit/Device loader

When checked, the kit/device loader tab option "Show Loader" (Figure 41) opens the loader at the next application startup. Unchecked, the loader will not automatically open.

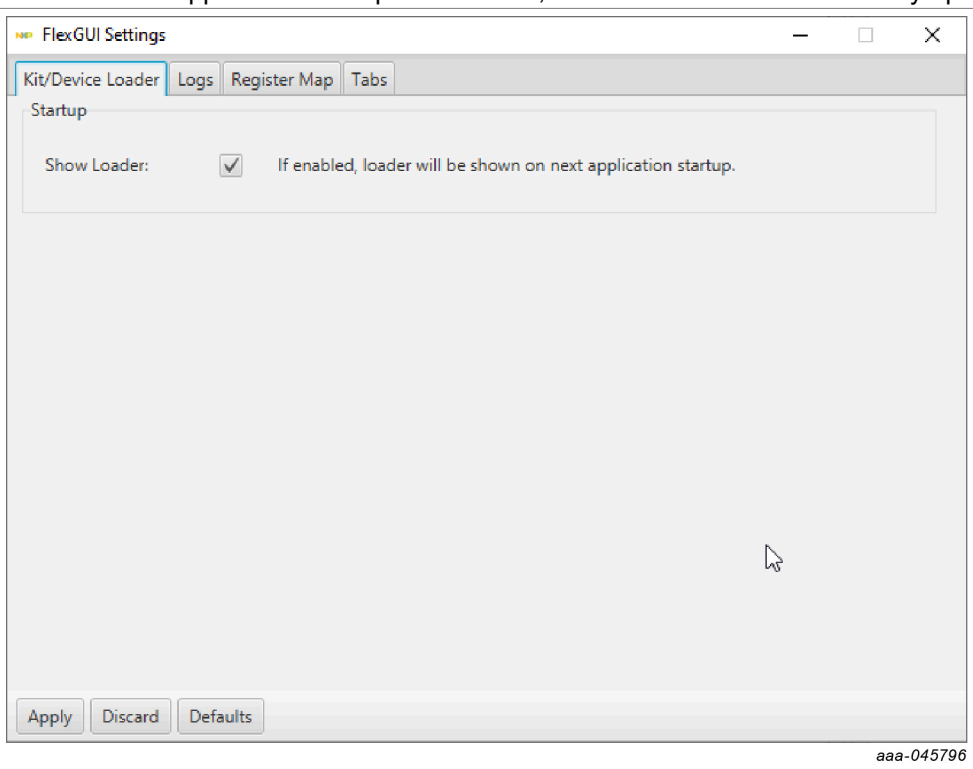


Figure 41. Kit/Device show loader option

4.4.2 Logs

Figure 42 presents the "Logs" tab of the workspace settings. The log tab contains configuration choices related to request logs. Users are able to select the desire level of detail and set the message limits.

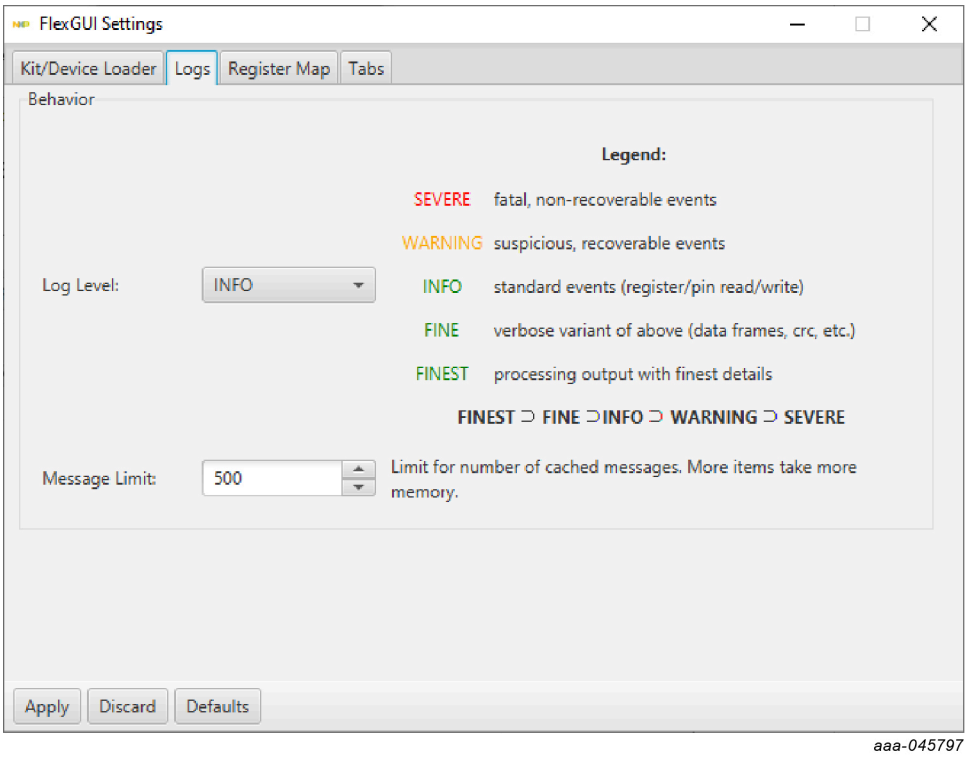


Figure 42. Log and message limits

The log levels pulldown menu is shown in Figure 43.

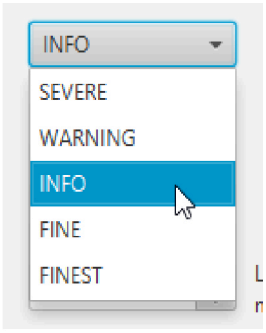


Figure 43. Log levels

4.4.3 Register map

The register map tab sets configuration settings to generate a register map automatically for target devices. The following options alter the register map appearance and behavior.

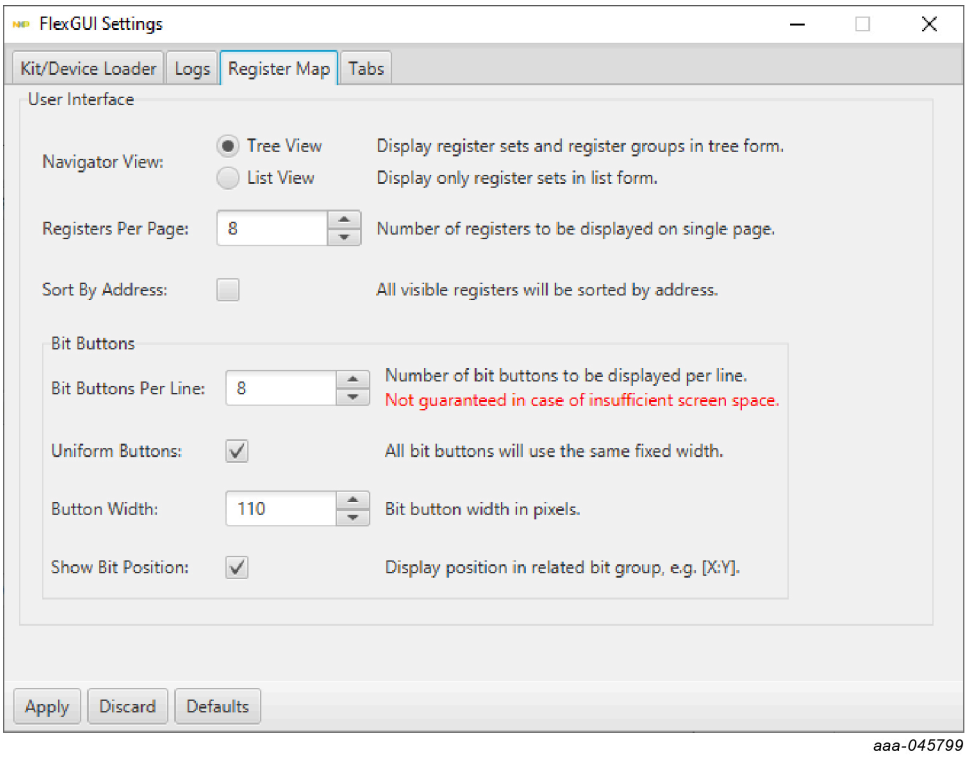


Figure 44. FlexGUI register map tab settings

The GUI works internally with this register structure, describing the register map of the device.

Register Set (set of groups of related registers)

Register Group (set of related registers)

Register (single register)

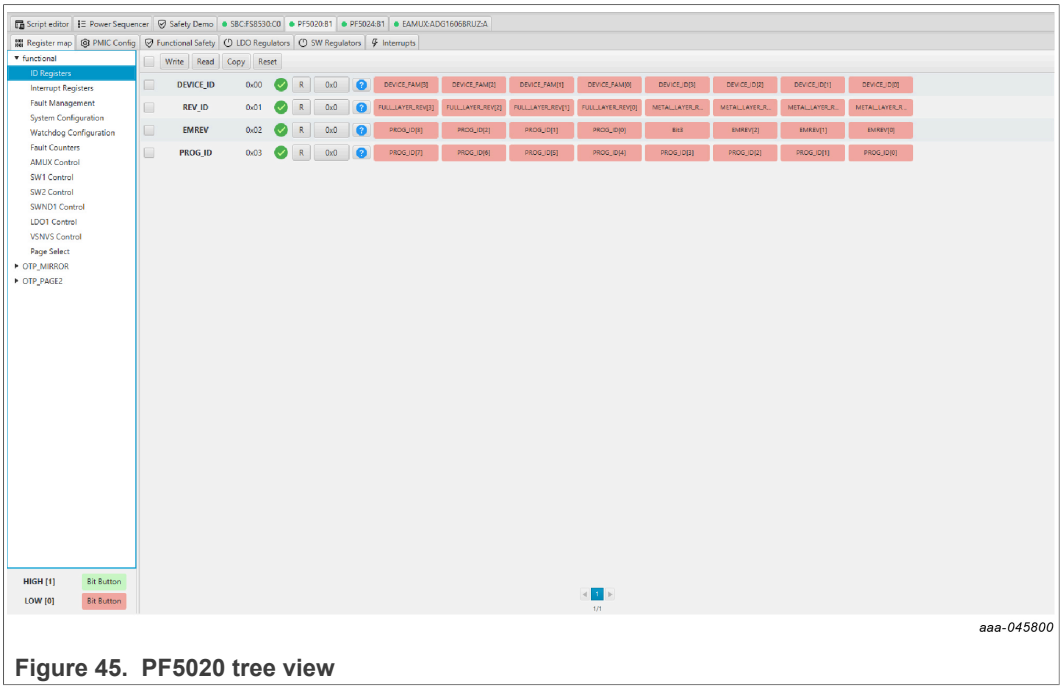
Bit Group (bitwise group in register, which represents configuration item)

Bit Option (possible configuration choice for given bit group)

The Navigator View option is tightly coupled with this structure.

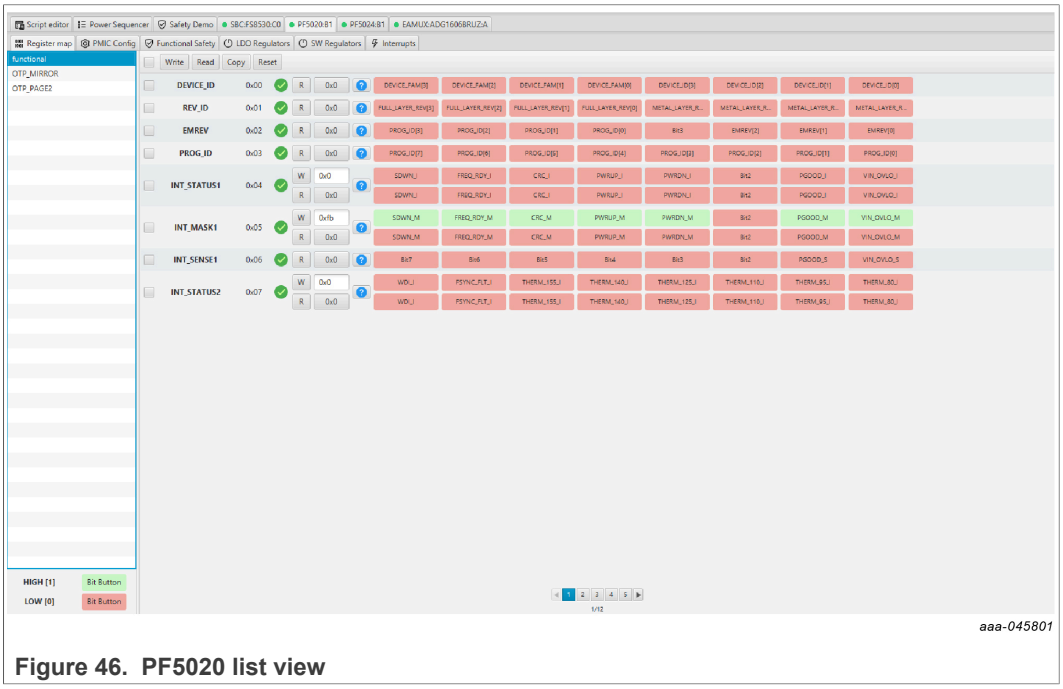
- **Tree View** – displays a view where register sets and register groups are displayed in tree form. The tree view option enables users to navigate quickly through the available registers by narrowing down the choices related only to registers.
- **List View** – displays a view where only the register sets are presented, in list form, leaving the user fewer choices for selection. Registers are grouped by the parent register sets.

Note: *Figure 45 and Figure 46 present PF5020 registers in tree and list view.*



aaa-045800

Figure 45. PF5020 tree view



aaa-045801

Figure 46. PF5020 list view

4.4.4 Tabs

This tab contains configuration options related to all GUI tabs, see Figure 47.

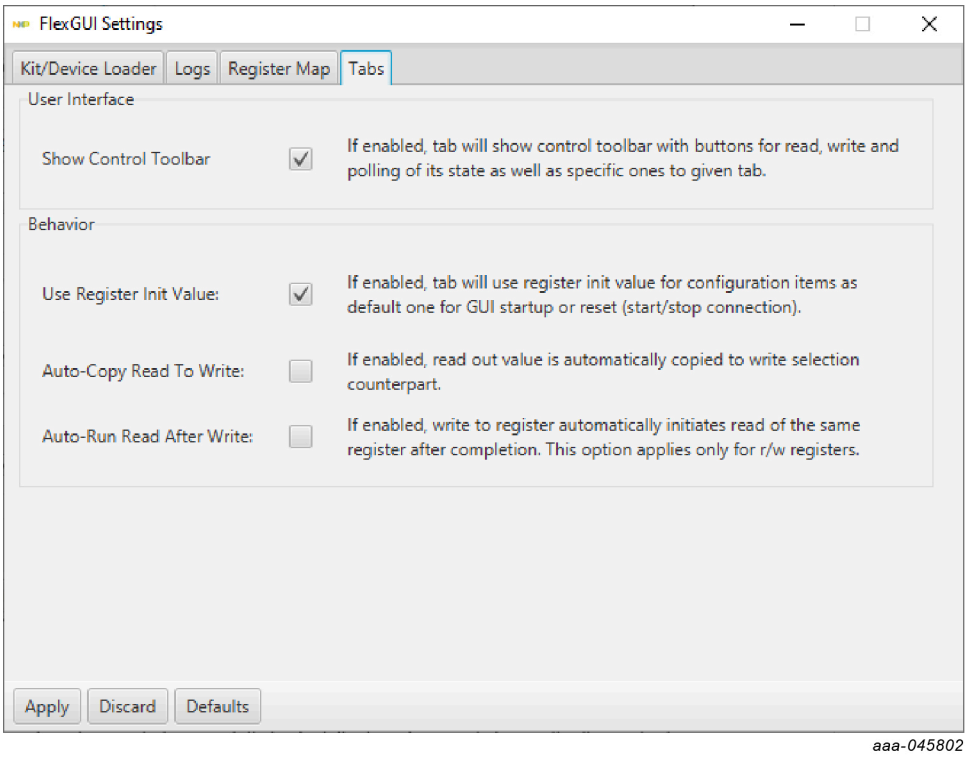


Figure 47. GUI tabs configuration

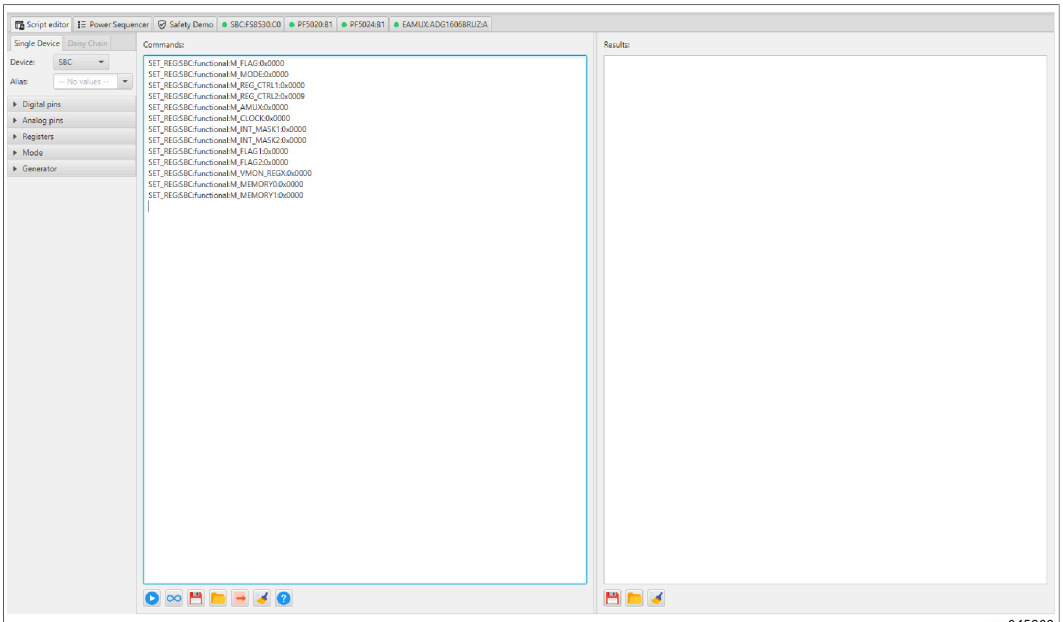
Note:

It is guaranteed that these preferences are reflected in any flexible tab (with blue boxes) and register map.

Some device-specific tabs may not implement these preferences, therefore any modification on this tab may not work as expected for those devices.

4.5 Script editor tab

The script editor is an embedded tool enabling the sequential execution of commands, which can access registers, and the digital and analog pins of a device. The graphical interface facilitates the creation of and working with commands. Section 4.5.1 through Section 4.5.5 provide detailed information.

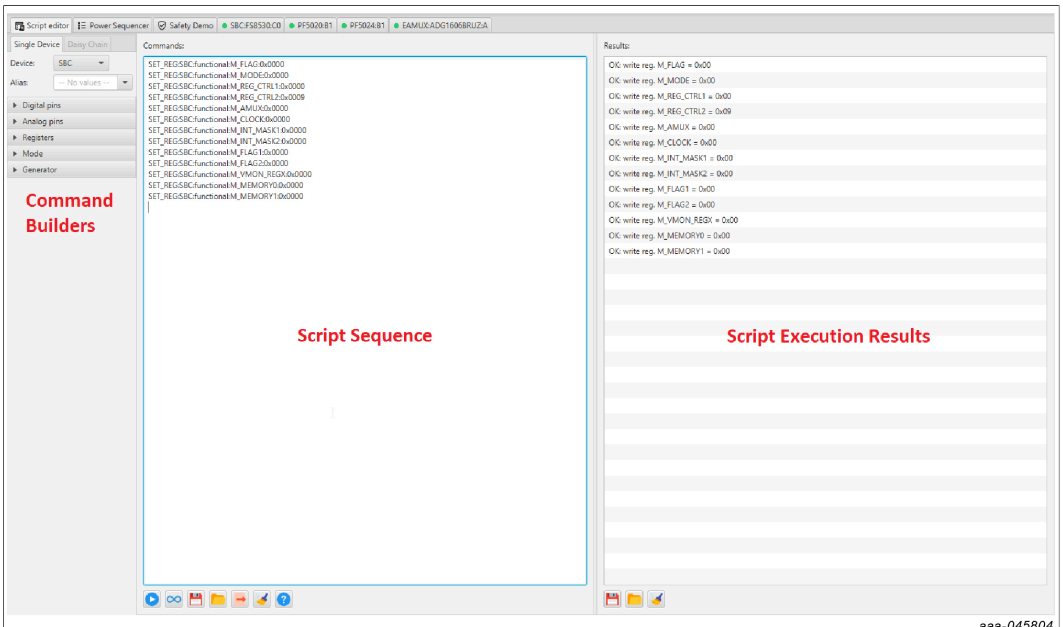


aaa-045803

Figure 48. Script editor tab

4.5.1 Script editor layout

The script editor layout consists of three blocks as shown in Figure 49.



aaa-045804

Figure 49. Script editor layout

- **Command Builders** – assists the user with the creation of commands or allows the user to generate pre-defined script sequences for the current device configuration.
- **Script Sequence** – contains a list of commands prepared for execution, one command per line. Users can modify the script directly or use command builders. The command format is described in Section 4.5.3.

- **Script Execution Results** – displays the script processing results.

4.5.2 First steps

Use the following steps that describe how to create and execute a simple script. For reference, each step is identified in Figure 50:

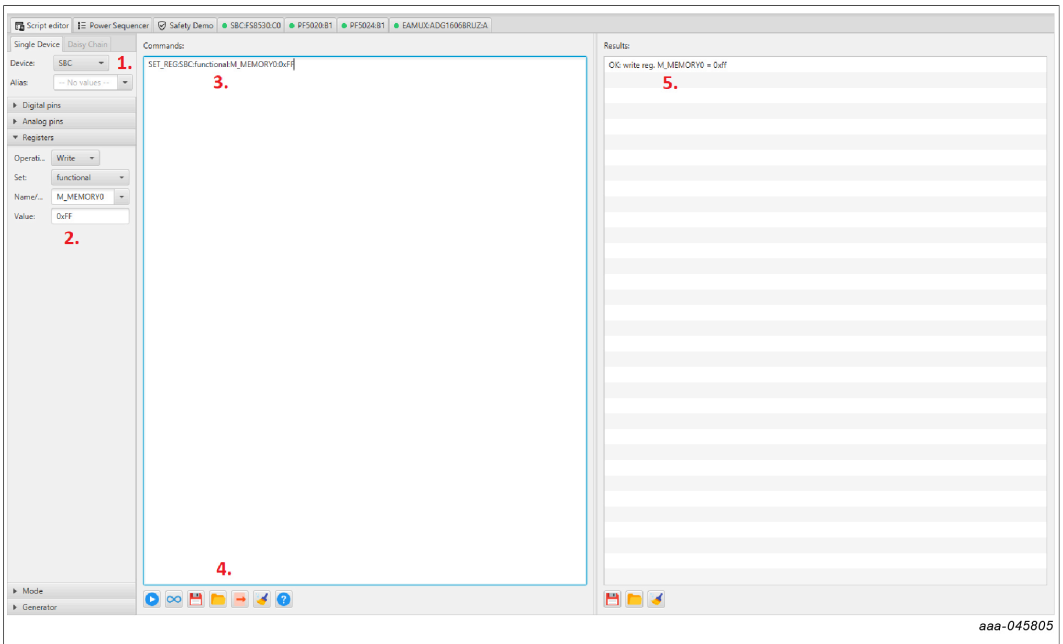


Figure 50. Creation and evaluation of a script

1. In the command builders block, select a device. The tool loads the appropriate values into boxes (register names and pin names, for example).
   a. In the event the script was generated with a different device name, a spreadsheet, for example, an alias name may be used, making the GUI backward compatible. Using the command builders (digital/analog pins, registers, and so forth), automatically generates commands with the selected device name including the alias if provided.
2. Create commands to be executed. Users can modify the script sequence directly or utilize command builders which ensure valid options.
   a. To insert a command, select command attributes and the tool inserts a new line into the script sequence. The write register command requires the `Reg. value` which is decimal or hexadecimal value (0x prefix, for example, 0x12).
3. With the cursor in the write value text field, press enter to include the write command into the script sequence.
4. To process the script in a loop, set the run in using the loop option in the controls found under script sequence then run the script. Using the loop option, the script repeats until the user presses Stop.
5. The script execution results block is populated with result for each single comment.
   Note: *These results are cleared when another process starts.*
   The main notification area reports the overall result, see Figure 50.

Users may save the created script into a file or load a prepared script. Likewise the execution results may be saved or loaded from a file. An `export` button stores write register commands only, all other commands are omitted in ATE format as shown:

```
<register address>,<register value to be written>,
```

4.5.3 Definition of commands

This section describes commands supported by the script editor and their format. All commands are listed in Table 3.

Table 3. Script editor commands

Command name	Description
SET_REG	Sets value of a defined selected register (either by name or address).
GET_REG	Gets value of a defined selected register (either by name or address).
SET_REG_ND	Sets value of an undefined selected register (only by address).
GET_REG_ND	Gets value of an undefined selected register (only by address).
SET_REG_DC	Sets value of selected register of lead device for the whole daisy chain group.
GET_REG_DC	Gets value of selected register of lead device for the whole daisy chain group.
SET_DPIN	Sets value of a selected digital pin.
GET_DPIN	Gets value of a selected digital pin.
GET_APIN	Gets value of a selected analog pin. Returned value is in mV.
PAUSE	Shows a dialog with a user-defined message. The script execution is paused until the user confirms the dialog.
DELAY	Delays script execution for specified amount of time in milliseconds.
EXIT	Stops execution of the script.
SET_MODE	Sets device mode. List of modes is device-dependent.

***Note: *_ND (not defined) commands** are intended for internal use only unless otherwise stated. These commands allow communication with any register address, even if it is not explicitly defined (for example, metadata for size, feasible options, and so forth).*

4.5.4 Format of commands

The general format of script editor commands is:

```
<command name>:<list of parameters separated by a colon>
```

Table 4 shows parameters of script editor commands. All parameters are mandatory.

Table 4. Parameters of script editor commands

Command name	1. item	2. item	3. item	4. item
SET_REG	Device	Reg. set	Reg. name or Reg. address	Reg. value
GET_REG	Device	Reg. set	Reg. name or Reg. address	—
SET_REG_ND	Device	Bus (routing rule)	Reg. address	Reg. value
GET_REG_ND	Device	Bus (routing rule)	Reg. address	—
SET_REG_DC	Group	Reg. set	Reg. name	Reg. value
GET_RED_DC	Group	Reg. set	Reg. name	—
SET_DPIN	Device	Pin name	Pin value	—
GET_DPIN	Device	Pin name	—	—
GET_APIN	Device	Pin name	—	—
PAUSE	Message	—	—	—
DELAY	Time [ms]	—	—	—
EXIT	—	—	—	—
SET_MODE	Device	Mode	—	—

The parameters mentioned in Table 4 are described as follows:

- **Group:** daisy chain group.
- **Device:** device name.
- **Reg. set:** register set name. Register sets allow users to associate registers having similar function.
- **Reg. name:** register name as defined in a data sheet.
- **Reg. address:** register address in the decimal or hexadecimal (with 0x prefix) format.
- **Reg. value:** register value in the decimal or hexadecimal (with 0x prefix) format.
- **Bus (routing rule):** bus which should be used for data transfer.
- **Pin name:** name of a digital or analog pin as defined in a device data sheet.
- **Pin value:** value of digital pin. Allowed strings are low and high.
- **Message:** a message to be displayed in a dialog. It cannot contain the colon character, which is used as a delimiter of parameters.
- **Time [ms]:** time in milliseconds (max. is 60000) for script execution delay, note that it takes approximately 15 ms to 25 ms to complete a single command.
- **Mode:** device mode.

4.5.5 Script example

In this example script, the name of registers, register sets, devices, and pins depend on the particular device.

```
// Sets the 'M_FLAG' register in the 'functional' register set
 to value 0x00.
SET_REG:SBC:functional:M_FLAG:0x00
```

```
// Gets value of the 'M_FLAG' register in the 'functional'
 register set.
GET_REG:SBC:functional:M_FLAG
// Sets the register with address '0x01' to value 0x00 and uses
 SPI1 routing rule.
SET_REG_ND:SBC:SPI1:0x01:0x00
// Gets value of the register with address '0x01' and uses SPI1
 routing rule.
GET_REG_ND:SBC:SPI1:0x01
// Sets the 'M_FLAG' register in the 'functional' register set
 to value 0x00 for all devices in chain named as 'Group 0'.
SET_REG_DC:GROUP_0:functional:M_FLAG:0x00
// Gets value of the 'M_FLAG' register in the 'functional'
 register set for all devices in chain named as 'Group 0'.
GET_REG_DC:GROUP_0:functional:M_FLAG
// Sets value of 'WAKE2_KL25Z' digital pin as high.
SET_DPIN:SBC:WAKE2_KL25Z:HIGH
// Gets value of the 'FS0B' digital pin (low/high).
GET_DPIN:SBC:FS0B
// Gets value of the 'AMUX' analog pin.
GET_APIN:SBC:AMUX
// Shows a dialog with user defined message.
PAUSE: "Please continue when jumper XYZ is set as ... and
 switch XYZ is ON/OFF."
// Delays script execution for 1000 ms.
DELAY:1000
// Stops script execution.
EXIT
// Sets SBC's mode as 'test-mode'.
SET_MODE:SBC:test-mode
```

4.6 Register map tab

The register map tab is a common tab, enabling users to work with device registers, for example, to read and write (Figure 51). The tab contains a navigation view panel (Figure 52) with a list of register sets or a register tree structure, and pagination with lists of underlaying registers.

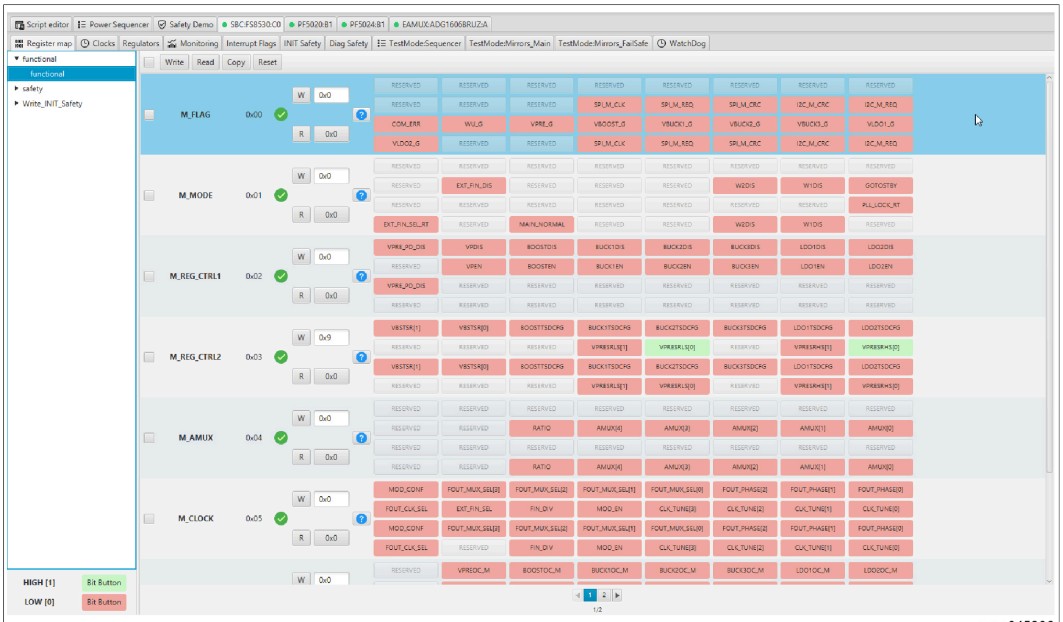


Figure 51. FS8xx0 register map

The left panel in the tab, a flat view enumerates list of register sets (Figure 52). When users click a register set item, the register map displays all underlaying registers.

Figure 52. Navigation view panel

Each register item is represented by read and/or write row depending on its definition. A single item consists of register selection checkbox, register name, register address, read/write control, register value help, and bit buttons. Bit buttons represent values of individual bits, green represents logic 1 and red represents logic 0.

For read, click the “R” button (Figure 53) and the read out value appears in button on the right. If clicked, this button, copies and pastes the read out value into write value input box above.

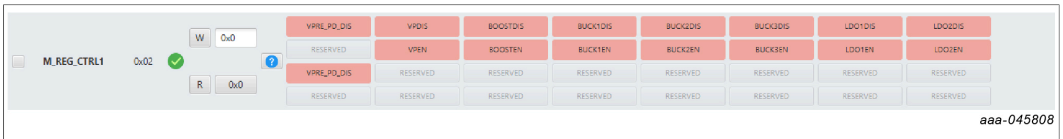


Figure 53. FS8xx0 register read example

For write, provide write value and click the “W” button (Figure 53). The write value can be provided as raw number (hexadecimal or decimal), with use of bit buttons or register value helper. The status icon changes to modified until a write action happens.

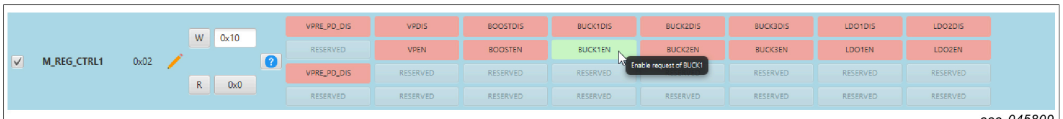


Figure 54. FS8xx0 register write example

The register value helper encodes and decodes feasible register value options in user-friendly way.

Figure 55. FS8xx0 register value helper

In case the read or write operation fails, the status icon changes to red cross (Figure 56).

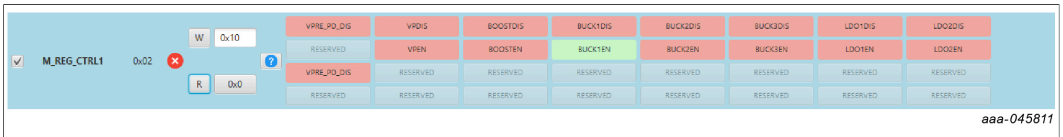


Figure 56. FS8xx0 operation fails icon

All value changes are stored locally until user explicitly confirms them via R/W buttons. If user decides to confirm the changes all at once, simply select all register items and use the related operation buttons above the current page. (Figure 57)

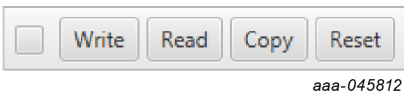


Figure 57. FS8xx0 R/W button

The buttons operate as follows:

- **Write:** writes all selected registers.
- **Read:** reads all selected registers.
- **Copy:** copies read out values to write values for all selected registers.
- **Reset:** resets modified values to their last know values for all selected registers.

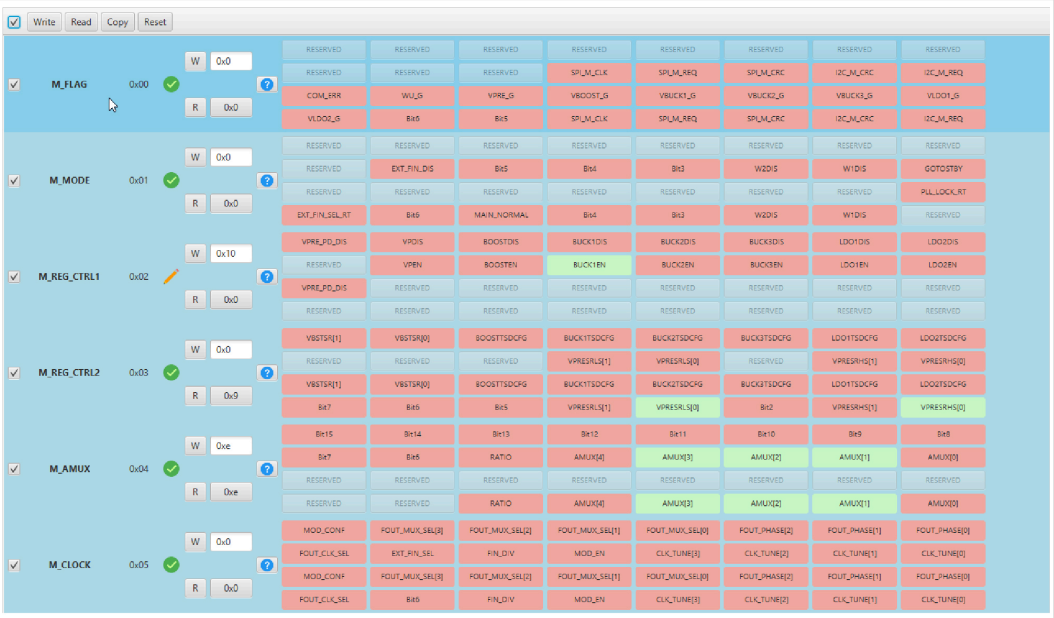


Figure 58. FS8xx0 R/W button

If there are more register items than registers per page limit, use the pagination controls to move over pages. (Figure 59)

Figure 59. FS8xx0 pagination controls

5 FS8510 tabs and features

This section describes the tabs and features specific to the FS8510.

5.1 Working modes

The GUI enables users to evaluate the device in the following modes (besides user mode).

- **Normal Mode**
- **Debug Mode**
  - **Enter:** Apply 5 V on the DBG pin before the power switch is turned ON.
  - **Leave:** Write DBG_EXIT bit = '1' in FS_STATES register.
  - **Behavior:**
    - The Watchdog window fully opens, the Deep Fail-Safe request from the Fail-safe state machine (DFS = 1) is masked, the 8 s timer monitoring of RSTB pin is disabled, the Failsafe output pin FS0B cannot be released, and the OTP emulation and programming of a raw device by SPI/I^2C is possible.
    - No watchdog refresh is required. Disabling the watchdog allows an easy debug of the hardware and software routines (i.e SPI/I^2C commands). However, the whole

watchdog functionality is kept on (seed, LFSR, WD refresh counter, WD error counter...). WD errors are detected and counted with reaction on the RSTB pin.

- **Test Mode**
  - **Pre-conditions:** Device must be in Debug Mode (DBG pin asserted high).
  - **Enter/Leave:** Send special **key-sequence** (embedded in the application).
  - **Behavior:**
    - In this mode, it is possible to configure mirror registers and apply the configuration either **temporarily** or **permanently**. See Section 5.4 for details.

5.2 Communication

The FS8510 supports communication via SPI or I^2C (SPI only for FS6600). Data frames are assembled according to description in the Figure 60 for SPI and respectively in the Figure 61 for I^2C.

The CRC calculation is handled for both types of communication. See the device data sheet for details on CRC calculation.

If the CRC integrity check fails for received data, it is evaluated as invalid response frame and the user is properly informed.

	B31	B30	B29	B28	B27	B26	B25	B24								
MOSI	M/FS	Adr_5	Adr_4	Adr_3	Adr_2	Adr_1	Adr_0	R/W								
MISO	General device status															
	B23	B22	B21	B20	B19	B18	B17	B16	B15	B14	B13	B12	B11	B10	B9	B8
MOSI	Data_15	Data_14	Data_13	Data_12	Data_11	Data_10	Data_9	Data_8	Data_7	Data_6	Data_5	Data_4	Data_3	Data_2	Data_1	Data_0
MISO	Extended device status, or device internal control register content or device flags															
									B7	B6	B5	B4	B3	B2	B1	B0
MOSI									CRC_7	CRC_6	CRC_5	CRC_4	CRC_3	CRC_2	CRC_1	CRC_0
MISO									CRC_7	CRC_6	CRC_5	CRC_4	CRC_3	CRC_2	CRC_1	CRC_0

aaa-045815

Figure 60. SPI communication frame

B39	B38	B37	B36	B35	B34	B33	B32	B31	B30	B29	B28	B27	B26	B25	B24
ID_6-0							R/W	0	0	Adr_5-0					
Device Address							Read/Write			Register Address					
B23	B22	B21	B20	B19	B18	B17	B16	B15	B14	B13	B12	B11	B10	B9	B8
Data_15	Data_14	Data_13	Data_12	Data_11	Data_10	Data_9	Data_8	Data_7	Data_6	Data_5	Data_4	Data_3	Data_2	Data_1	Data_0
Data MSB								Data LSB							
								B7	B6	B5	B4	B3	B2	B1	B0
								CRC_7	CRC_6	CRC_5	CRC_4	CRC_3	CRC_2	CRC_1	CRC_0
								CRC							

aaa-045816

Figure 61. I^2C communication frame

5.2.1 Mirror register aliases / transparent access

The device control logic provides transparent access to the main/fail-safe mirror registers.

In order to read/write the desired configuration, specifying data in the **MIRRORDATA** and address + R/W in **MIRRORCMD** registers is usually required.

The GUI addresses the previously mentioned data specification issue so the user can directly address mirror registers using their aliases (**OTP_CFG_XXX**).

5.3 Generated scripts

This section identifies scripts which can be generated and exported by the GUI.

Table 5. Scripts for export

Script name	Purpose
mirror-otp-config	Enters Test Mode. Configures mirror registers. Commits CRC.
functional-config	Configures functional registers.

5.4 Programming

The GUI enables users to program the device with predefined content of mirror registers. This procedure is temporary and is cleared after a system power-down.

Note: *Ensure that the correct device model is selected matching the samples being used on the kit.*

5.4.1 Emulation

The emulation procedure enables users to temporarily program the device with a custom configuration.

In order to temporarily program the device, follow these steps:

1. Make sure that GUI is in **Test mode**, see Section 5.1 for details.
2. Load the custom configuration.
   a. Make a custom configuration using provided tabs.
      i. Mirrors:Main - see Section 5.5.9 for details.
      ii. Mirrors:FailSafe - see Section 5.5.10 for details.
   b. Load previously generated script called “**mirror-otp-config**” into the script editor and run it.
3. Release DEBUG pin to GND.
4. A custom configuration is **temporarily** programmed and ready to be evaluated.

5.5 FS8510 tabs

This section describes the FS8510 specific tabs.

5.5.1 Register map tab

See Section 4.6 for details on using this tab.

5.5.2 Clocks tab

The clocks tab provides clock configuration for voltage regulators. It adapts to selected device modes and enables only accessible options, the rest are grayed out.

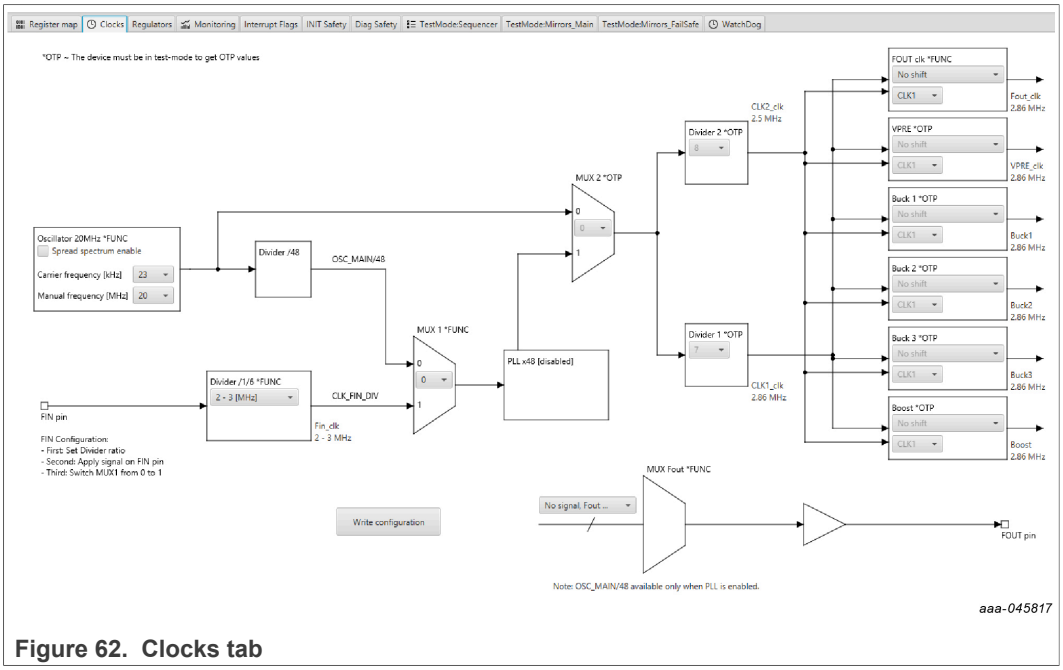


Figure 62. Clocks tab

5.5.3 Regulators tab

The regulators tab provides configuration options for low voltage regulators and a static calculator for high-voltage buck slope compensation network.

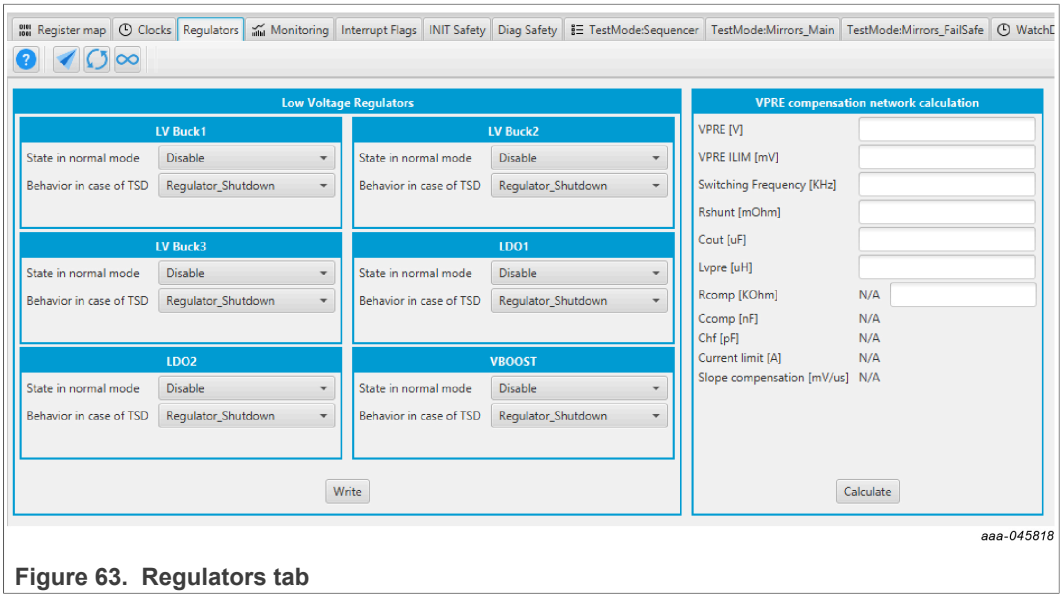


Figure 63. Regulators tab

5.5.4 Monitoring tab

The monitoring tab provides ADC monitoring of possible outputs of the AMUX pin and other pins dedicated for individual voltage regulators.

aaa-045819

Figure 64. Monitoring tab

The tab presents two separate charts for monitoring of voltages and temperature.

5.5.5 Interrupt flags tab

The interrupt flags tab provides configuration options and an overview for device events. These might be optionally cleared or masked. It can be read out one-time or periodically with polling. Indicators in the status column can be interpreted as no event occurred or event occurred.

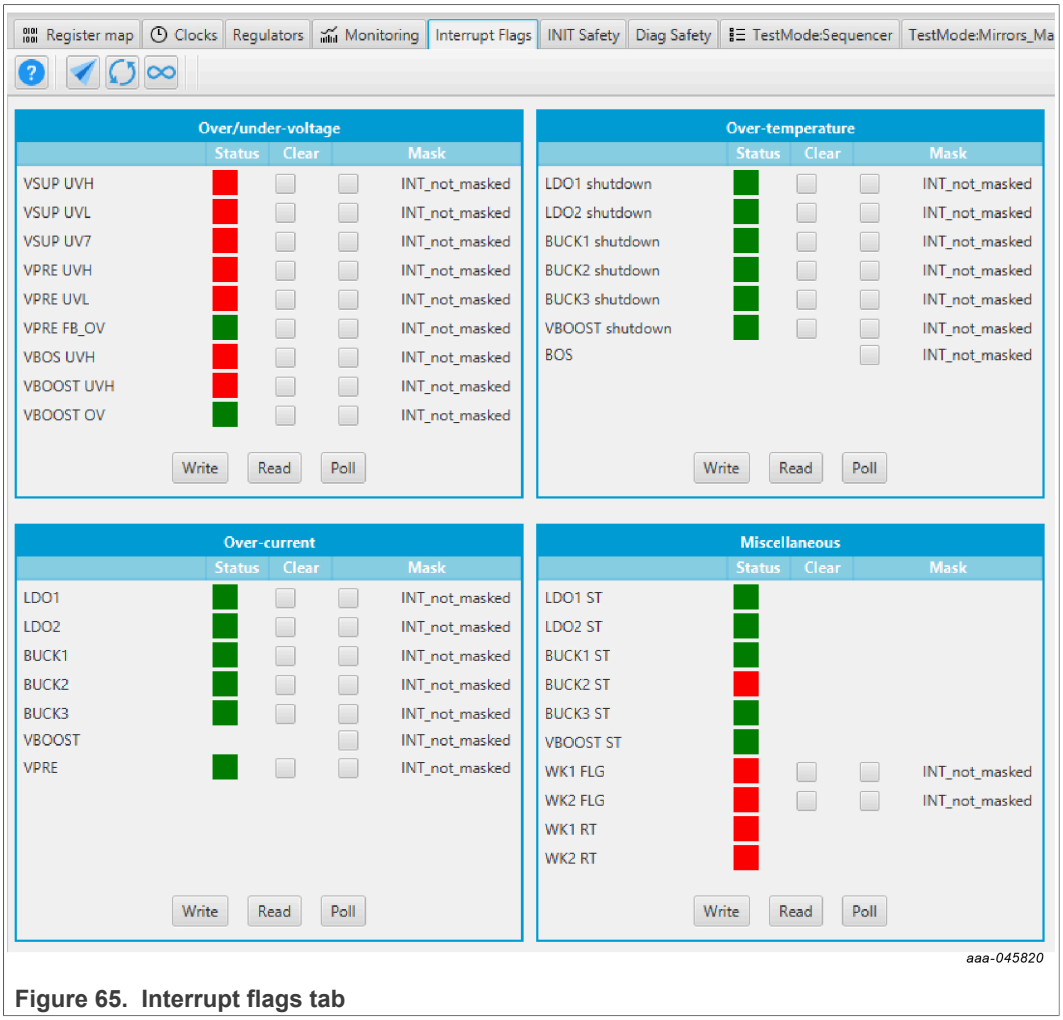


Figure 65. Interrupt flags tab

5.5.6 INIT safety tab

The INIT safety tab provides configuration options during the INIT phase. Indicators in the FS0B and RSTB columns can be interpreted as a fault error with impact on (green) or a fault error w/o impact on (red).

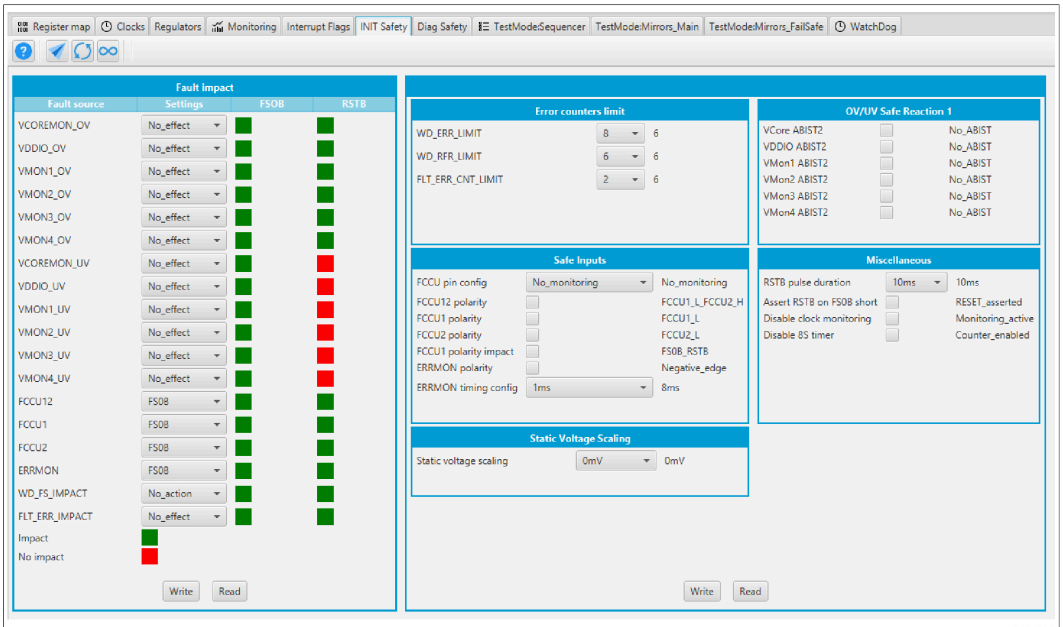


Figure 66. INIT safety tab

5.5.7 Diag safety tab

The diag safety tab provides configuration options during INIT phase and safety diagnostics.

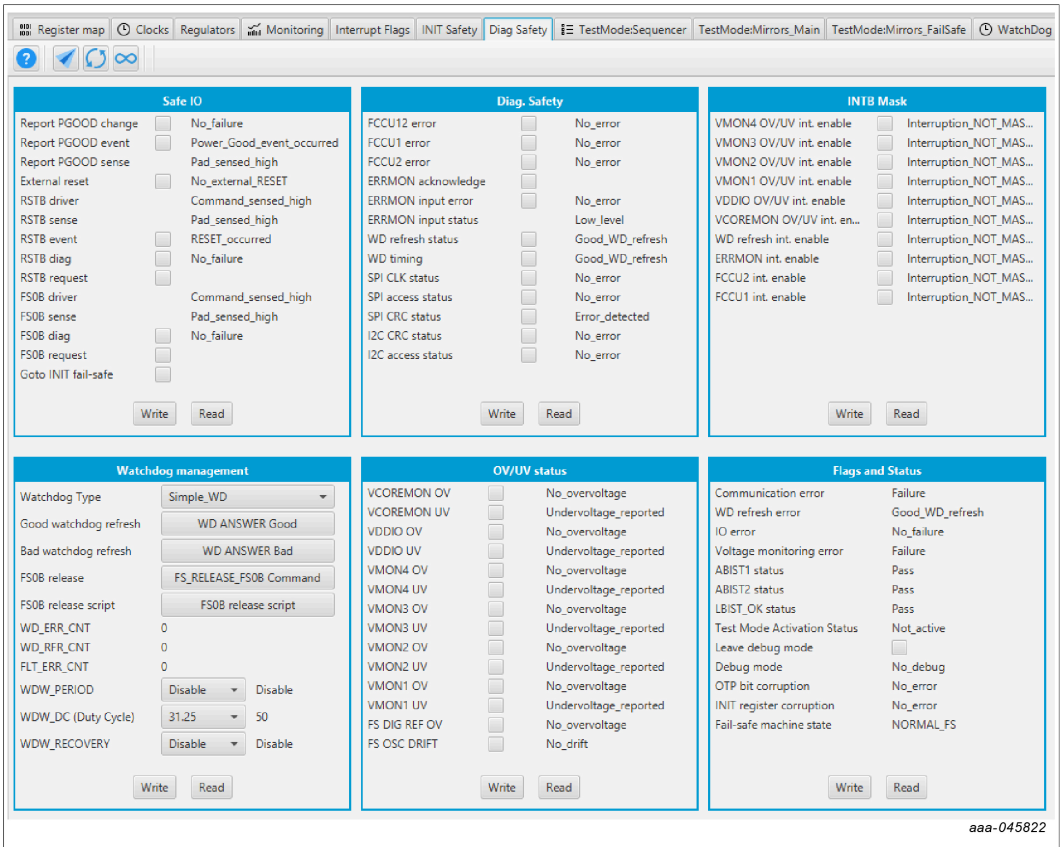


Figure 67. Diag safety tab

5.5.8 Sequencer tab

The sequencer tab provides configuration options for the power-up sequence of regulators.

Each regulator can be assigned to specific slot, which determines power up time during startup.

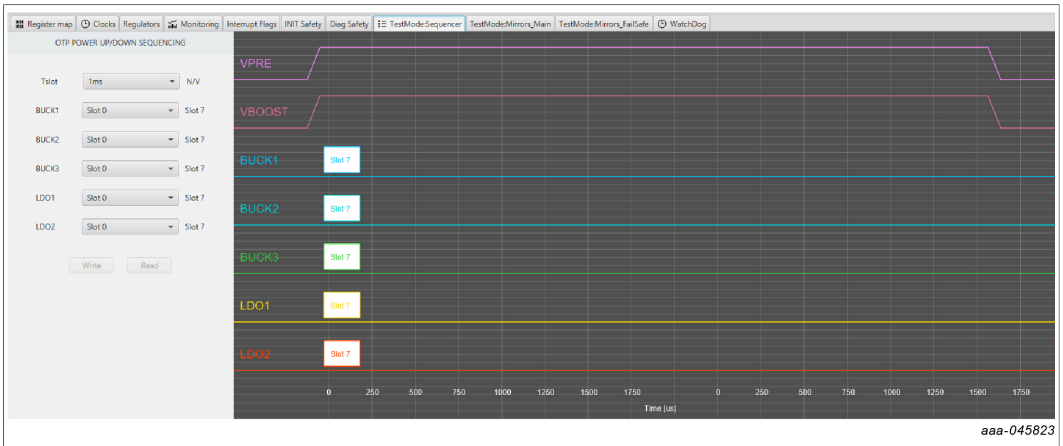


Figure 68. Sequencer tab

5.5.9 Mirrors:Main tab

The Mirrors:Main tab provides configuration options for the mirror register of the main state machine. It is enabled only if test mode is active.

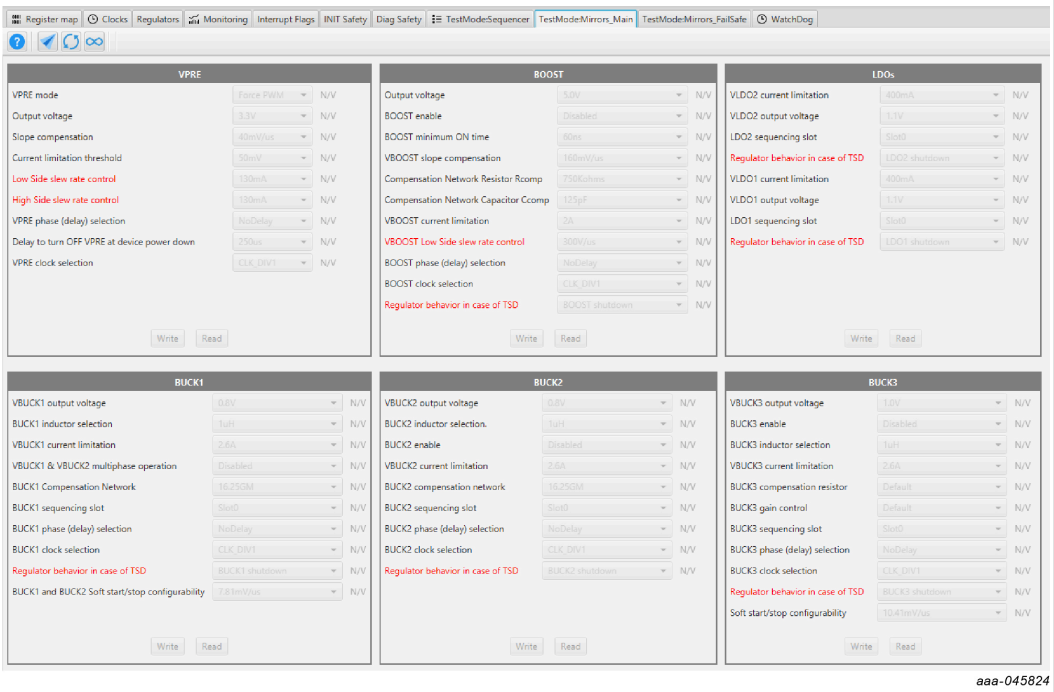


Figure 69. Mirrors:Main tab

5.5.10 Mirrors:FailSafe tab

The Mirrors:FailSafe tab provides configuration options for mirror registers of the fail-safe state machine. It is enabled only if test mode is active.

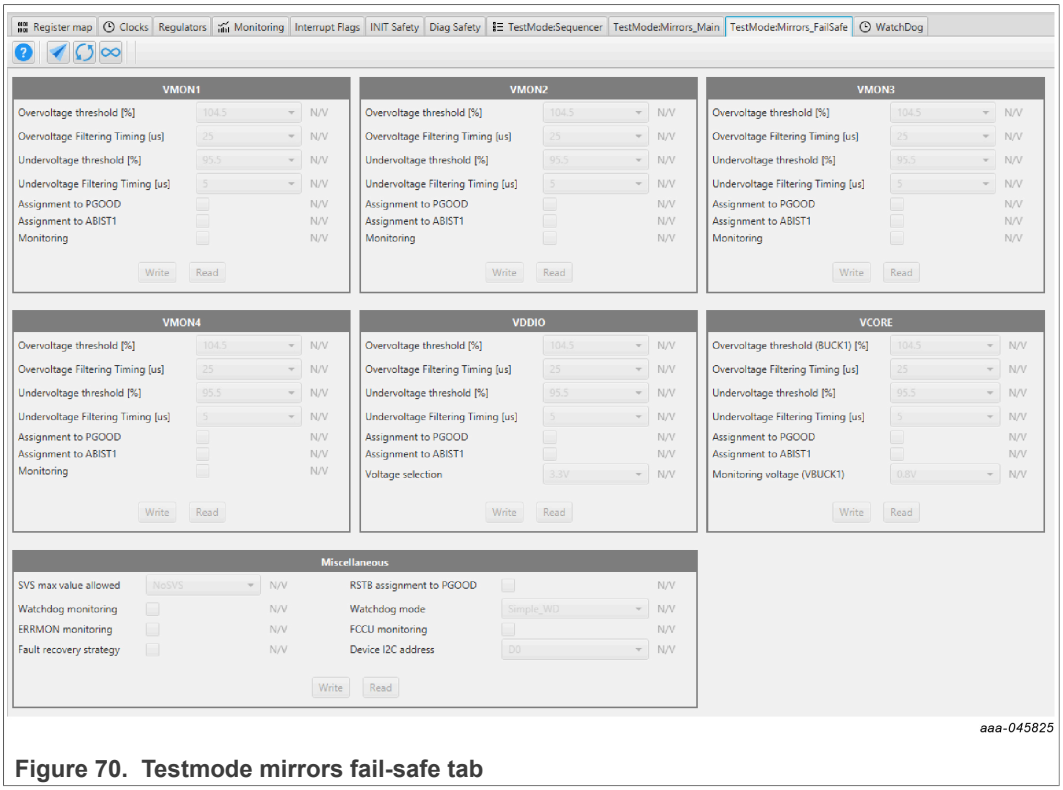


Figure 70. Testmode mirrors fail-safe tab

6 PF502x tabs and features

This section describes the specific tabs and features for the PF5020 and PF5024.

6.1 Working modes

The GUI supports following modes in addition to user mode:

- **Run mode (called normal mode in the GUI)**
  - **Enter:** If the power-up state is successfully completed, the state machine transitions to the run state. In this state, RESETBMCU is released high, and the MCU is expected to boot up and set up specific registers on the PMIC as required during the system boot up process. PWRON is pulled high, TBBEN pin is pulled low.
  - **Behavior:**
    - The run mode is intended for use as the normal mode of operation for the system.
    - Each regulator has specific registers to control its output voltage, operation mode, and/or enable/disable state during the run state.
    - In a typical system, each time the processor boots up (PMIC transitions from off mode to run state), all output voltage configurations are reset to the default OTP

configuration and the MCU should configure the PMIC to its desired usage in the application.

- Standby mode (called normal mode in the GUI)
  - **Enter:** The standby state is entered when the STANDBY pin is pulled high or low as defined by the STANBYINV bit. The STANDBY pin is pulled high/low by the MCU to enter/exit system low-power mode.
  - **Behavior:**
    - The standby state is intended for use as a low power (state retention) mode of operation. In this state, the voltage regulators can be preset to a specific low power configuration in order to reduce the power consumption during system sleep or state retention modes of operations.
    - Each regulator has specific registers to control its output voltage, operation mode, and/or enable/disable state during the standby state.
    - At power-up, the standby registers are loaded with the same default OTP values as the run mode. The MCU is expected to program the desired standby values during boot up.
- **TBB mode**
  - **Enter:** In order to access the TBB mode, pull the the TBBEN high and PWRON low.
  - **Behavior:**
    - Allows a temporary configuration to debug or test a customized power-up configuration in the system.
    - In TBB mode, the following conditions are valid:
      - I^2C communication uses standard communication with no CRC and secure write disabled.
      - Default I^2C address is 0x08 regardless of the address configured by OTP.
      - Watchdog monitoring is disabled (including WDI and internal watchdog timer).
    - The device communicates through I^2C as long as VDDIO is provided to the PMIC externally.

6.2 Communication

The PF502x devices integrate an I^2C interface for data transfer with speeds up to 3.4 MHz. Common data frame configuration is used: 7bit address with R/W bit, 1 byte for command, and trailing byte for data.

The CRC calculation is handled automatically. See the device data sheet for details on CRC calculation.

In case of CRC integrity check failure when enabled, the data frame is considered invalid and the event is reported via the GUI information panel.

6.3 Generated scripts

Table 6 identifies the following scripts generated and exported by the GUI.

Table 6. Scripts for Export

Script Name	Purpose
tbb-config	Enters TBB mode. Configures mirror registers. Enters run mode in order to apply the configuration.
mirror-registers	Configures mirror registers.
functional-registers	Configures functional registers.

6.4 Programming

The GUI facilitates OTP burning procedures with predefined content of mirror registers. This procedure is temporary and is cleared after a system power-down.

Note: *Ensure the correct device model matching the samples used on the kit is selected.*

6.4.1 Try-Before-Buy evaluation

The Try-Before-Buy evaluation enables users to temporarily program the device with a custom configuration.

Follow these steps to temporarily program the device.

1. Ensure sure that the GUI is in **TBB mode**.
2. Load the custom configuration.
   a. Make your custom configuration using the provided tabs. Only relevant configuration items are available or
   b. Load previously generated script called “**mirror-config**” into the script editor and run it.
3. In device panel/scripts, select the script called “**tbb-script**” to be generated into the script editor.
4. Run the script.
5. The custom configuration is **temporarily** programmed and ready to be evaluated.

6.5 PF5020 tabs

This section describes the PF5020 specific tabs.

6.5.1 Register map tab

See Section 4.6 for details on using this tab.

6.5.2 PMIC configuration tab

The PMIC configuration tab provides configuration options for general device operation. It adapts to the selected device mode and enables only accessible options, the rest are grayed out.

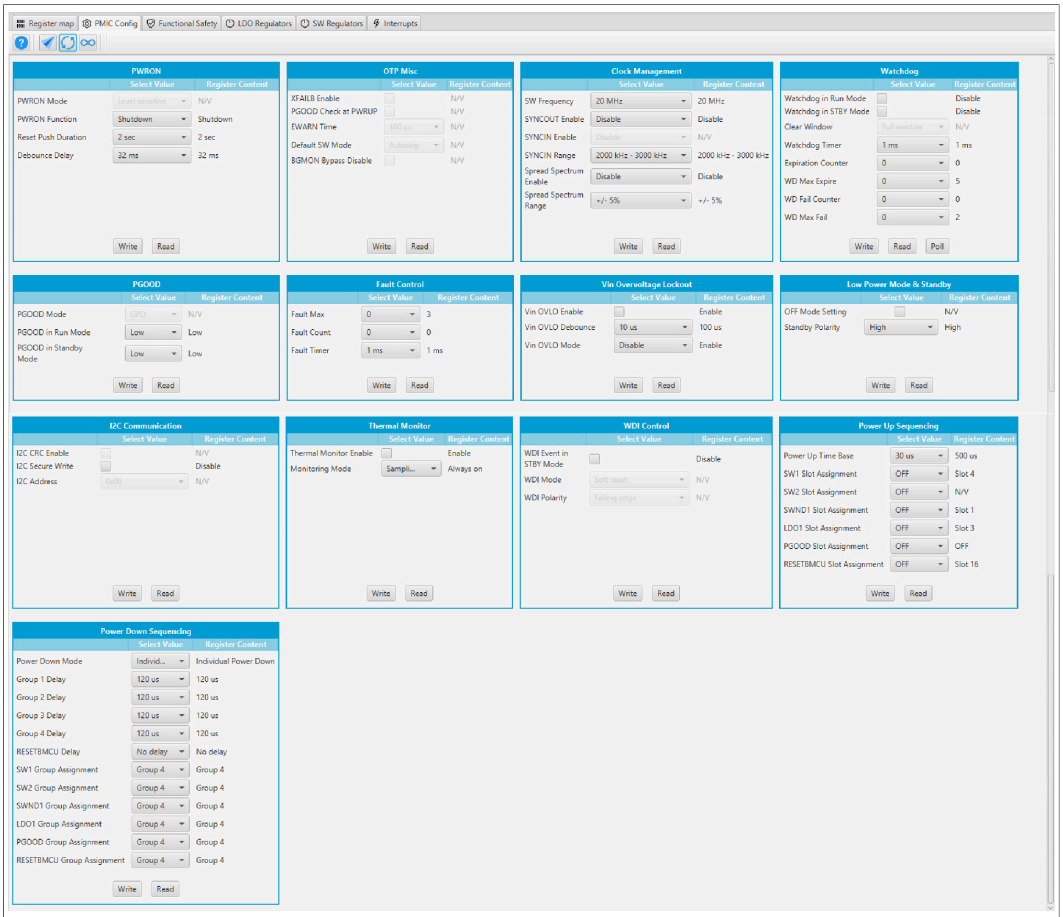


Figure 71. PMIC configuration options

6.5.3 Functional safety tab

The functional safety tab provides configuration options for functional safety features. It adapts to selected device mode and enables only accessible options, the rest are grayed out.

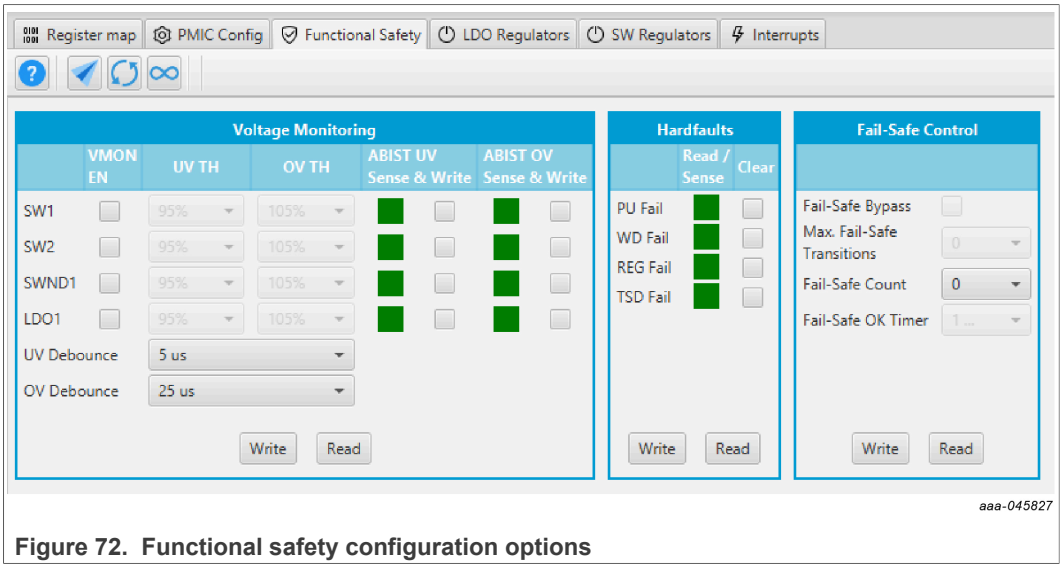


Figure 72. Functional safety configuration options

6.5.4 LDO regulators tab

The LDO regulators tab provides configuration options for LDO regulators and VSNVS. It adapts to the selected device mode and enables only accessible options, the rest are grayed out.

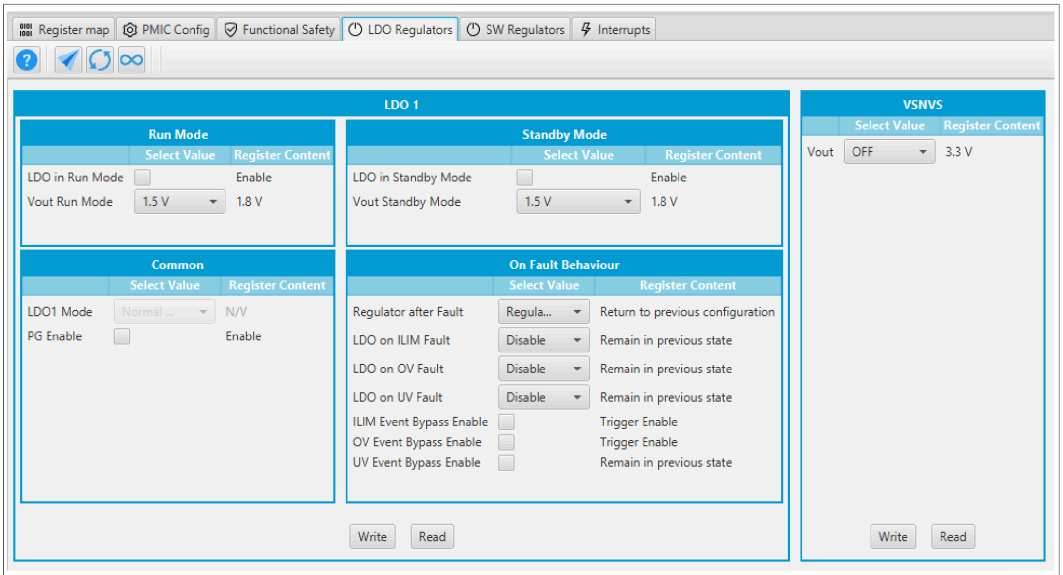
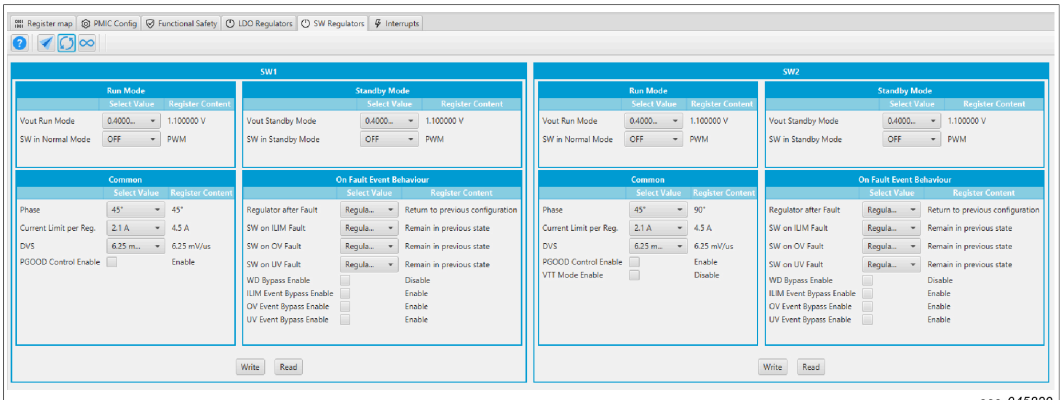


Figure 73. LDO regulator configuration options

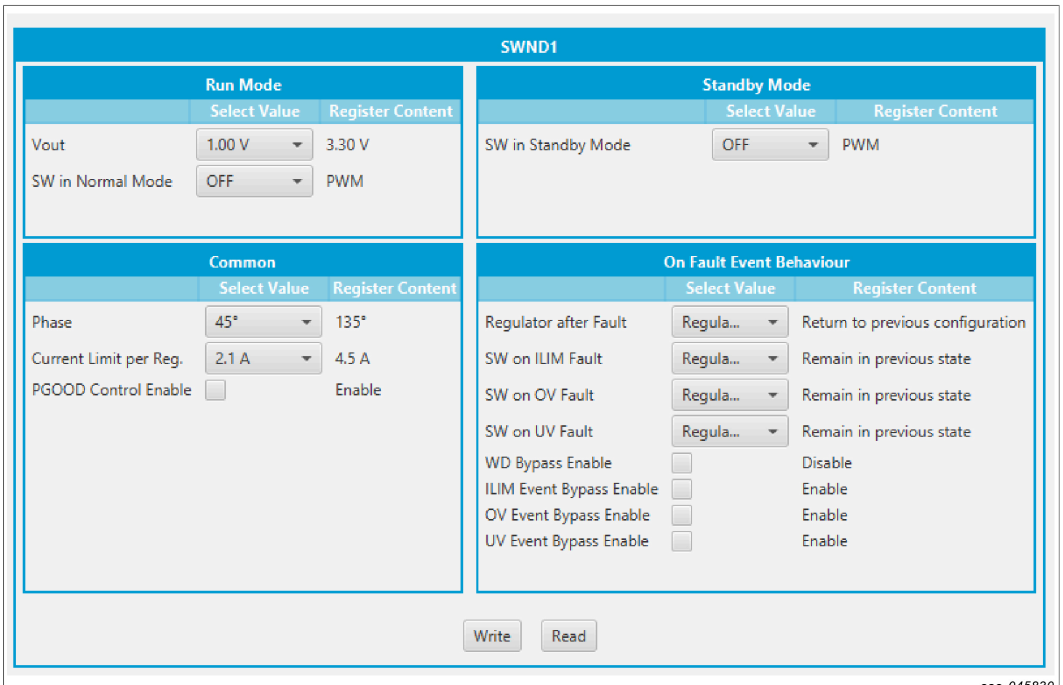
6.5.5 SW regulators tab

The SW regulators tab provides configuration options for SW regulators. It adapts to selected device mode and enables only accessible options, the rest are grayed out.



aaa-045829

Figure 74. SW1 and SW2 regulator configuration options



aaa-045830

Figure 75. SWND1 regulator configuration options

6.5.6 Interrupts tab

The Interrupts tab provides configuration options and an overview for device events. These may be optionally cleared or masked. It can be read out one-time or periodically with polling.

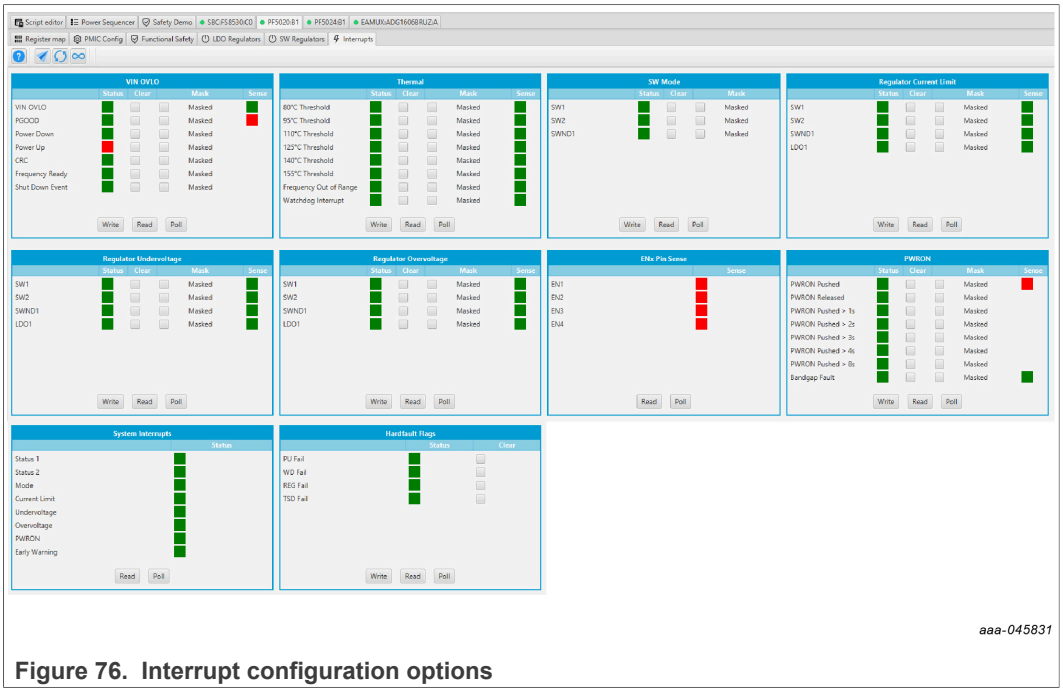


Figure 76. Interrupt configuration options

6.6 PF5024 Tabs

This section describes the tabs specific to the PF5024.

6.6.1 Register map tab

See Section 4.6 for details on using this tab.

6.6.2 PMIC config tab

The PMIC config tab provides configuration options for general device operation. It adapts to the selected device mode and enables only accessible options, the rest are grayed out.

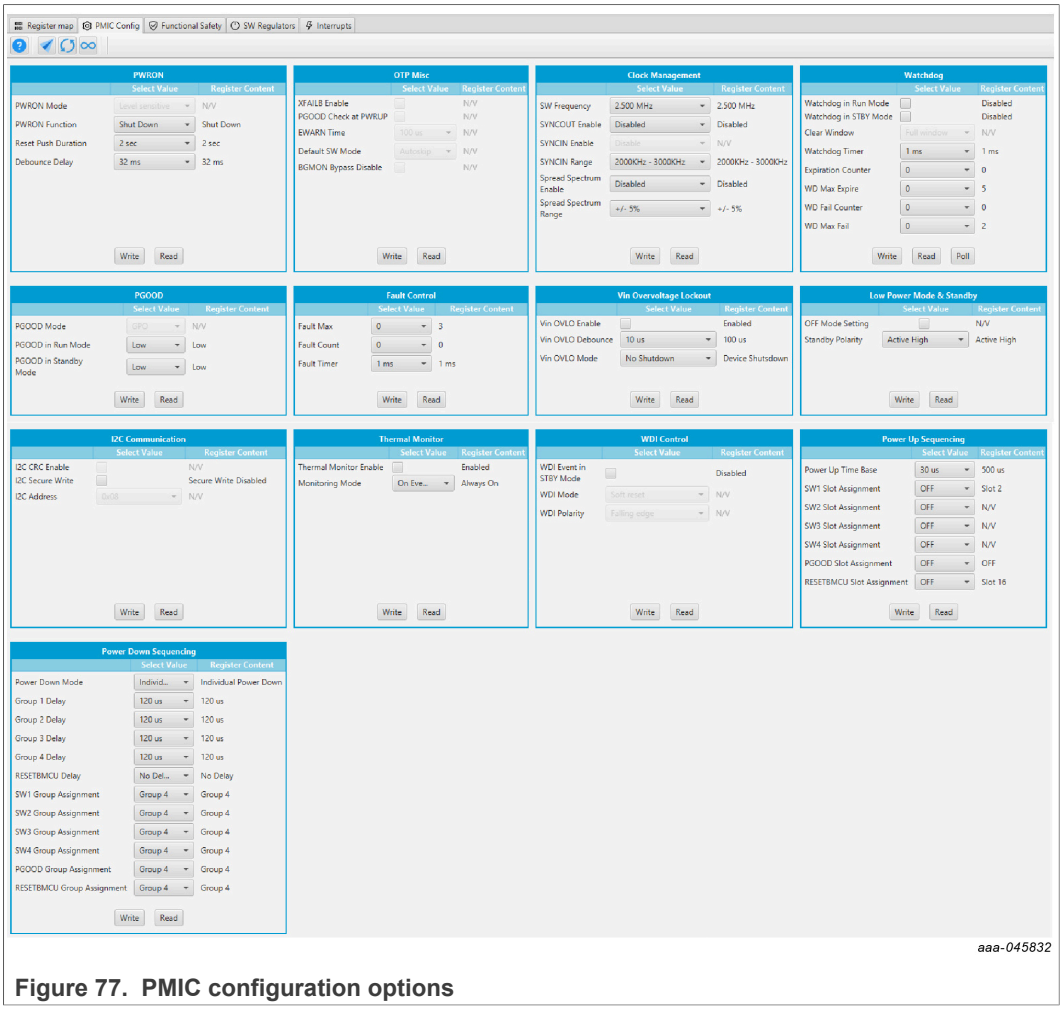


Figure 77. PMIC configuration options

6.6.3 Functional safety tab

The functional safety tab provides configuration options for functional safety features. It adapts to the selected device mode and enables only accessible options, the rest are grayed out.

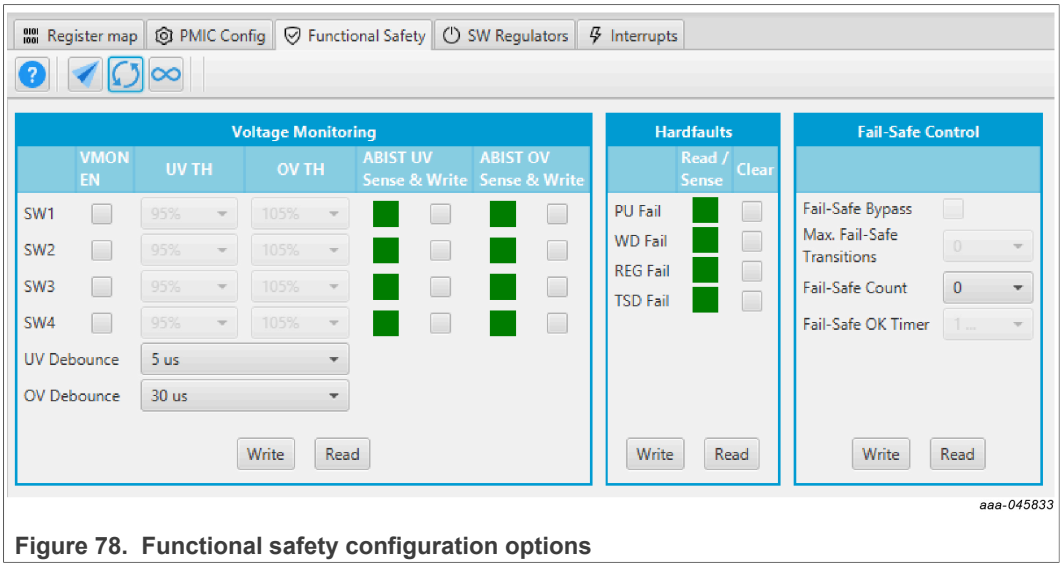


Figure 78. Functional safety configuration options

6.6.4 SW regulators tab

The SW regulators tab provides configuration options for SW regulators. It adapts to the selected device mode and enables only accessible options, the rest are grayed out.

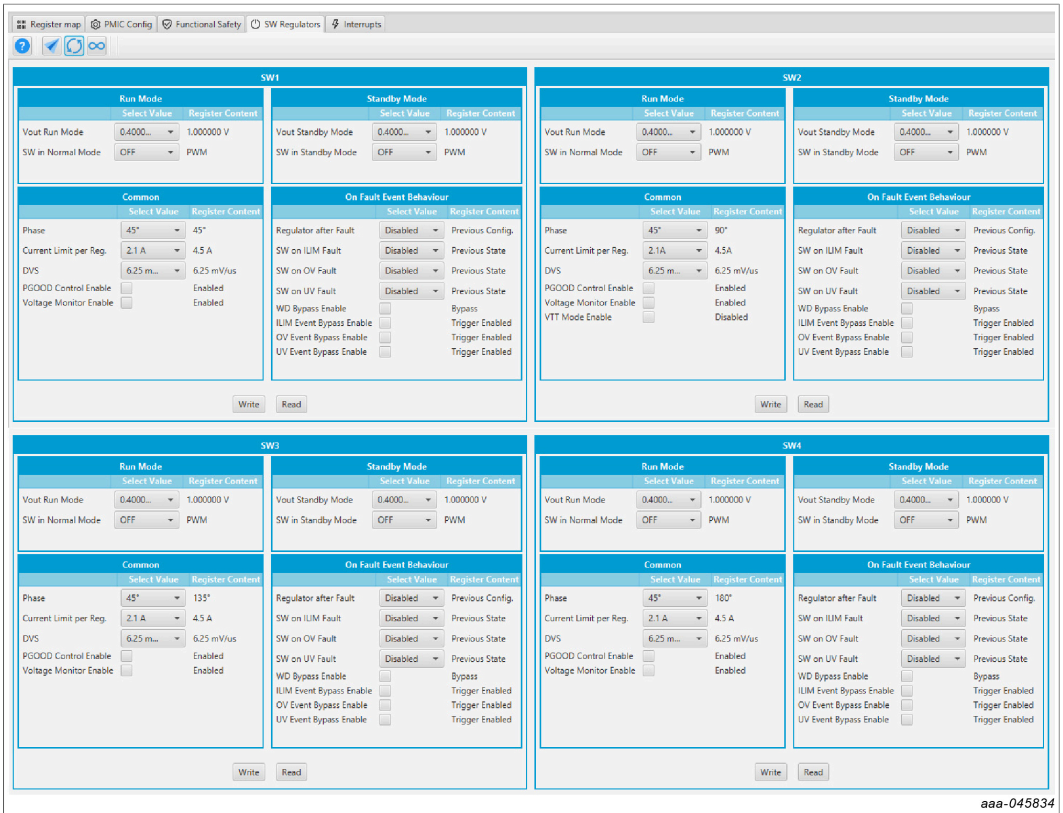
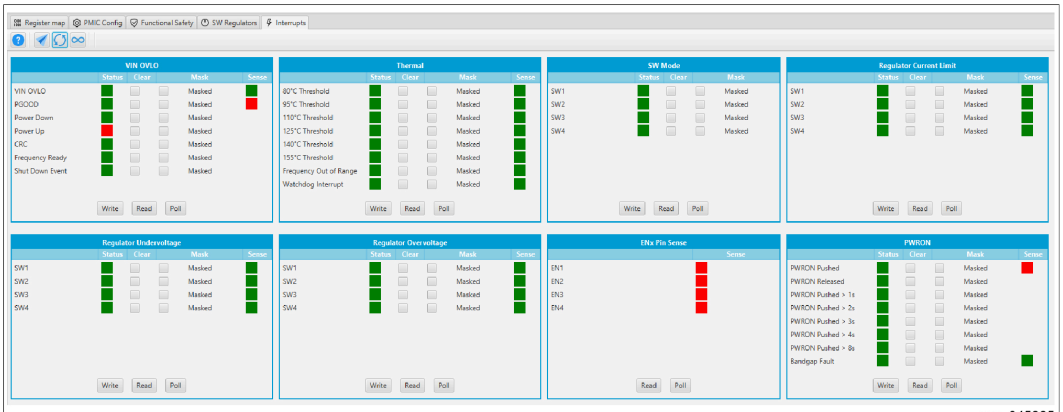


Figure 79. SW regulators configuration options

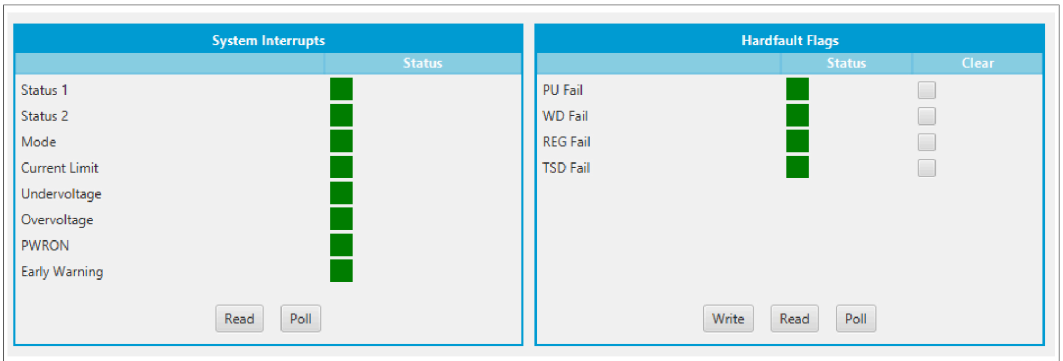
6.6.5 Interrupts tab

The interrupts tab provides configuration options and overview for device events. These may be optionally cleared or masked. It can be read out one-time or periodically with polling.



aaa-045835

Figure 80. Interrupt configuration options



aaa-045836

Figure 81. System and hard fault flag configuration options

7 BYLMP GUI specific tabs and features

This section describes the tabs and features specific to the BYLMP EVB[1].

7.1 Multi-PMIC / power sequencer

The multi-PMIC / power sequencer tab allows users to evaluate the power up/down sequences of FS85 and PF502x on the BYLMP EVB[1]. Both sequencing events are visualized in separate charts for FS85 and PF502x. It also provides a simple table with comparison of set and real-time slots for configurable regulators. Additionally, users are allowed to inject OV/UV/OC faults on selected regulators and evaluate triggered power down sequence.

aaa-045837

Figure 82. Power sequencer

Power Up/Down Control enables users to regularly power up/down FS85/PF502x. Both actions trigger the signal state capture routine on the MCU. The results are then sent back to the GUI for post-processing (charts, set/real slot comparison).

Fault Injection enables users to inject an OV/UV/OC fault either on the FS85 LDO1 or on the PF5020's LDO1 regulator. This action might lead to a device power down event (depends on device configuration). The sequence is captured and processed similarly as above.

Signal Table enables users to read and set slots for configurable regulators and clear current results.

Notice: *If GUI runs fast polling for pin or register values, it might influence accuracy of results due to MCU being clogged with higher priority requests. Solution is to not use polling when working with the power sequencer.*

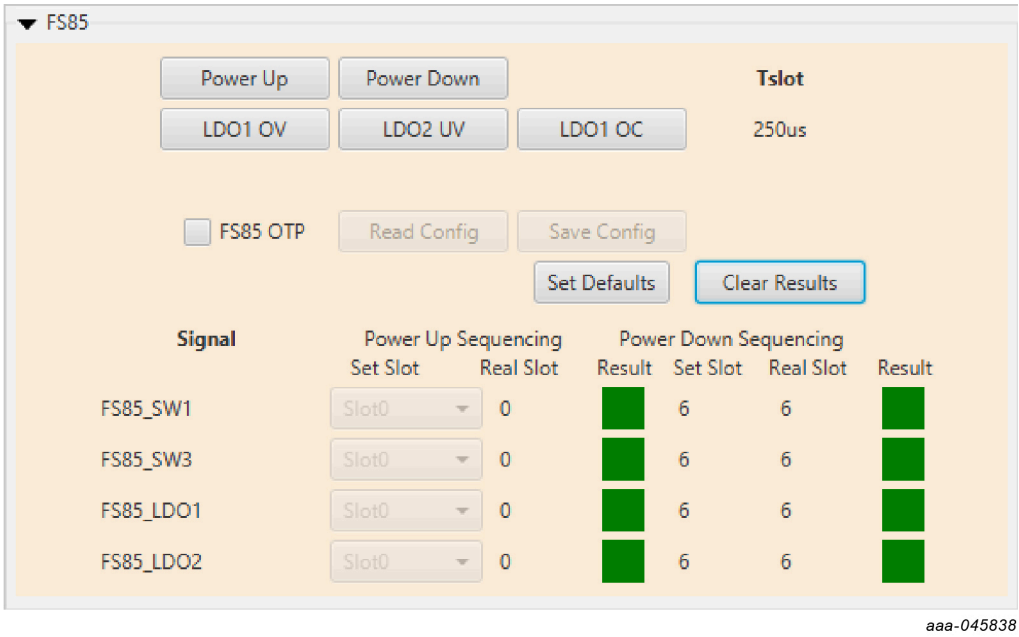


Figure 83. FS85 power sequencer control

In order to read out or modify FS85 OTP values used for slot selection, users must perform the following steps:

1. If powered up, power down the PF502x, otherwise the PF502x prevents the FS85 from powering down.
2. Ensure the SW1 (WAKE1) on the board is off, otherwise it prevents the FS85 from powering down.
3. Power down the FS85 using the dedicated button.
4. Set jumper J7 on the board to position 2-3 so the GUI can operate the DBG_CTRL pin.
5. Check the FS85 OTP. The GUI instructs the FS85 to transition into debug/test mode.
6. Wait for completion. Once completed, the GUI enables the signal slot selection combo boxes. **If transition to debug/test mode fails for any reason, the combo boxes remain disabled (grayed out).**
7. Modify the signal slot selection as needed.
   a. Read the configuration to view what the default values are.
   b. Select the custom slot signal configuration and save the configuration.
   c. Read the configuration to confirm that the choices were applied as expected.
8. Uncheck the FS85 OTP to leave debug/test mode.
9. Power up the FS85 with custom signal slot configuration.

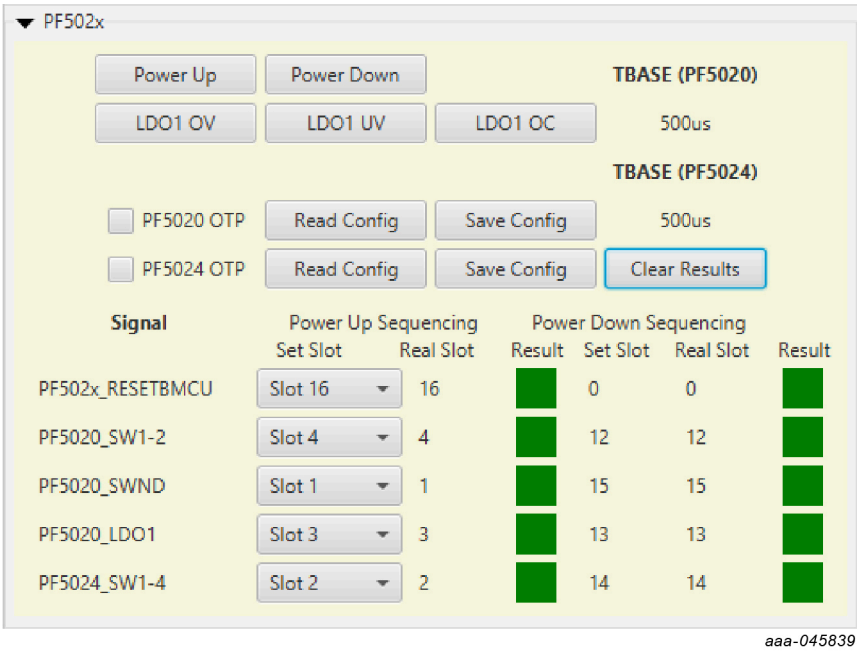


Figure 84. PF502x Power Sequencer Control

The PF502x allows users to read out and modify the signal slot configuration in both normal mode and TBB (try before buy) mode.

- In normal mode, the PF502x is already powered up, therefore custom changes are only applied to the power down sequence and then the provided configuration is lost.
- In TBB mode, PF502x is not powered up yet, therefore custom changes are applied to power up sequence and possibly also to the power down sequence (if not further modified).

In order to read out or modify the PF502x OTP values used for slot selection, perform the following steps:

1. Power down the PF502x with the dedicated button.
2. Check the PF502x OTP. The GUI instructs the PF502x to transition into TBB (try before buy) mode.
3. Wait for completion.
4. Modify the signal slot selection as needed.
   a. Read the configuration to view what the default values are.
   b. Select the custom slot signal configuration and save the configuration.
   c. Read the configuration to confirm that the choices were applied as expected.
5. Uncheck the PF502x OTP to leave TBB mode.
6. Power up the PF502x with the custom signal slot configuration.

Note: *Checking the PF5020 OTP and PF5024 OTP simultaneously is not possible because in TBB mode the PF502x device uses the default I^2C address and would create ambiguity on I^2C bus.*

Note: *If the RESETBMCU slot configuration is changed, the RESETBMCU slot configuration for both the PF5020 and the PF5024 device should also be changed. This signal is shared with both chips and different configurations may cause unexpected behavior.*

7.2 Multi-PMIC / safety demo

The safety demo tab shows the BYLMP Multi-PMIC power solution system block diagram. It facilitates understanding of BYLMP system architecture. Users can check the status of all signals and power rails, operate WAKE2 and PWRON virtual switches, change SBC working mode, release the FS0B, and inject faults on selected regulators.

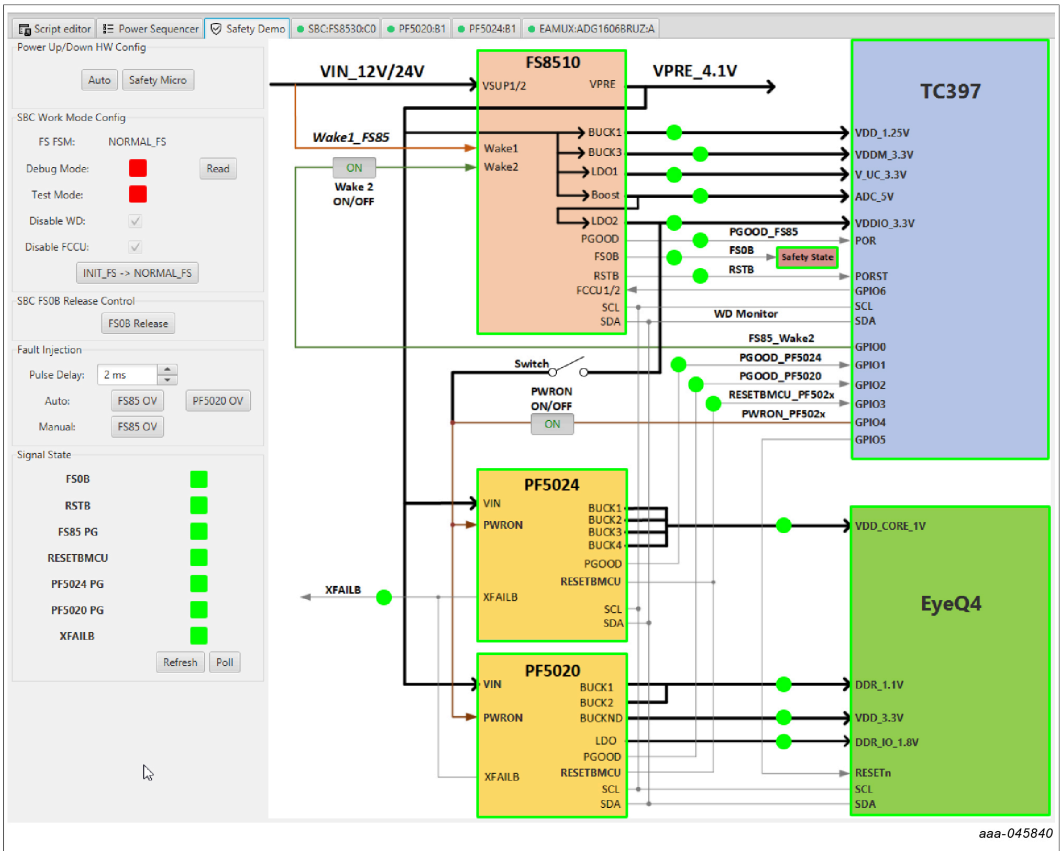


Figure 85. Safety demo

Note: *This architecture can be used by multiple MCUs with different pin names, therefore the TC397, and EyeQ4 box names and all related signal names can manually be changed. The GUI remembers the change and will use it for the next run.*

Power Up/Down HW Config box enables users to control the state of SBC and PMICs. Selection between Auto and Safety Micro should correspond to configuration of jumpers **J20** and **J22** on the BYLMP EVB[1].

- **Auto** corresponds to the setup when the PF502x PMICs power up/down is triggered by the FS85 LDO2.
- **Safety Micro** corresponds to the setup when the PF502x PMICs power up/down is triggered by the KL25Z output pin (PWRON).

SBC Work Mode Config box informs about the current the SBC work mode and enables users to initialize the SBC. Both the mandatory watchdog and FCCU are disabled during this action.

- **FS FSM** stands for fail-safe finite state machine and can enter one of following states (*INIT_FS, WAIT_ABIST2, ABIST2, ASSERT_FS0B and NORMAL_FS*).

- **Debug Mode** indicator reflects whether the SBC entered Debug mode during startup (DBG = 1).
- **Test Mode** indicator reflects whether the SBC entered test mode, which is mandatory for access to mirror registers.

These values are taken from safety/FS_STATES register, FSM_STATE, TM_ACTIVE and DBG_MODE bit groups.

INIT_FS -> NORMAL_FS button performs the following:

1. Configure the WD window to be disabled.
2. Disable the FCCU.
3. Set VMON3 to LDO1.
4. Initial good WD refresh to leave INIT_FS state.
5. Leave Debug mode.

SBC power up/down can be operated by both WAKE1 and WAKE2 pins.

In case of **WAKE1**, when turned ON via SW1 on BYLMP EVB[1], DBG is also set to 1 during SBC startup. This leads to a scenario, when SBC ends up in INIT_FS state, but w/o constraints on SBC initialization during 256 ms window. In this situation, the user can provide a custom fail-safe configuration which is accessible only in this phase. Once SBC is properly configured (or the defaults are used), later on the user can manually initiate the transition to NORMAL_FS.

In case of **WAKE2** (assume that WAKE1 = 0), when turned ON via MCU GPIO, DBG remains at 0 during SBC startup. This leads to the scenario when SBC also ends up in INIT_FS state, but with a constraint on SBC initialization during 256 ms window. Since it would not be possible for the user to meet this timing constraint manually, the MCU handles SBC initialization. It is triggered when WAKE2 and RSTB rising edges are detected. This SBC auto-init feature is available only for WAKE2 and can be useful when RSTB is asserted low due to expiration of the fault error counter (fault injection, wrong watchdog refresh, and so forth).

SBC FS0B Release Control box enables the user to execute the FS0B release routine when the FS0B has been asserted low as a configured impact for a fault event.

FS0B release button performs the following:

1. Clears the fault error counter.
2. Reads the WD seed, reverses it and writes it back.

Basically the **INIT_FS -> NORMAL_FS** represents lines 1-4 and **FS0B release** represents lines 5-11 in Figure 86.

FS0B release sequence after POR from Debug mode			
Simple WD		Challenger WD	
Register Name	Data	Register Name	Data
FS_WD_WINDOW	0x0200	FS_WD_WINDOW	0x0200
FS_NOT_WD_WINDOW	0xFDFF	FS_NOT_WD_WINDOW	0xFDFF
FS_WD_ANSWER	0x5AB2	FS_WD_ANSWER	0xA54D
FS_STATES	0x4000	FS_STATES	0x4000
FS_WD_ANSWER	0x5AB2	FS_WD_ANSWER	0x4A9A
FS_WD_ANSWER	0x5AB2	FS_WD_ANSWER	0x9535
FS_WD_ANSWER	0x5AB2	FS_WD_ANSWER	0x2A6A
FS_WD_ANSWER	0x5AB2	FS_WD_ANSWER	0x54D4
FS_WD_ANSWER	0x5AB2	FS_WD_ANSWER	0xA9A9
FS_WD_ANSWER	0x5AB2	FS_WD_ANSWER	0x5353
FS_RELEASE_FS0B	0xB2A5	FS_RELEASE_FS0B	0x6565

aaa-045841

Figure 86. FS0B release sequence after POR from Debug mode

Fault Injection box enables the user to inject OV fault either on the FS85 LDO1 or the PF5020's LDO1 regulator. In the case of the FS85, each fault injection increments the fault error counter. Depending on the configuration, it might assert FS0B and RSTB. Once the fault error counter expires, SBC is shut down. This leads to a change in the signal states represented by indicators. If SBC was operated via WAKE1, it ends up in 4 s auto-retry loop (or shutdown) due to DBG=0, therefore missing the mandatory 256 ms window for SBC initialization. If the SBC was operated via WAKE2, the auto-init feature kicks in and reinitializes the SBC.

- **Pulse Delay** can be used for customization of the fault injection pulse length on given power rail.
- **Auto** triggers the fault injection pulse with a specified delay.
- **Manual** triggers the fault injection until the user releases the button.

Signal State box reflects the state of all signals used in this setup. It can be manually refreshed or polled each second.

Conditions used for box colors.

- The FS85 box is set to green if VPRE went up, otherwise it remains gray.
- The TC397 box is set to green if RSTB went up, otherwise it remains gray.
- The PF5020 box is set to green if PWRON, VPRE, PF5020_PGOOD, and RESETBMCU went up, otherwise it remains gray.
- The PF5024 box is set to green if PWRON, VPRE, PF5024_PGOOD, and RESETBMCU went up, otherwise it remains gray.
- The EyeQ4 box is set to green if PGOOD and RESETBMCU went up, otherwise it remains gray.
- The FS0b box is set to green if FS0B went up, otherwise it remains red.

7.3 FS85 (SBC) / watchdog

The watchdog tab enables users to evaluate the SBC watchdog, which is considered to be an essential safety feature. It can be operated in both work modes of the SBC, debug and normal.

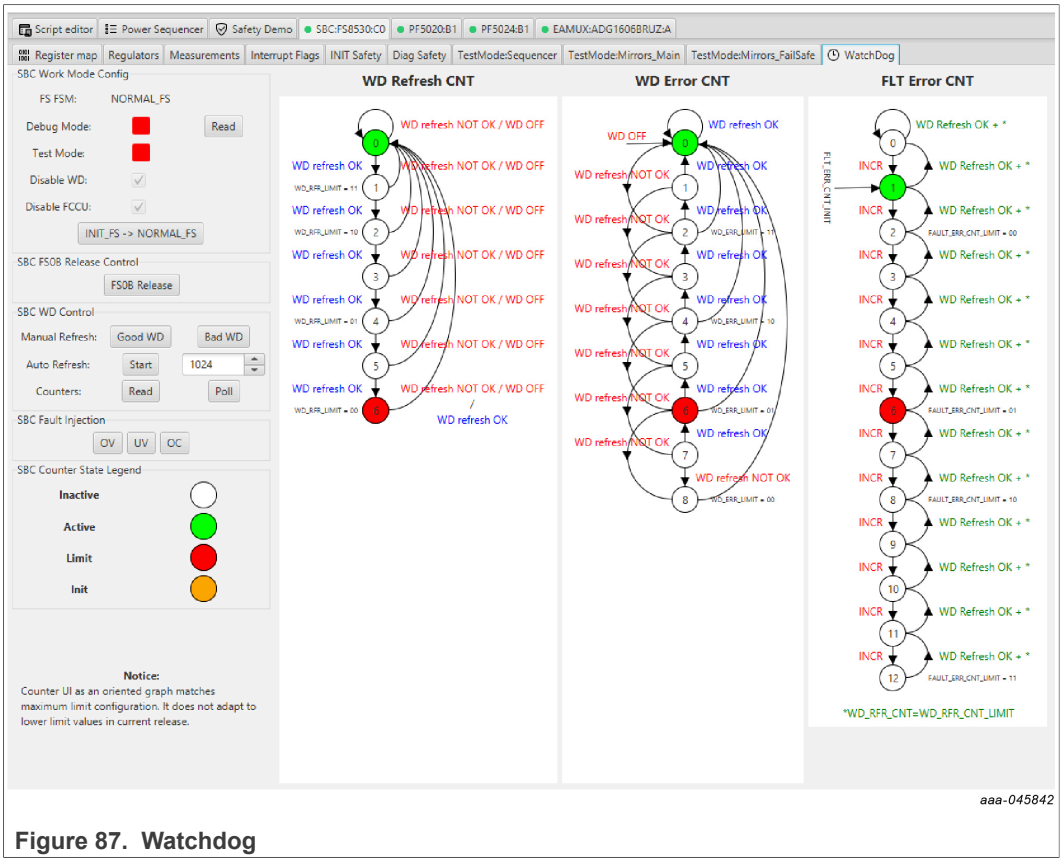


Figure 87. Watchdog

SBC Work Mode Config box informs the user about the current work mode of the SBC and enables the user to initialize the SBC. Both mandatory watchdog and FCCU are disabled during this action. See Section 7.2 for a full description of this box.

SBC FS0B Release Control box enables users to execute the FS0B release routine when the FS0B has been asserted low as a configured impact for a fault event. See Section 7.2 for a full description of this box.

SBC WD Control box enables the user to operate the watchdog either manually or automatically via MCU timers.

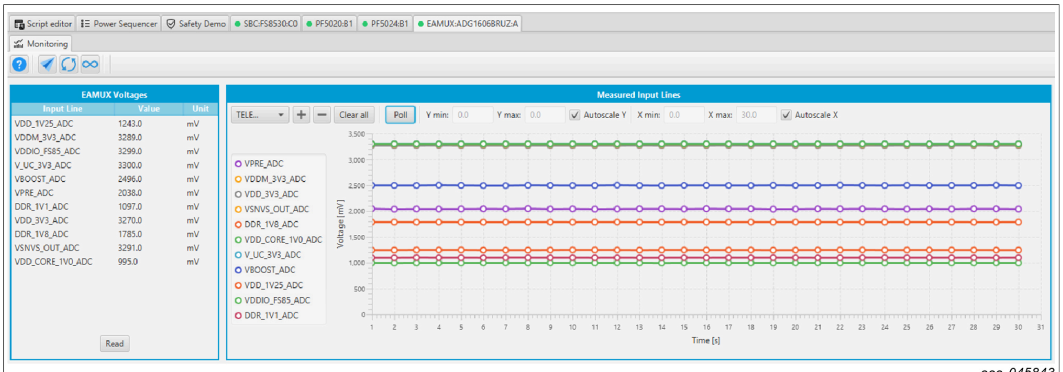
- **Manual Refresh** sends single watchdog good/bad refresh command, which results in WD refresh/error counter change.
- **Auto Refresh** enables/disables the timer on the MCU, which periodically executes the routine sending a good watchdog refresh.
- **Counters** can be read only once or polled with a period matching the auto refresh. The minimum is set to 100 ms due to GUI ⇔ MCU communication speed limits. The maximum is set to 1000 ms and the value can be adjusted with 5o ms steps. These values are taken from the Write_INIT_Safety/FS_I_WD_CFG and FS_I_FSSM registers.

SBC Fault Injection box enables the user to inject OV, UV, or OC fault on the FS85's LDO1 regulator. Each fault injection increments the fault error counter, and depending on the configuration, might assert FS0B and RSTB. Once the fault error counter expires, the SBC is shut down. If the SBC was operated via WAKE1, it ends up in 4 s auto-retry loop (or shutdown) due to DBG=0, therefore missing the mandatory 256 ms window for SBC initialization. If the SBC was operated via WAKE2, the auto-init feature kicks in and reinitializes the SBC.

SBC Counter State Legend describes the color code used in state machines representing WD refresh, WD error, and fault error counters.

7.4 ADG1606BRUZ (EAMUX) / Monitoring

The ADG1606BRUZ (EAMUX) / monitoring tab provides users with all possible measured inputs on EAMUX in a visual format. Refer to the BYLMP EVB[1] schematics for details.



aaa-045843

Figure 88. EAMUX monitoring tab

8 References

[1] **BYLMP EVB** — *HW Product Summary Page*
TBD

[2] **BYLMP GUI SW** — *SW Product Summary Page*
https://www.nxp.com/design/analog-expert-software-and-tools/flexgui-software-tool-for-evaluation-of-reference-design-kits:FLEXGUI-SW

[3] **PEmicro's GDB server** — *PEmicro's GDB server for Arm download page*
http://www.pemicro.com/downloads/download_file.cfm?download_id=412

[4] **GNU Arm GDB** — GNU Arm GDB download page
https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads

[5] **FS85** — *HW Product Summary Page*
https://www.nxp.com/products/power-management/pmics-and-sbcs/safety-sbcs/safety-system-basis-chip-for-s32-microcontrollers-fit-for-asil-d:FS8500

[6] **PF5020** — *HW Product Summary Page*
https://www.nxp.com/products/power-management/pmics-and-sbcs/pmics/multi-channel-5-pmic-for-automotive-applications-4-high-power-and-1-low-power-fit-for-asil-b-safety-level:PF5020

[7] **PF5024** — *HW Product Summary Page*
https://www.nxp.com/products/power-management/pmics-and-sbcs/pmics/multi-channel-4-pmic-for-automotive-applications-4-high-power-fit-for-asil-b-safety-level:PF5024

9 Legal information

9.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

9.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

Suitability for use in automotive and/or industrial applications — This NXP product has been qualified for use in automotive and/or industrial applications. It has been developed in accordance with ISO 26262 respectively IEC 61508, and has been ASIL- respectively SIL-classified accordingly. If this product is used by customer in the development of, or for incorporation into, products or services (a) used in safety critical applications or (b) in which failure could lead to death, personal injury, or severe physical or environmental damage (such products and services hereinafter referred to as "Critical Applications"), then customer makes the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, safety, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. As such, customer assumes all risk related to use of any products in Critical Applications and NXP and its suppliers shall not be liable for any such use by customer. Accordingly, customer will indemnify and hold NXP harmless from any claims, liabilities, damages and associated costs and expenses (including attorneys' fees) that NXP may incur related to customer's incorporation of any product in a Critical Application.

9.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Tables

Figures

Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 1 April 2022
Document identifier: UG10029