

S32 Configuration Tools 1.7 Update 8 Release Notes

Contents

1.	Read Me First.....	2
1.1	Installation.....	2
1.2	Problem Reporting Instructions.....	3
2	What's New	3
2.1	New Features.....	3
2.2	New Regressions	4
3	Release Description.....	4
3.1	Pins Tool.....	4
3.2	Clocks Tool.....	9
3.3	Peripherals Tool	13
3.4	Device Configuration Data Tool	25
3.5	Image Vector Table Tool	29
3.6	QuadSPI Tool	45
3.7	DDR Tool	53
3.8	eFuse Tool.....	78
3.9	GTM Tool.....	83
3.10	S32 Design Studio Integration	87
3.10.1	Open S32 Configuration Tools	87
3.10.2	Add SDK drivers to project.....	89
4	Known limitations	91

1. Read Me First

S32 Configuration Tools product is composed of a suite of tools for configuring NXP processors and generating initialization code. This support consists of the following tools:

- **Pins Tool**
- **Clocks Tool**
- **Peripherals Tool**
- **Device Configuration Data Tool**
- **Image Vector Table Tool**
- **QuadSPI Tool**
- **DDR Tool**
- **eFUSE Tool**
- **GTM Tool**

1.1 Installation

S32 Configuration Tools - Version 1.7 Update 8 is integrated into S32 Design Studio 3.5 Update 12 which is based on Eclipse Version 2021.06.

One can access S32 Design Studio release announcements and/or [S32 Design Studio page](#) on nxp.com to find out latest releases and more info about the compatible S32 Software Development Kit / S32 Real Time Drivers packages, including their download location.

S32 Configuration Tools supported in this release:

- IVT Tool – Intuitive Image Vector Table Definition
- DCD Tool – Quick Peripherals Configuration
- QuadSPI Tool – Configuration of the QSPI parameters for boot sequence
- DDR Tool – LPDDR4, DDR3L training, DDR initialization and stress test
- eFUSE Tool – Graphical fuse configuration
- GTM Tool – Configuration and initialization of General Timer Module

There are additional S32 Configurations Tools supported in this release and enabled with S32 Software Development Kit / S32 Real Time Drivers NPI data:

- Pins Tool – Graphical tool to easy assign I/O functions to different pins
- Clock Tool – Graphical clock tree for easy configuration of complicated device clocks
- Peripherals Tool – configures the device initialization and software drivers of an SDK

S32 Configuration Tools product was verified on:

- Windows 10, 64bit

1.2 Problem Reporting Instructions

Problems found in this release, or any suggestions of improvement should be posted to:

- Internal users: [S32 Configuration Tools Jira project](#)
- External users: [S32 Design Studio Community](#)

2 What's New

S32 Configuration Tools 1.7 Update 8 release brings stable framework with important additions in the DDR, GTM and IVT tools. For all other supported tools, maintenance and bug fixes have been done.

2.1 New Features

[DDR] Enable Inline ECC support for LPDDR5

[DDR] Enable multiple frequencies for LPDDR5

[GTM] Interactive diagram view

[GTM] Support for configuring general Clusters clock dividers.

[GTM] Support for configuring CCM (Cluster Clock Management)

[GTM] Support for configuring TBU(Time Base Unit)

[GTM] Support for configuring TIM (Timer Input Module)

[GTM] Added Help documentation for all UI options.

[GTM] Updated templates of generated configuration files for CMU, CCM, TBU, TIM, ATOM and TOM.

[IVT] Added option to keep the input binary files untouched on CLI mode, available on a limited set of processors

[IVT] Added option to perform auto align during the blob export process

[IVT] Allow the import of adkp files that contain the hex prefix

[IVT] Added a reset button the Authentication Tag Generation structure

[DCD] Added the option to validate the DCD configuration in CLI mode

2.2 New Regressions

[Regression][IVT] For some processors, import blob might now work when importing large blob files (around 1GB)

3 Release Description

Next sections offer an insight into the most significant capabilities of all tools.

3.1 Pins Tool

Pins Tool displays, inspects, modifies any aspect of pins configuration and muxing of the device. Pins routing can be done in multiple views:

- Pins view
- Peripheral Signals view
- Package view
- Routed Pins

The screenshot displays the S32 Design Studio interface with the Pins Tool open. The main window shows the 'Package [Pins Bottom]' view, which is a grid representing the physical package pins. The left sidebar shows a tree of peripheral signals, with 'FTM_1' selected. The right sidebar shows a list of registers, with 'SIUL2_MSCR12' highlighted. At the bottom, the 'Routed Pins' table is visible, showing the configuration for various pins.

#	Peripheral	Signal	Route to	Label	Identifier	Power group	Direction	Output Buf...	Open Drain	Input Buffer	Hysteresis	Output Inv...	S
B10	CAN_0	nd	CAN_0,nd		n/a	VDD_GPL...	Input	Disabled	Disabled	Enabled	CMOS	Disabled	5
AB25	ENET	nc_dv	ENET,nc_dv		n/a	VDD_HV_L...	Input	Disabled	Disabled	Enabled	CMOS	Disabled	5
A9	JTAG	tdi	JTAGtdi		n/a	VDD_GPL...	Input	Disabled	Disabled	Enabled	CMOS	Disabled	2
AC20	FTM_1	chn, 0	FTM_1,chn		n/a	VDD_GPL...	Input	Disabled	Disabled	Enabled	Schmitt	Disabled	5
G11	FTM_1	chn, 3	FTM_1,chn		n/a	VDD_GPL...	Output	Enabled	Disabled	Disabled	CMOS	Disabled	5

WARNINGS: v2: There are warnings in the configuration, generated source code might not be complete or can cause compilation warnings.

Figure 1: Pins Tool

Pins Tool is used for pin routing configuration, validation and code generation, including pin functional/electrical properties, run-time configurations, with the following main features:

- Muxing and pin configuration with consistency checking
- Graphical processor package view
- Multiple configuration blocks/functions
- Easy-to-use device configuration
- Selection of Pins and Peripherals
- Package with IP blocks
- Routed pins with electrical characteristics
- Registers with configured and reset values
- Source code for C/C++ applications
- Automatic pins routing

Automatic routing tries to resolve conflicts, it routes signals on different pins until a valid configuration is reached. The used algorithm is a mixed approach between running maximum matching algorithm in a signals – pins bipartite graph and processing remaining signals with backtracking.

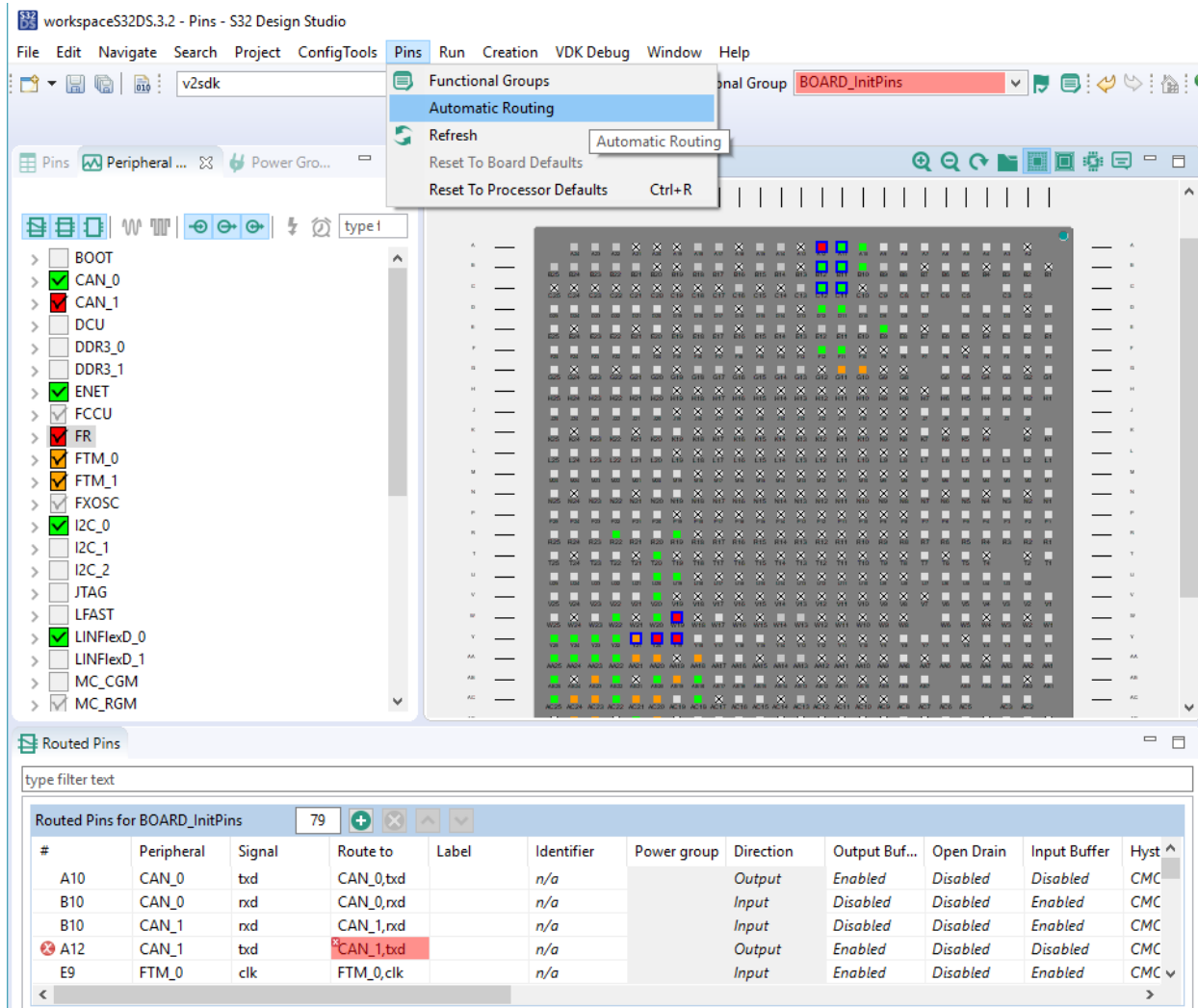


Figure 2: Pins Tool – Automatic routing

Enabled code changes highlighting for all tools. Two styles are currently supported:

- Margin highlighting
- Full line highlighting

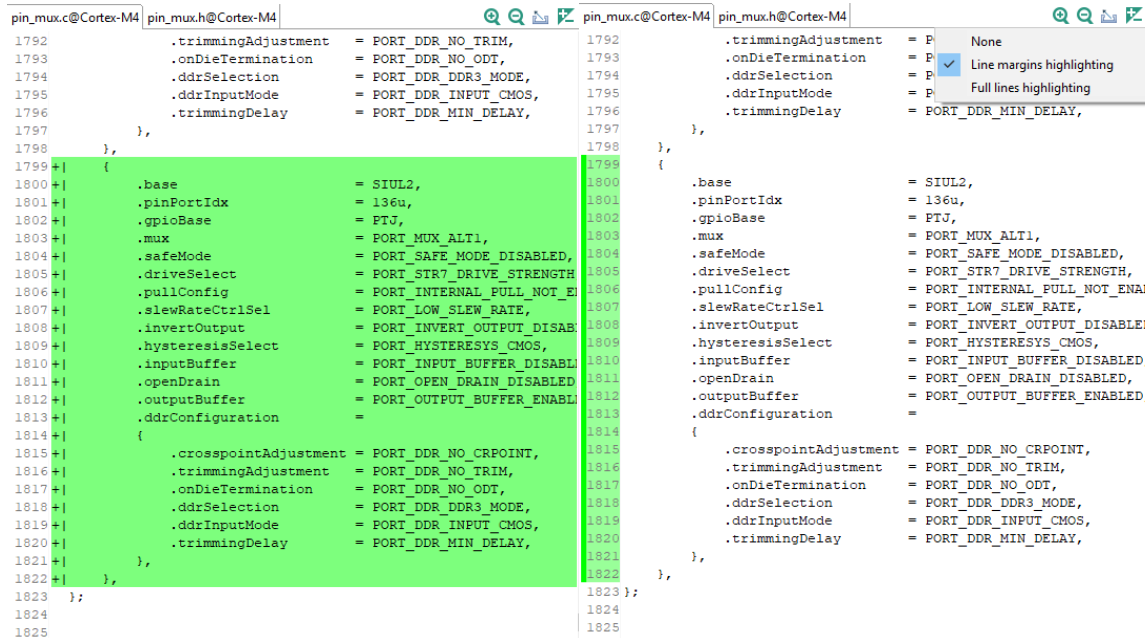


Figure 3: Pins Tool – Code changing highlighting

Export Pins view configuration as HTML and CSV format

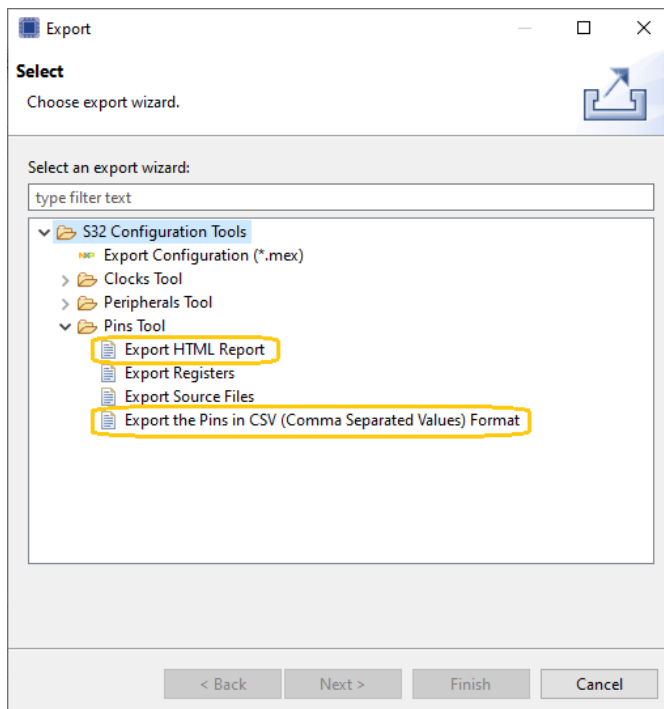


Figure 4: Pins Tool – Export pins

New preference called “Require identifier for Pins (Pins tool)”. This new preference allows user to control the generation of identifier related warnings. With this preference enabled, warnings will be generated for bidirectional signals with no identifier set.

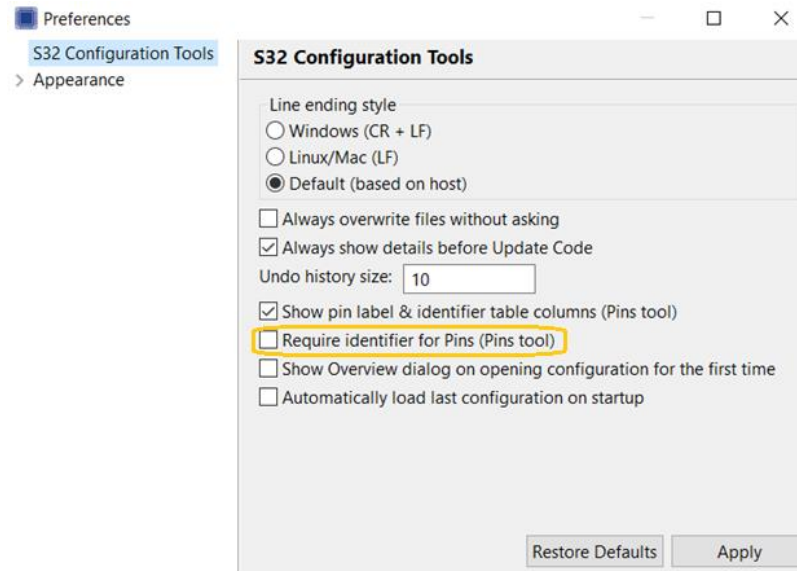


Figure 5: Pins Tool – enable/disable identifier from preference dialog

In case *Identifier* is not used at all, no warnings will be triggered.

Extended Import EPC feature to load pins settings. A mapping file is required in the process, that links the pins from ASR format to S32CT.

This is used by internal SDK/RTD teams while developing components with Config Tools.

Extended Import EPC functionality to support different pin setting mapping files for different derivatives. The link between them is done through the package name.

3.2 Clocks Tool

Clocks tool allows users to easily configure initialization of the system clock – core, system, bus, peripheral clocks – and generate C code with clock initialization functions and configuration structures. The tool validates clock elements and calculates resulting output and clock frequencies.

Advanced settings can be done in Diagram and Details views. Other capabilities are listed next:

- Inspect and modify configuration of elements on clock path from clock source up to the core/peripherals
- Validate clock elements settings and calculate the resulting output clock frequencies
- Generate a configuration code aligned with latest version of header files
- Table view of clock elements with their parameters allowing the user to modify the settings and see the outputs
- Diagram view allows easy navigation and displays important settings and frequencies
- Peripheral Clock view which contains the list of peripherals – if a chain in form of selector, divider and gate is found, an entry is created
- Find clock elements settings that fulfills given requirements for outputs

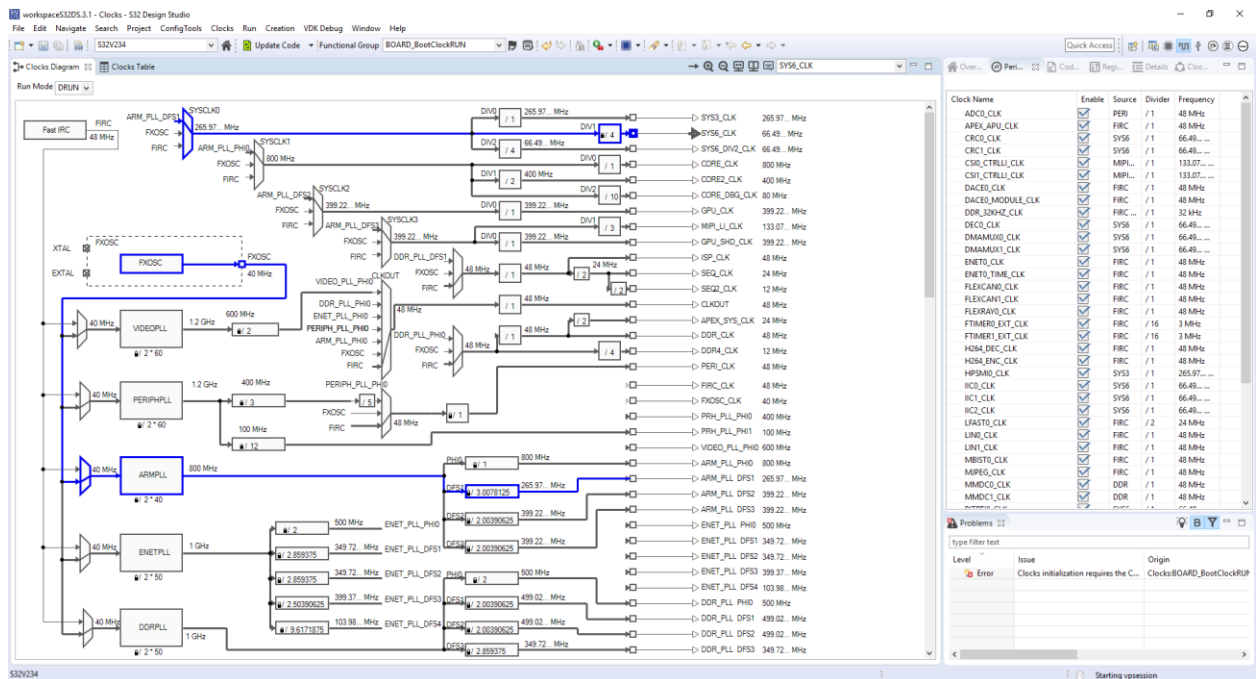


Figure 6: Clocks Tool

- Find near value for clock frequencies

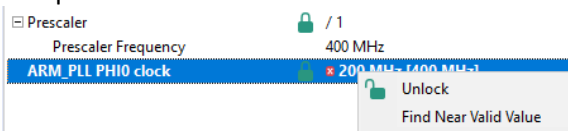


Figure 7: Clocks Tool – Find near value

- Reverse frequency calculation onto fractional elements in clocks path

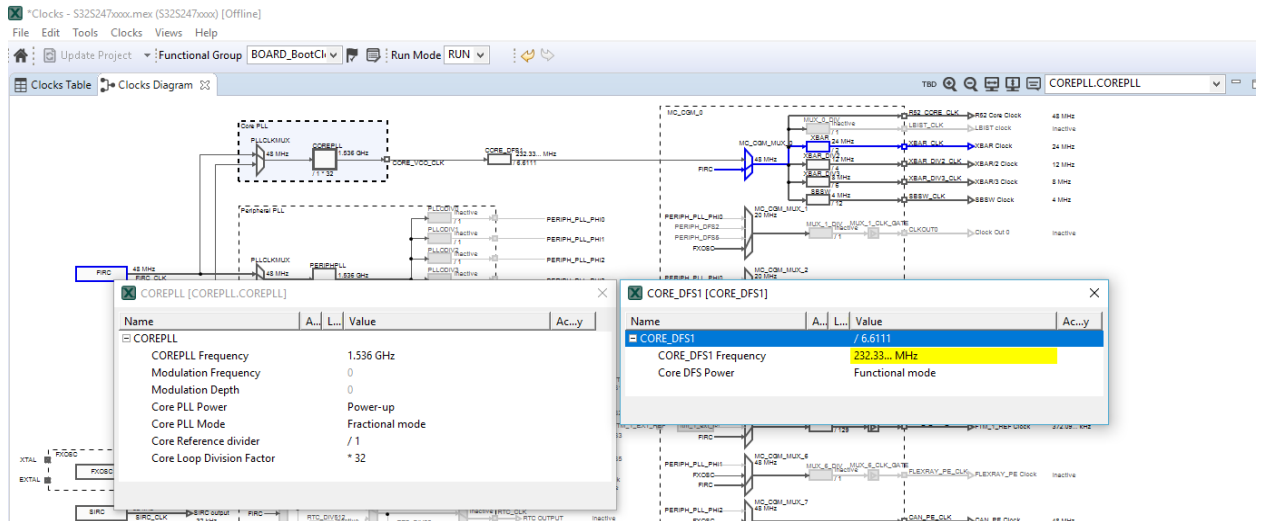


Figure 8: Clocks Tool – Fractional elements

- Allow automatic update of enable state for clock elements based on power modes
- Re-order tabs in Clock perspective, Clocks Diagram and Peripherals Clock views have focus
- Easy to find elements in clock tree, use Find hotkey to search for elements

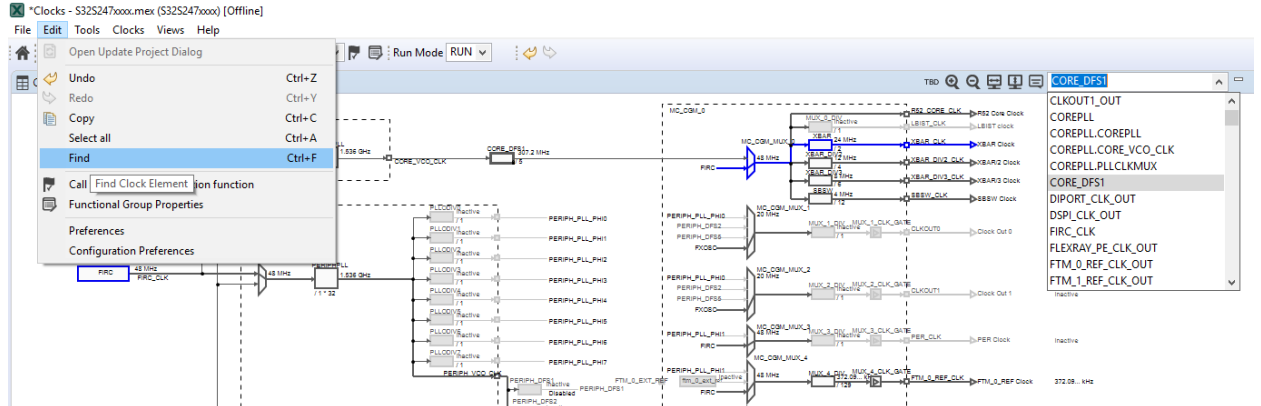


Figure 9: Clocks Tool – find elements

- Export Clock diagram

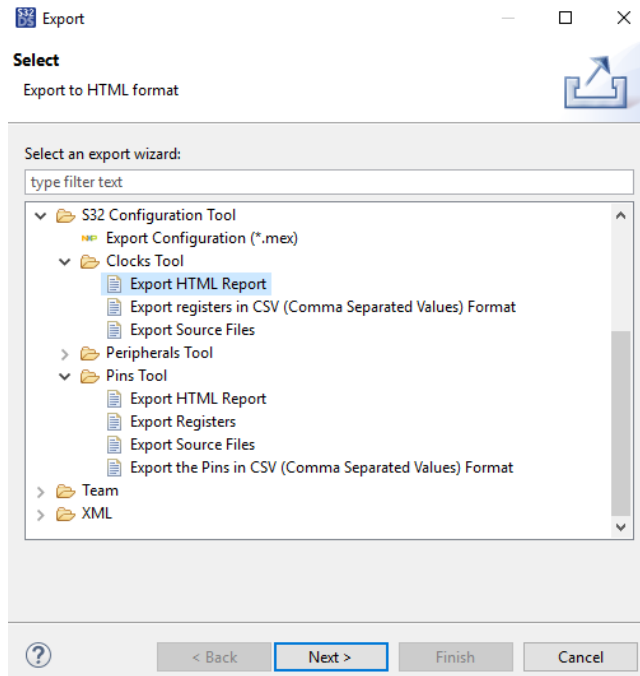


Figure 10: Clocks Tool – Export diagram

- Add support for zoom on cursor in Clocks Diagram
- New “Control” and “DivType” columns in Peripherals Clock view

Overview Peripheral Clock View Code Preview Registers Details Clock Consumers									
Clock Name	Enable	Control	Source	Divider	DivType	Frequency	Monitor	LOW Freq	HIGH Freq
ADC0_CLK	<input checked="" type="checkbox"/>		PER...	/ 1		48 MHz			
ADC1_CLK	<input checked="" type="checkbox"/>		PER...	/ 1		48 MHz			
CLKOUT0_CLK	<input checked="" type="checkbox"/>		FXO...	/ 13		3.07... M...			
CLKOUT1_CLK	<input checked="" type="checkbox"/>		FXO...	/ 1		40 MHz			
CRC0_CLK	<input checked="" type="checkbox"/>		XBA...	/ 1		133.33.....			
CTU0_CLK	<input checked="" type="checkbox"/>		PER...	/ 1		48 MHz			
CTU1_CLK	<input checked="" type="checkbox"/>		PER...	/ 1		48 MHz			
DAPB_CLK	<input checked="" type="checkbox"/>		COR...	/ 4		200 MHz			
DMA0_CLK	<input checked="" type="checkbox"/>		XBA...	/ 1		400 MHz			
DMA1_CLK	<input checked="" type="checkbox"/>		XBA...	/ 1		400 MHz			
DMAMUX0_CLK	<input checked="" type="checkbox"/>		XBA...	/ 1		133.33.....			
DMAMUX1_CLK	<input checked="" type="checkbox"/>		XBA...	/ 1		133.33.....			
DMAMUX2_CLK	<input checked="" type="checkbox"/>		XBA...	/ 1		133.33.....			
DMAMUX3_CLK	<input checked="" type="checkbox"/>		XBA...	/ 1		133.33.....			
DMA_CRC0_CLK	<input checked="" type="checkbox"/>		XBA...	/ 1		400 MHz			
DMA_CRC1_CLK	<input checked="" type="checkbox"/>		XBA...	/ 1		400 MHz			
EIM0_CLK	<input checked="" type="checkbox"/>		A53...	/ 1		4.8 MHz			
EIM1_CLK	<input checked="" type="checkbox"/>		XBA...	/ 1		66.66... ..			
FIM2 CLK	<input checked="" type="checkbox"/>		XRA...	/ 1		66.66			

Figure 11: Clocks Tool – Peripherals Clock view

“Control” column reflects the configuration from output element meaning if the element is or not under driver control, possible values are ENABLED/DISABLED.

“DivType” reflects the configuration element from divider element with possible values COMMON_TRIGGER_UPDATE/IMMEDIATE_TRIGGER_UPDATE.

- Exposed more general clock properties in the UI:

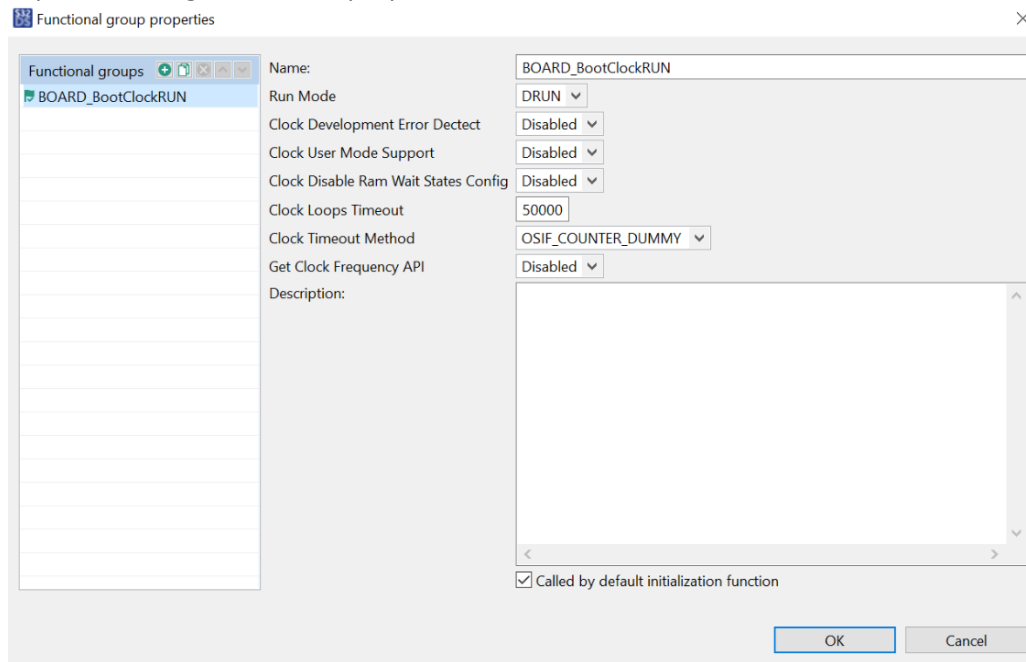


Figure 12: Clocks Tool - general properties

3.3 Peripherals Tool

Peripherals Tool offers support to initialize, configure peripherals and generate code for S32 SDK drivers.

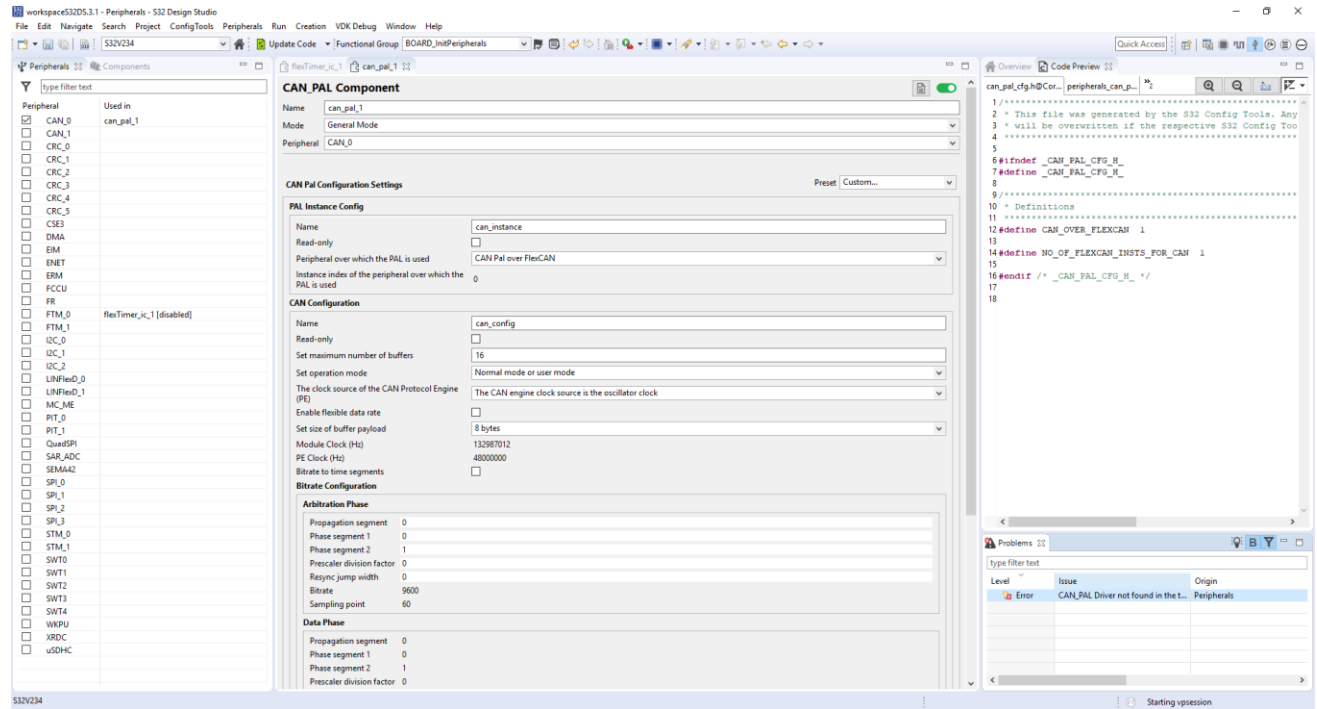


Figure 13: Peripherals Tool

Other capabilities are listed next:

- Configuration and initialization for SDK drivers
- User friendly user interface allowing to inspect and modify settings
- Smart configuration component selection along the SDK drivers used in toolchain project

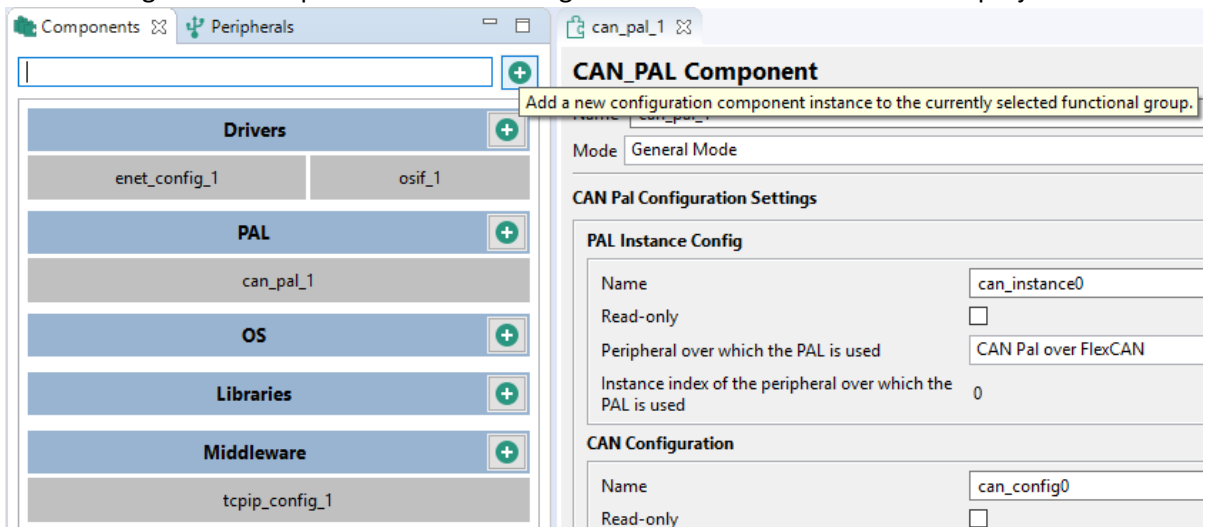


Figure 14: Peripherals Tool – SDK components

- Generation of initialization source code using SDK function calls
- Multiple function groups support for initialization alternatives
- Configuration problems are shown in Problems view and marked with decorators in other views
- Instant validation of basic constraints and problems in configuration

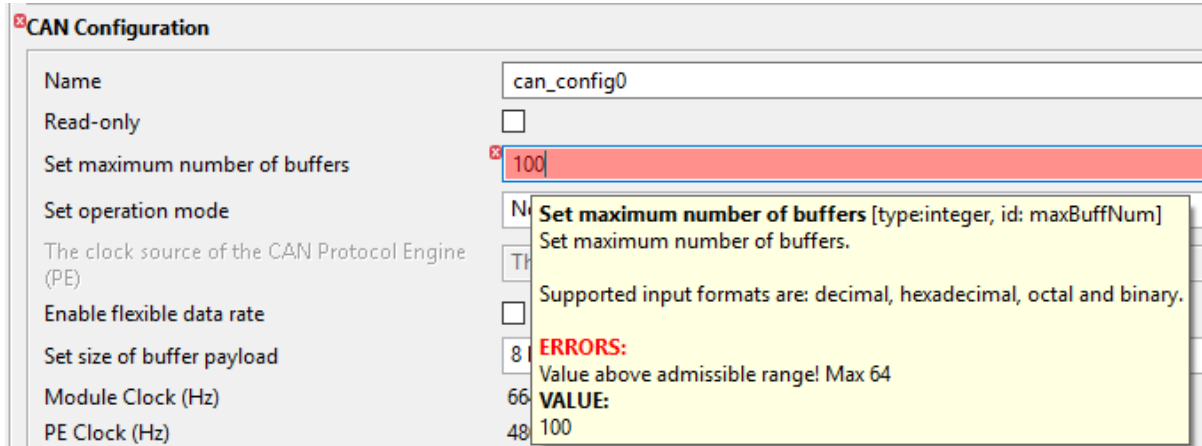


Figure 15: Peripherals Tool – constraints

- Missing drivers are added to project with one click using dedicated option

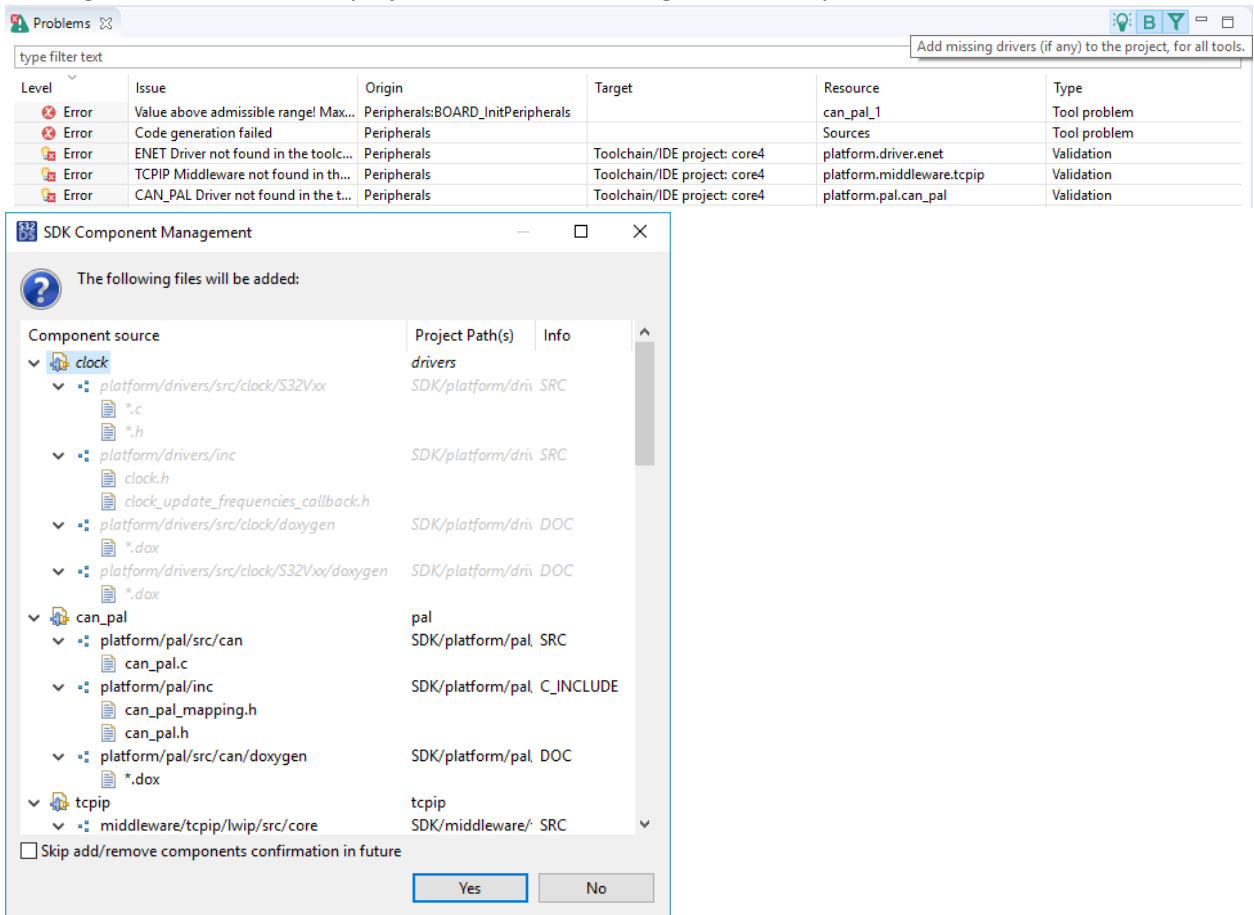


Figure 16: Peripherals Tool – SDK Component Management, add files

- Manage SDK Components in sync with Peripherals view

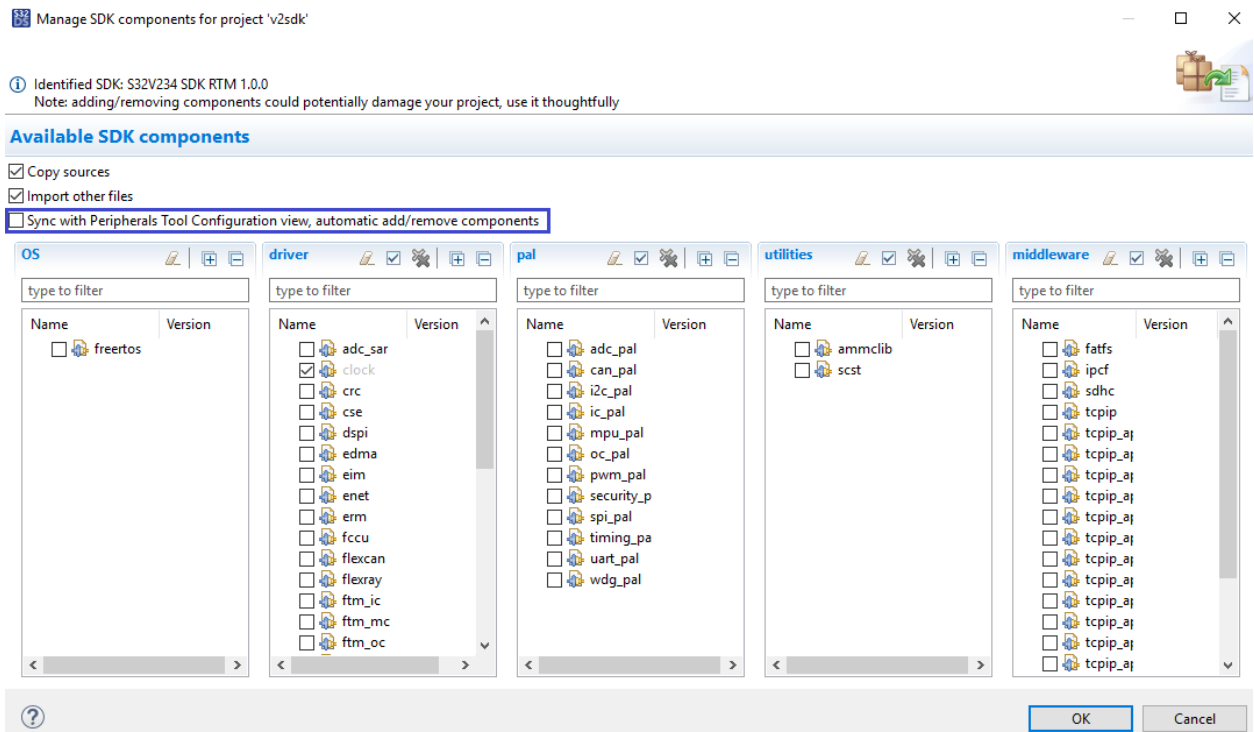


Figure 17: Peripherals Tool – SDK Component Management

- Move generated files to project for all tools

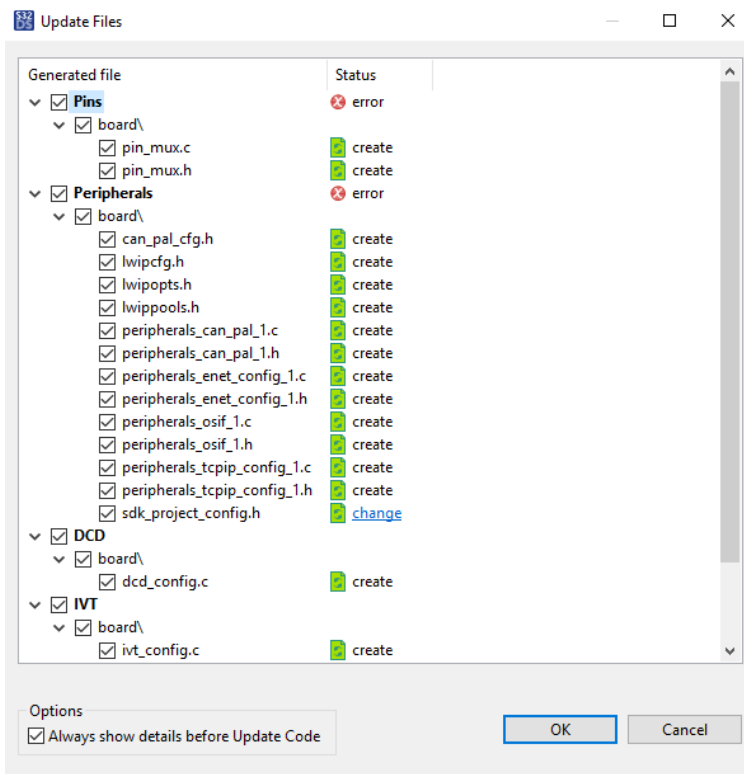


Figure 18: Peripherals Tool – Update Code

- Help mechanism for peripheral components

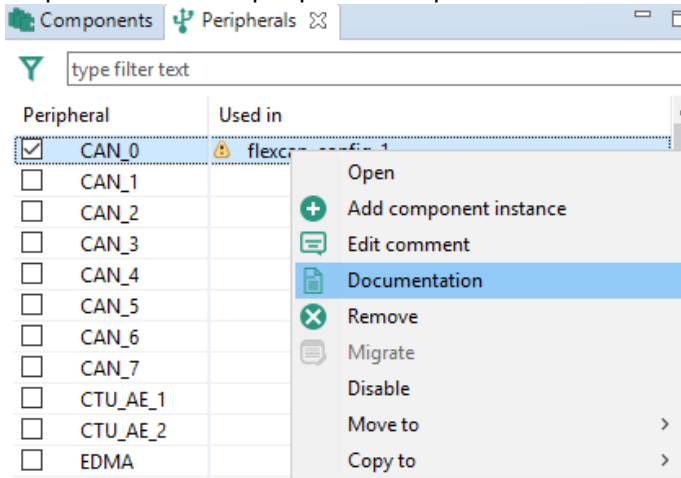


Figure 19: Peripherals Tool – Show documentation

- Show Problem moves focus on the UI control that reported the error
- Various enhancements for better user experience:
 - Collapse/expand large arrays/structures to have a clear view of the entire content of the driver

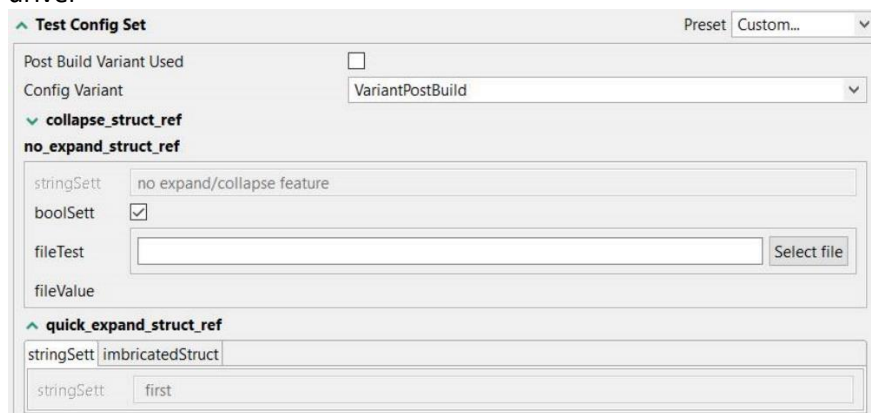


Figure 20: Peripherals Tool – Collapse/expand large structures

- Enable horizontal/vertical scrollbar when size of array items becomes greater than the screen allows

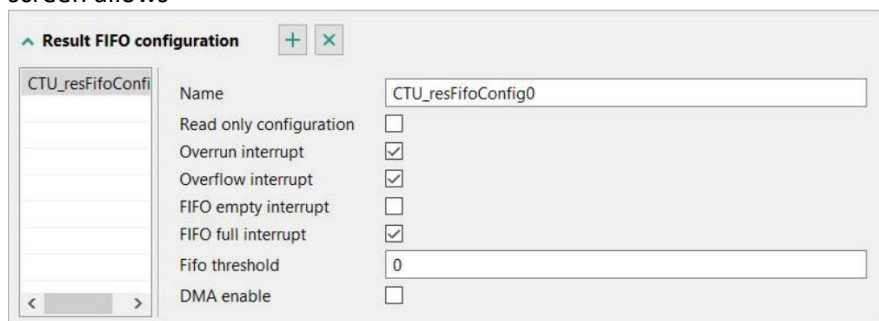


Figure 21: Peripherals Tool – Horizontal scrollbar

- Possibility to insert a string spanning on more lines

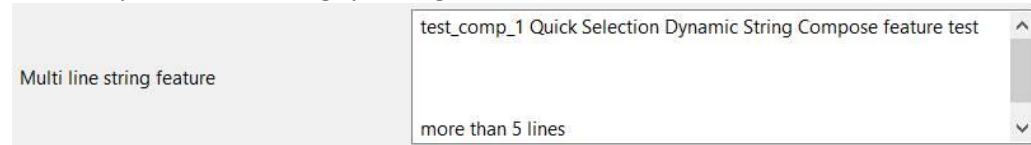


Figure 22: Peripherals Tool – Multi line span

- Implemented possibility to Browse for a file in settings view

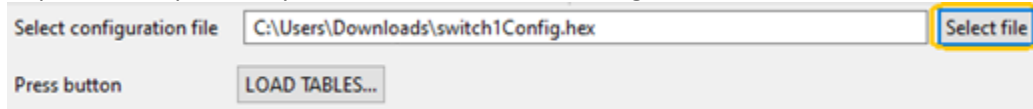


Figure 23: Peripherals Tool - Browse file

- New sub-settings element with the possibility to open pop-up dialog for more in depth configuration:

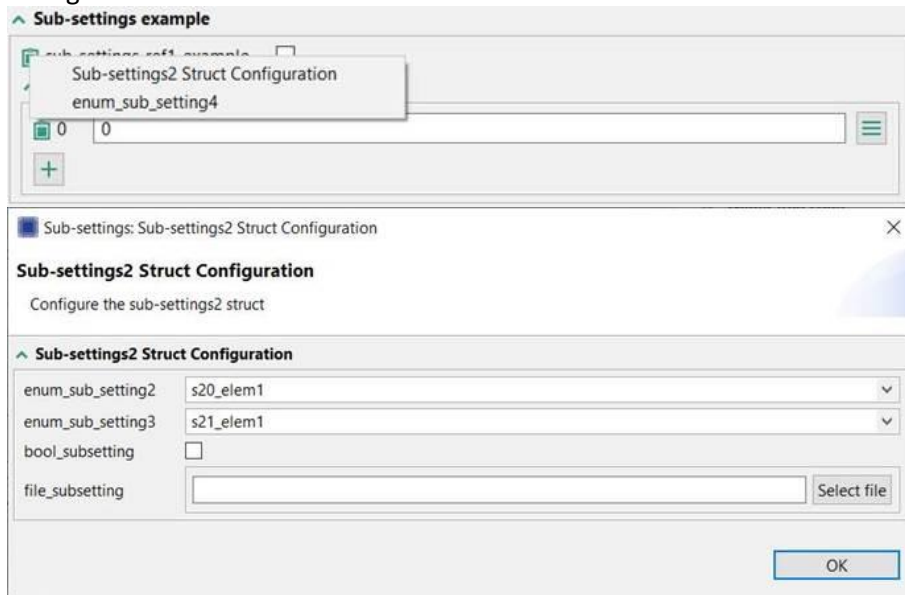


Figure 24: Peripherals Tool - Sub-settings

Sub-settings per each setting feature offers the possibility to display and configure more settings that can influence the state of the current configuration and the code that is generated. These settings can be unique or shared with other settings elements.

- Update code with dependencies from command line

Create new configuration by importing toolchain project	-ImportProject {path}	Creates new configuration by importing toolchain project based on the found .mex or yaml info. It updates code and validates configuration. Parameter is path to the root of the toolchain project
---	-----------------------	--

Figure 25: Peripherals Tool - Update code from command line

When executed from IDE, it imports toolchain project and updates code looking also into the sdk_manifest.xml file(s) if it contributes to the code generation and brings all defined components dependencies.

- Import AUTOSAR ECU configuration file

S32 Configuration Tools framework can recognize and import AUTOSAR ECU configuration files and display the content into Peripherals view based on the extracted info:

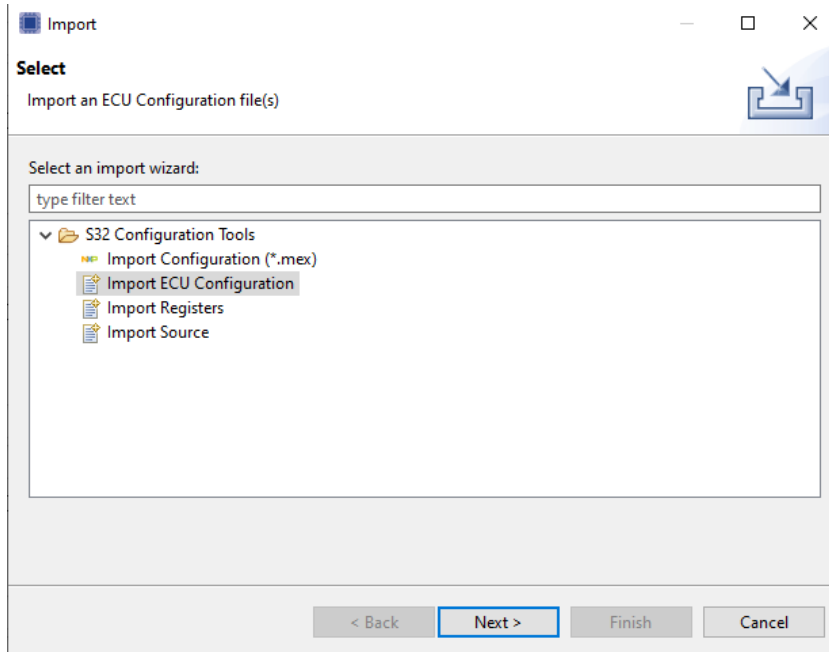


Figure 26: Peripherals Tool - Import ECU wizard

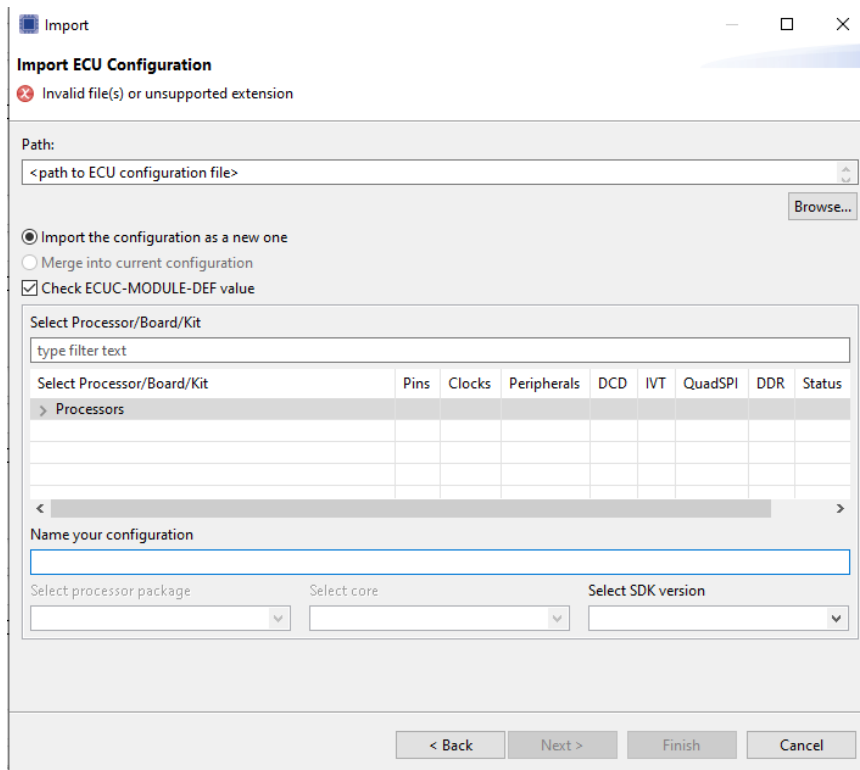


Figure 27: Peripherals Tool - Import ECU configuration file

One can import an ECU Configuration generated previously in the current configuration or as a new configuration.

Support is available in UI through new project wizard or in command line using:

Import ECU Configuration files	-ImportEcuConf	Import ECU Configuration file(s) (*.arxml, *.*) into configuration. Importing is done after loading mex or creating a new configuration and before generating outputs
--------------------------------	----------------	--

Figure 28: Peripherals Tool - Import ECU, command line

Note: Currently, ECU configuration can be imported into the Peripherals tool only.

Note: Imported ECU Configuration file(s) should be compatible with the AUTOSAR 4.4 schema version.

- Synchronize a setting's value between different instances of the same component

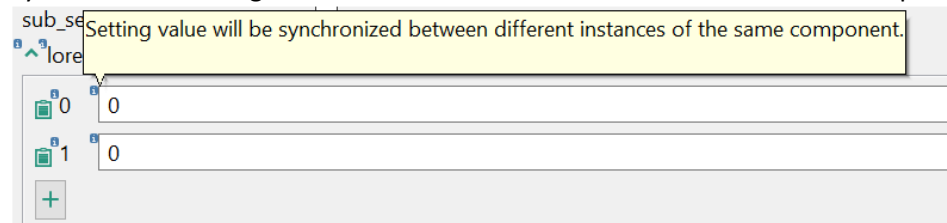


Figure 29: Peripherals tool - synchronize values

The values of such settings are shared between all instances of a component and is highlighted in the UI with a specific decorator as shown in the screenshot.

- New preference introduced that controls if code is generated or not in "Enable Code Preview"

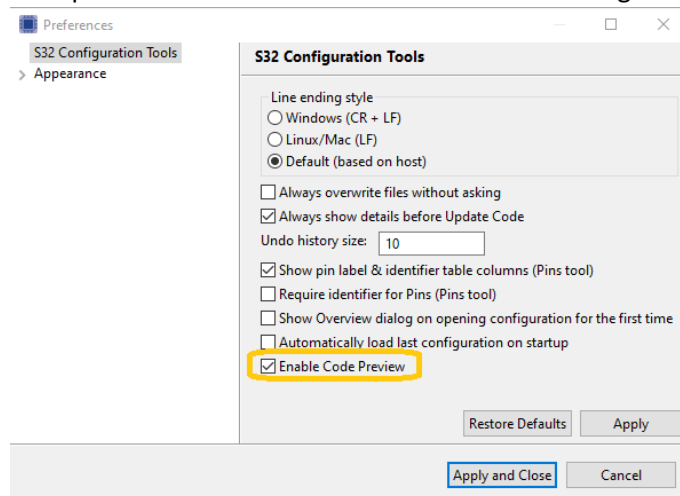


Figure 30: Peripherals Tool - Enable Code Preview

When this preference is enabled, code generation is performed automatically after every change in the configuration and the Code Preview is updated accordingly. When this preference is disabled, code generation is stopped, warning message is displayed in Code Preview window, and the action can be manually triggered by using one of the available options:

- By pressing “Generate Code” icon located on the toolbar



Figure 31: Peripherals Tool - Generate Code

- By pressing “generate code” link highlighted in the warning message from the Code Preview window
- By pressing “Update Code” button on the main toolbar, where code update is preceded by code generation

- Peripherals Tool allows selection of generated artifacts through Command Line API.

Added new command line argument called `-ExportArgs` that can be used to export its given arguments to javascript via `scriptApi`. The exported arguments can be retrieved in javascript by calling: `scriptApi.getUtils().getExportedArgs()`.

Example of usage:

```
-HeadlessTool Peripherals -ExportArgs epc epd
```

This new option was introduced to pair with existing `-ExportAll` that exports everything no matter what. The new option does a filtering over generated artifacts.

This is used by internal SDK/RTD teams while developing components with Config Tools.

- Peripherals Tool allows component filtering through Command Line API.

Added new command line argument called `-ExportComponentIds` that can be used to export its given component ids to javascript via `scriptApi`. The exported arguments can be retrieved in javascript by calling:

`scriptApi.getUtils().getExportedComponentIds()`. This new argument should be used in combination with `-ExportArgs` flag to export certain files for the given list of component ids.

Example of usage:

```
-HeadlessTool Peripherals -ExportArgs epc epd -ExportComponentIds  
eth_gmac_43 adc_16
```

This new argument offers a new level of filtering to generate specific artifacts for specific component(s).

This is used by internal SDK/RTD teams while developing components with Config Tools.

- Support for multiple `sdk_features.xml` and `categories.xml` files that allow modular SDK/RTD development.

Added mechanism to support multiple `sdk_features.xml` and `categories.xml` files defined at once. This allows SDK/RTD teams which are integrating their components to define more such files and the tool will resolve all of them.

This is used by internal SDK/RTD teams while developing components with Config Tools.

- Enabled inheritance for component settings to improve component development process.

Added mechanism which allows creating component settings by starting from a base setting. There are cases when a lot of settings have common parts (e.g. same options, conditions). Using this mechanism component developers can define new settings starting from a base setting, and the newly created settings would contain the existing characteristics of the base setting avoiding re-writing the same options for each setting. This can be enabled by using `base_type` attribute where the id of the base setting needs to be specified. For example:

```
<struct id="baseStruct">
  <options_expr>
    <option id="structOption" expr="$this.getId()"/>
  </options_expr>
  <string id="name" label="Name"/>
</struct>
<struct id="struct2" base_type="baseStruct">
  <string id="value" label="Value"/>
</struct>
```

`struct2` has as a base type `baseStruct` inheriting all options and settings from its base. This is used by internal SDK/RTD teams while developing components with Config Tools.

- Enable/disable state for Config Time support, option is added in “Global settings” view.

Configuration classes and variants is an AUTOSAR concept, added support in the framework.

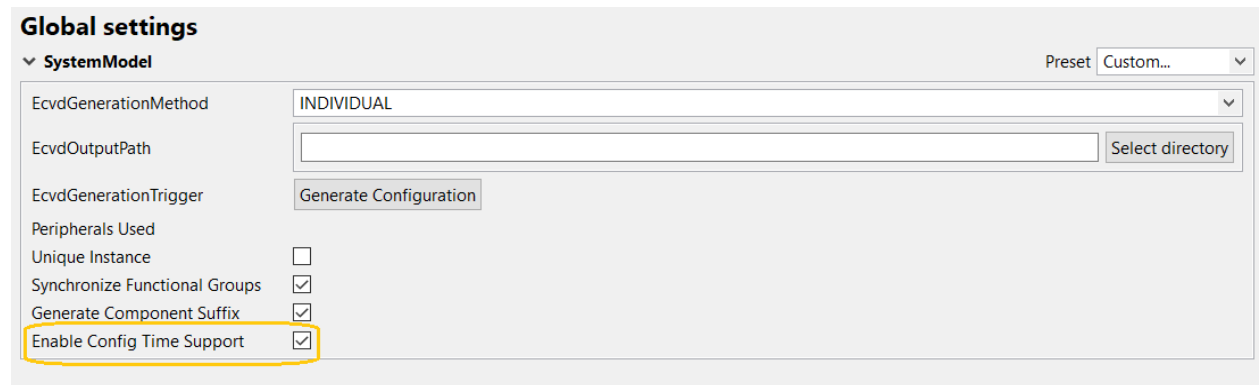


Figure 32: Peripherals Tool - Enable Config Time support

This is used by internal SDK/RTD teams while developing components with Config Tools.

- Support to execute a JavaScript file from another JavaScript file.
This is used by internal SDK/RTD teams while developing components with Config Tools.
- Removed the suffix from the instance name of a component when there is only one allowed instance.

Added new option called GENERATE_COMPONENT_SUFFIX based on which _1 suffix is added or not in the name of the component when max_instances="1" is set.
Name of the component with GENERATE_COMPONENT_SUFFIX option checked (true):

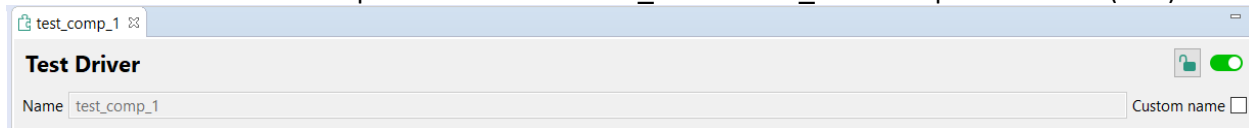


Figure 33: Peripherals Tool - Generate component suffix

Name of the component with GENERATE_COMPONENT_SUFFIX_option unchecked (false):

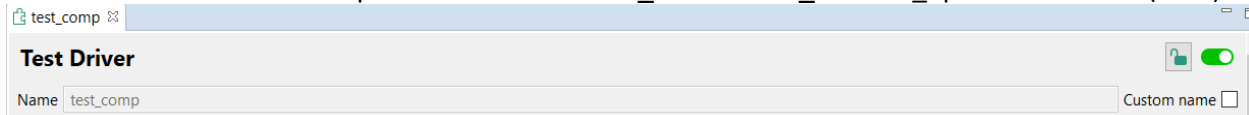


Figure 34: Peripherals Tool - Do not generate component suffix

This enablement is done in the `system.component` file.
This is used by internal SDK/RTD teams while developing components with Config Tools.

- Enabled synchronization of component instances between functional groups.

Added a new option called SYNC_FUNCTIONAL_GROUPS which enables synchronization of component instances between functional groups. More clearly, if a component is added/removed/renamed in one functional group, this action will be reflected to the rest of functional groups as well.

In order to enable this feature, the following must be added into `system.component` file:

```
# in component <options_expr> tag
<option id="SYNC_FUNCTIONAL_GROUPS"
available="$components.system.#global.syncFunctionalGroups.getValue()"/>
# in config set <settings> tag
<bool id="syncFunctionalGroups" label="Synchronize Functional Groups"/>
```

This is used by internal SDK/RTD teams while developing components with Config Tools.

- Enabled synchronization of the array size between different component instances.

Added a new option called SYNC_SIZE applicable to arrays that will enable the synchronization of the array size between different component instances. More clearly, when adding/removing an item from an array, the same action will be performed for other arrays of the same component instances.

Note: This will synchronize only the SIZE of the array, the array elements will remain unsynchronized.

Example of usage:

```
<array id="someId" type="someType" options="SYNC_SIZE">
```

This is used by internal SDK/RTD teams while developing components with Config Tools.

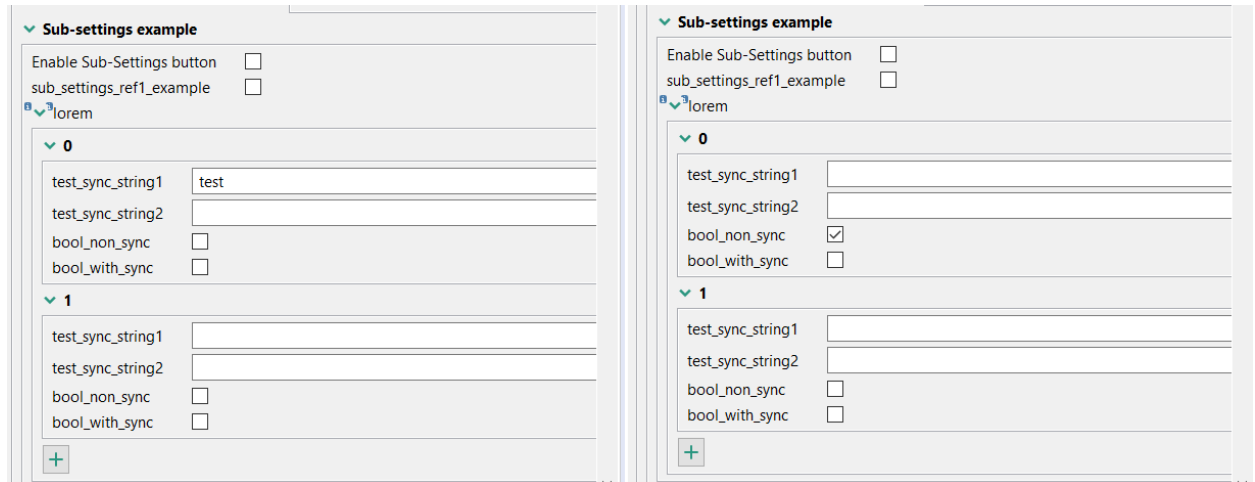


Figure 35: Peripherals Tool - Sync array size

- Extended the current support for querying Pins Tool data from Peripherals Tool adding the following options:

`configuredUIValue` – can be used to get any property of a routed signal

Example of usage:

will return slewRateControl value of PJ_07 pin

```
queryFeatureAdvanced(`PinName`, `PJ_07:slewRateControl`, `configuredUIValue`)
```

`assignedBitFieldValue` – can be used to get any bitfield value of an routed signal

Example of usage:

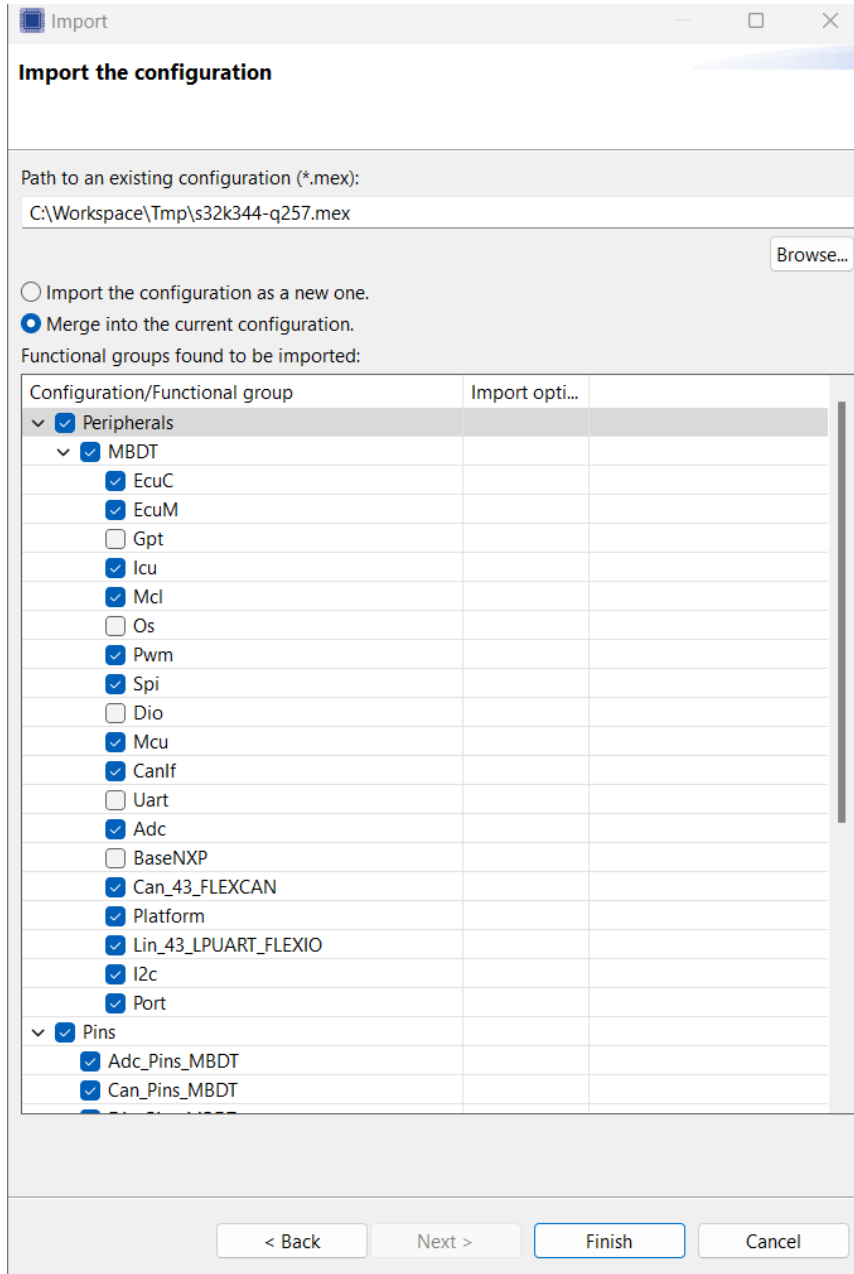
will return the value of SSS bitfield of SIUL2_1_IMCR398 register of PJ_07 pin

```
queryFeatureAdvanced(`PinName`, `PJ_07:SIUL2_1_IMCR398.SSS`,  
`assignedBitFieldValue`)
```

This is used by internal SDK/RTD teams while developing components with Config Tools.

- Added support to import an EPC file that has variant information inside.
This is used by internal SDK/RTD teams while developing components with Config Tools.
- Added support to take settings values from external references. Some settings can take their values not only from their definition file (.component) but also from an external file (used as a storage) which can contain values for different settings.
This is used by internal SDK/RTD teams while developing components with Config Tools.
- Added possibility to create shortcut links from a setting in a component file to another setting/node, in the same component file or a different one.

- Extended import functionality to allow selecting a component or a subset of components from the input configuration:



Import

Import the configuration

Path to an existing configuration (*.mex):
C:\Workspace\Tmp\s32k344-q257.mex Browse...

☐ Import the configuration as a new one.
☒ Merge into the current configuration.

Functional groups found to be imported:

Configuration/Functional group	Import opti...
✓ Peripherals	
✓ MBDT	
✓ EcuC	
✓ EcuM	
□ Gpt	
✓ Icu	
✓ Mcl	
□ Os	
✓ Pwm	
✓ Spi	
□ Dio	
✓ Mcu	
✓ CanIf	
□ Uart	
✓ Adc	
□ BaseNXP	
✓ Can_43_FLEXCAN	
✓ Platform	
✓ Lin_43_LPUART_FLEXIO	
✓ I2c	
✓ Port	
✓ Pins	
✓ Adc_Pins_MBDT	
✓ Can_Pins_MBDT	

< Back Next > **Finish** Cancel

Figure 36 Importing a subset of components from the input configuration.

3.4 Device Configuration Data Tool

Device Configuration Data (DCD) tool generates the DCD image using the format and constraints specified in the BootROM reference manual. BootROM reads and interprets the DCD image to configure various peripherals on the device. The location of the DCD is determined by the pointer in Image Vector Table (IVT) image.

DCD contains three types of commands:

- Write, write a list of bytes
- Check, test a list of bytes
- Nop, has no effect

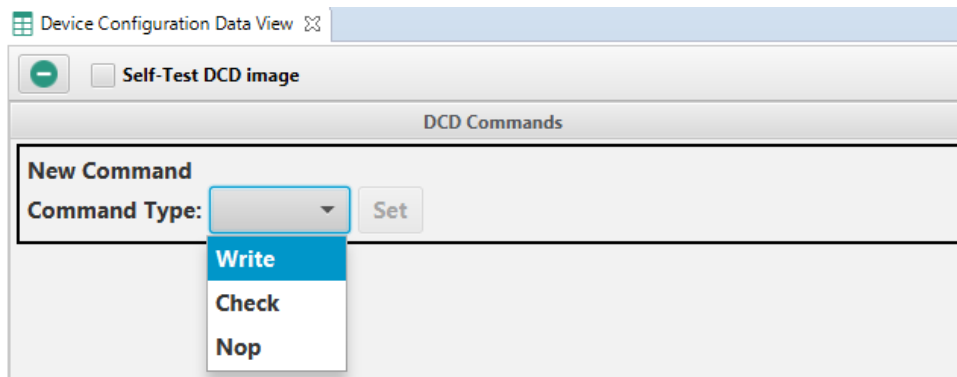


Figure 37: DCD Tool – Commands

There are two views:

- Binary view displays DCD image in a binary format
- DCD Command view

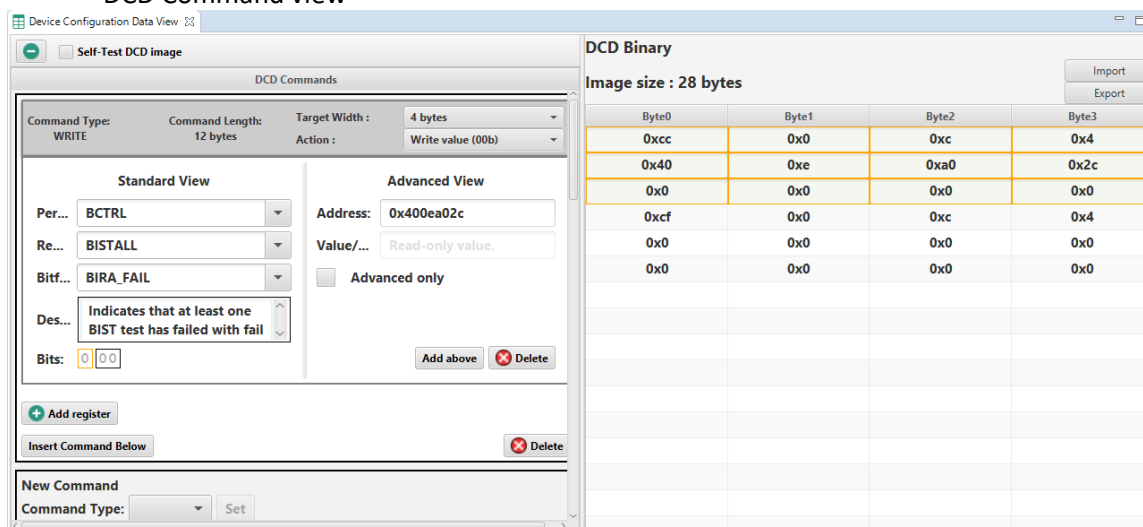


Figure 38: DCD Tool – Views

Device configuration records can be saved in a variety of formats (C array, binary and proprietary .mex format):

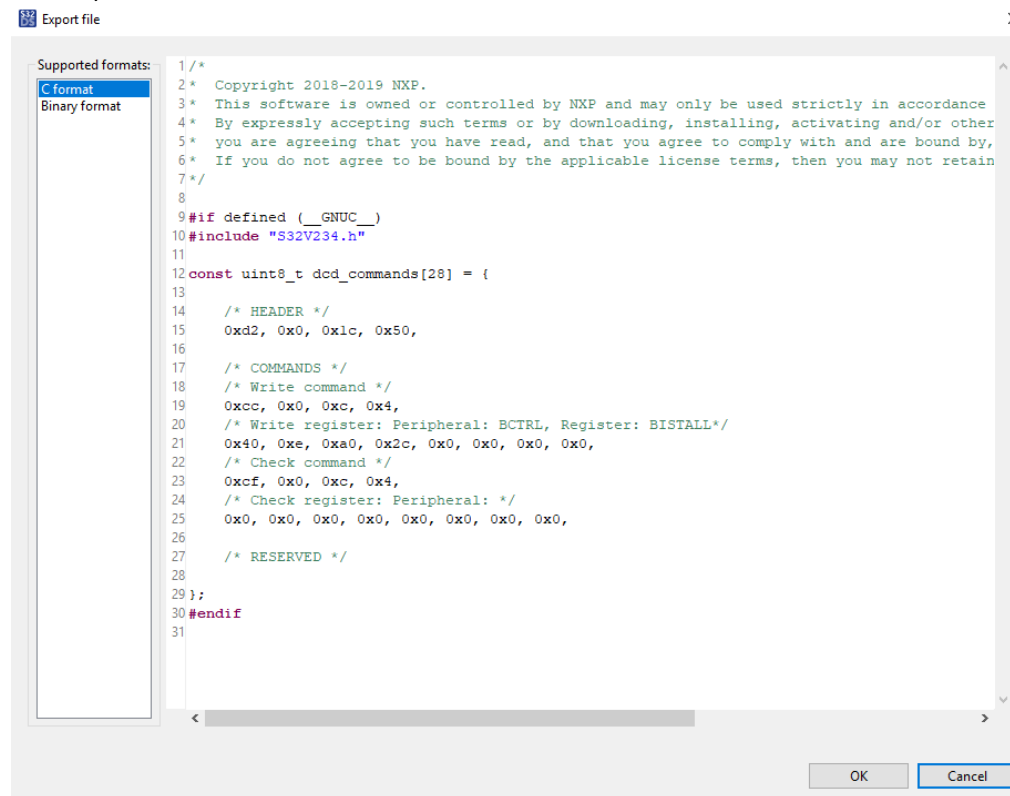


Figure 39: DCD Tool – Export image

Values can be reset to processor's defaults:

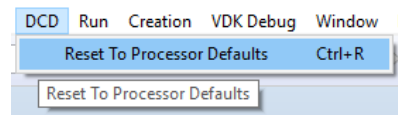


Figure 40: DCD Tool – Reset to Defaults

All reserved memory segments are set to 0xFF.

DCD Tool has command line support for a set of actions:

Command name	Command argument	Description
Import DCD binary image	-ImportBin	Import the binary DCD image in the current configuration. The path of the imported binary image is expected as argument.
Export DCD image in binary format	-ExportBin	Export the DCD image in binary format. Folder name is expected as argument.
Export DCD image in C format	-ExportC	Export the DCD image in C format. Folder name is expected as argument.
Export all generated files (to simplify all export commands to one command)	-ExportAll	Export generated files (with source code etc.). Code will be regenerated before export. Includes -ExportBin,-ExportC and in framework -ExportMEX. Folder name is expected as argument.

Figure 41: DCD Tool - Command line options

Duplicate functionality is available for the DCD commands:

The screenshot shows the 'Device Configuration Data View' window with the 'DCD Commands' tab selected. The 'Command Type' is set to 'CHECK', 'Command Length' is '12 bytes', and 'Target Width' is '4 bytes'. The 'Action' is 'All bits in mask clear'. The 'Standard View' section shows 'Peripheral' as 'PMUEVENTOBSERVER', 'Register' as 'CTL0', 'Bitfields' as 'Select bitfield', and 'Description' as an empty text box. The 'Bits' field shows '000000'. The 'Advanced View' section shows 'Address' as '0x0', 'Value/Ma...' as '0xc', and 'Count' as 'Infinite'. There is an 'Advanced only' checkbox. At the bottom, the 'Duplicate' button is highlighted with a red box, along with 'Insert Command Below' and 'Delete' buttons.

Figure 42 Duplicate command button

Record command is available for a limited set of processors:

The screenshot shows the 'Device Configuration Data View' window with the 'DCF Records' tab selected. The 'Command Type' is set to 'RECORD' and 'Command Length' is '8 bytes'. The 'Target Module' is 'Select module' and 'Client selection' is 'Select client'. The 'Data Word View' section shows 'Data Word' as '0x0', 'Bitfields' as 'Select bitfield', and 'Description' as an empty text box. The 'Bits' field is empty. The 'Control Word View' section shows 'Control ...' as '0x0', 'Chip Sel...' as 'Chip', 'Address' as '0x0', and 'Parity' as an empty checkbox. At the bottom, the 'New Record command' button is highlighted with a red box, along with 'Insert Command Below' and 'Delete' buttons.

Figure 43 Record command type

[illegible]

- Implemented CRC algorithm (32bit Castagnoli CRC) for ensuring the data integrity of the DCD image, applicable to a limited set of processors

New features in S32CT 1.7 U8 R version:

- Added the option to validate the DCD configuration in CLI mode

3.5 Image Vector Table Tool

Image Vector Table (IVT) tool configures and generates the IVT image, the first data structure that BootROM reads from boot device. The IVT contains the required data components like image entry point, pointer to DCD and other pointers used by the BootROM during boot process.

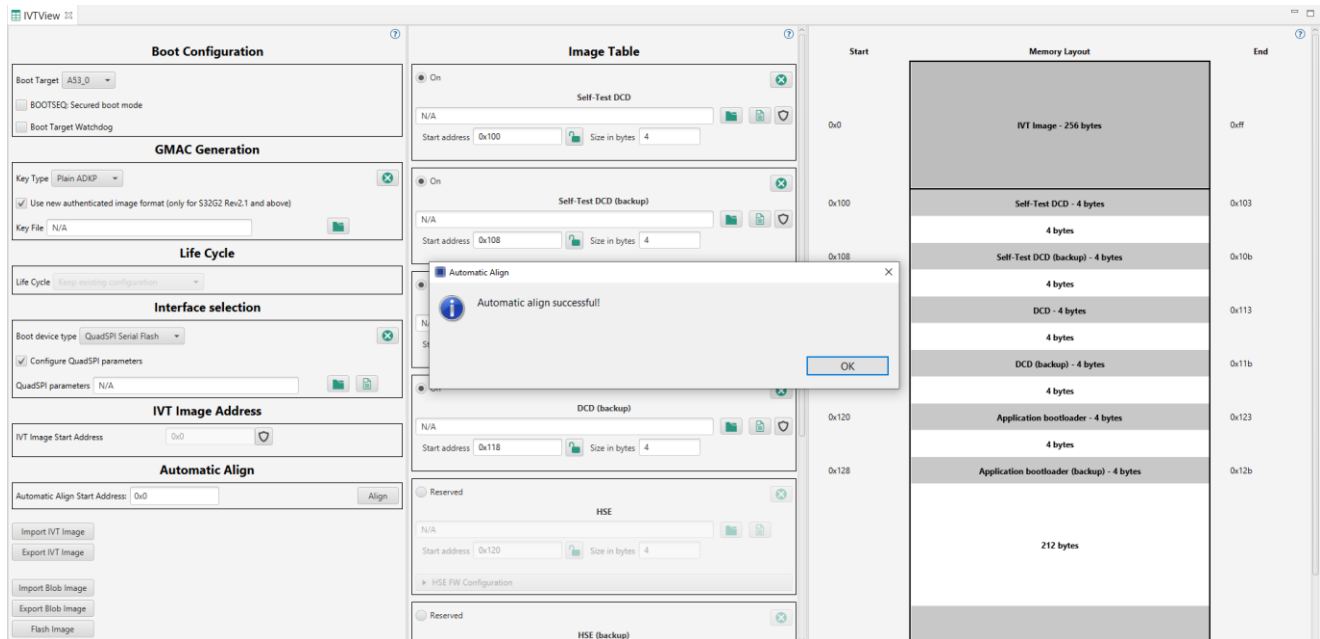


Figure 45: IVT Tool – Automatic Align

User can start the configuration using three views:

- Boot Configuration panel allows configuration of boot specific options
- Image Table view allows configuration of IVT pointers by selecting size and start address
- Memory Layout is a graphical representation of the memory to visualize the IVT segments

IVT tool segments can be automatically aligned to eliminate overlaps.

Application bootloader image can be generated where a pre-validation mechanism is used to check for its accuracy.

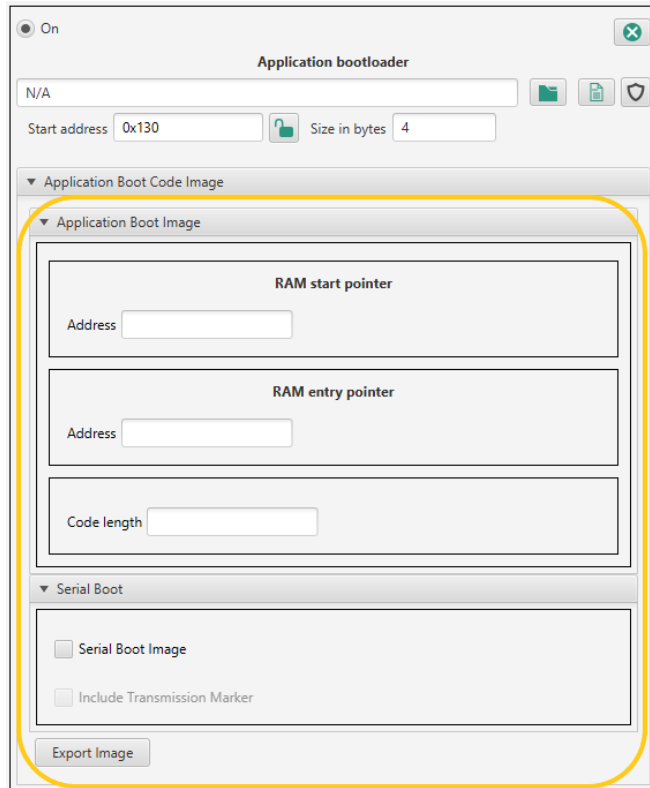


Figure 46: IVT Tool – Application Boot Image

Control mechanism for reserved sections and pointers will be set to 0xFFFFFFFF.

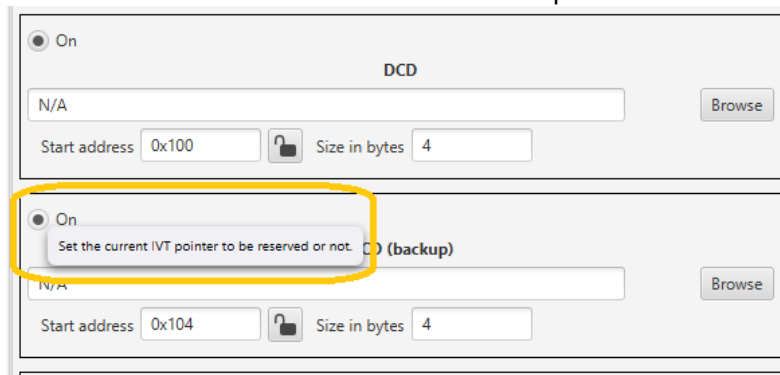


Figure 47: IVT Tool – Reserved pointers

The currently available import options are:

- Import IVT Image
- Import Image from any IVT pointers

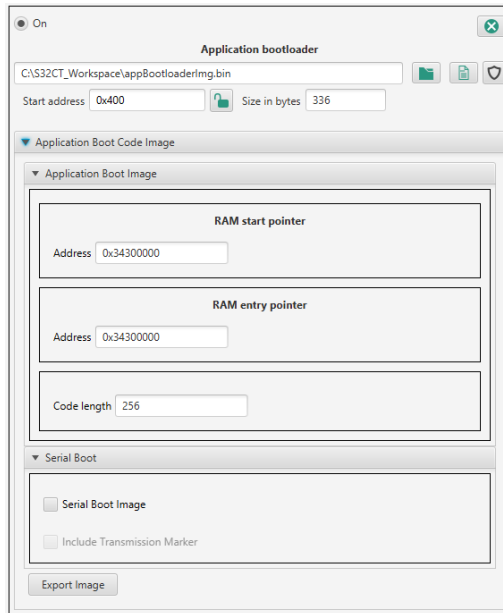


Figure 48: IVT Tool – Import Application bootloader

- Import Blob Image

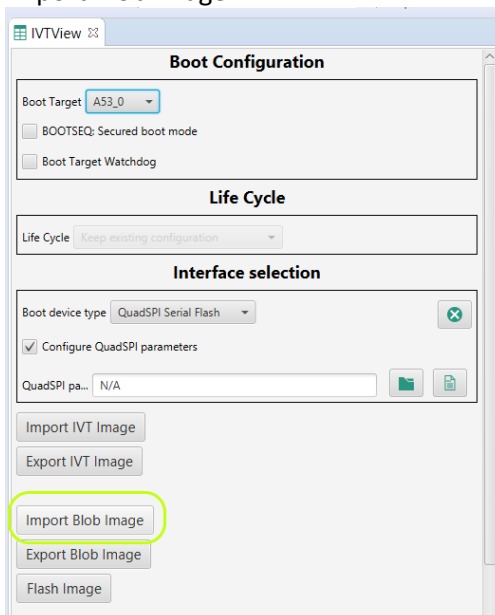


Figure 49: IVT Tool – Import blob image

Exporting all existing images as Blob file including IVT.bin, DCD.bin, HSE.bin, Application_bootloader.bin etc. is also possible.

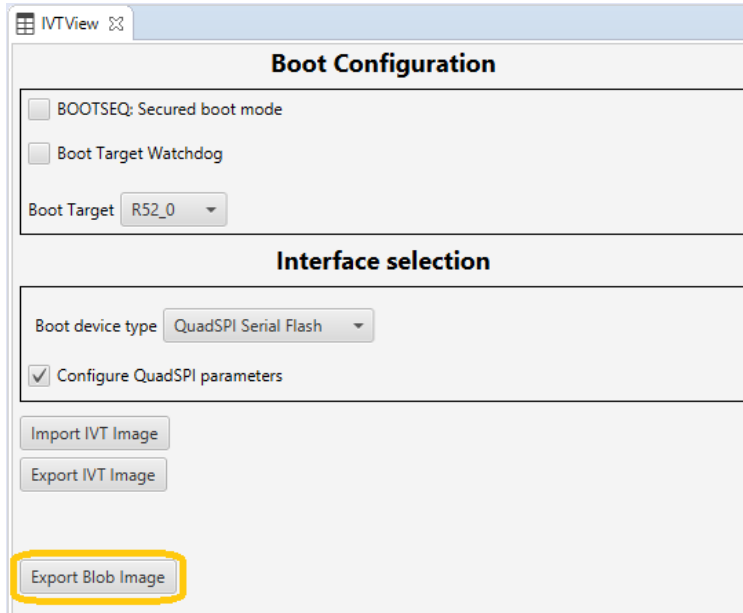


Figure 50: IVT Tool – Export Blob Image

The currently available export options are:

- Exporting IVT Image as binary
- Exporting IVT Image as C code
- Exporting Blob Image as binary
- Exporting Application Bootloader sub-image as binary



Figure 51: IVT Tool – Export image

IVT Tool provides a graphical user interface to flash the IVT image, making the interaction with the Flash Tool more effective.

The IVT Flash feature can be accessed in the IVT Tool view, in the Buttons area (near Export, Import and Export Blob buttons):

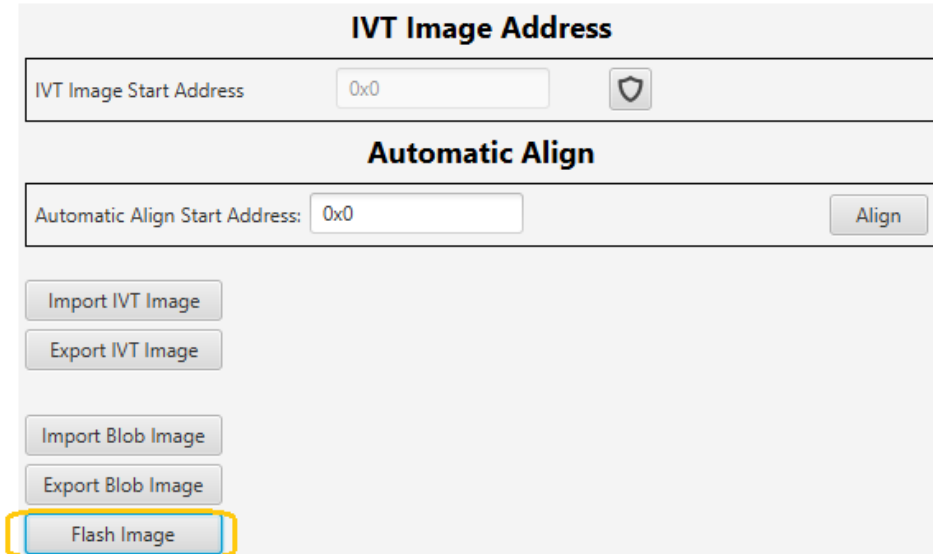


Figure 52: IVT Tool - Flash IVT image option

Once the button is clicked, a flash dialog pop-up should appear on the screen.

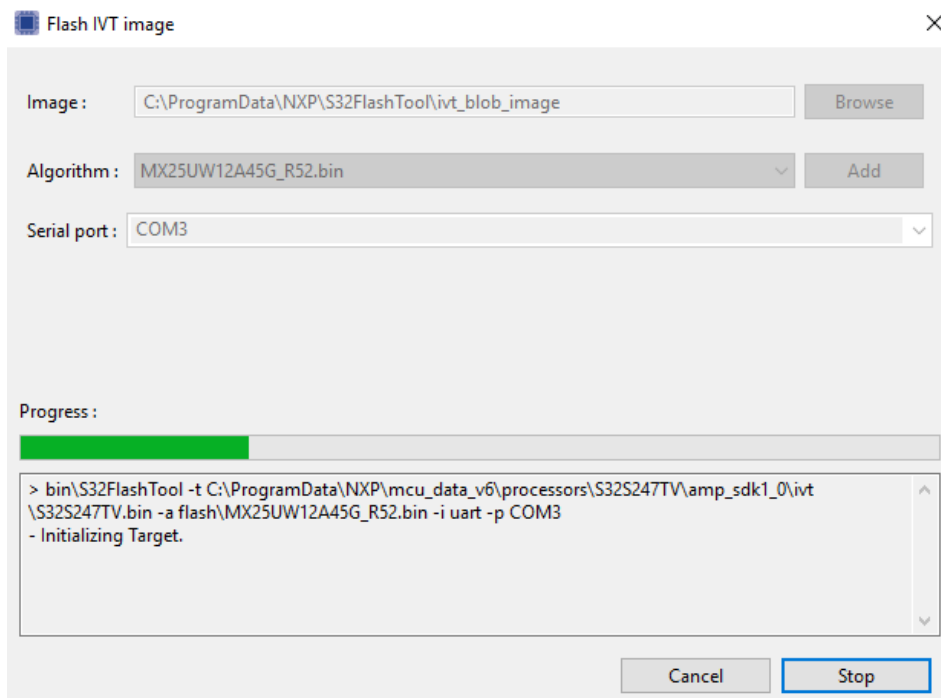


Figure 53: IVT Tool - Flash IVT image dialog

IVT Tool's operations have support in Command line:

Command name	Definition and parameters	Description
Import IVT binary image	-ImportBin	Import the binary IVT image in the current configuration. The path of the imported binary image is expected as argument.
Export IVT image in binary format	-ExportBin	Export the IVT image in binary format. Folder name is expected as argument.
Export IVT image in C format	-ExportC	Export the IVT image in C format. Folder name is expected as argument.
Export Blob image in binary format	-ExportBlob	Export Blob image (the binary which contains IVT image and all loaded pointer images). Folder name is expected as argument.
Export application bootloader image	-ExportAB	Export the complete application bootloader image. Folder name, path to the raw binary code file, start ram pointer address value and entry ram pointer address value are expected as arguments.
Insert value for RAM start pointer address	-start_pointer_addr	Value used for the RAM start pointer address for application bootloader. This value will be included in the header of the exported application bootloader.
Insert value for RAM entry pointer address	-entry_pointer_addr	Value used for the RAM entry pointer address for application bootloader. This value will be included in the header of the exported application bootloader.
Path to the raw binary code for the application bootloader pointer	-raw_binary	Path to a binary file which contains the raw code (no header) for the application bootloader pointer.
Export the application bootloader image in serial boot format	-serial_boot	Export the application bootloader image in serial boot format
Add transmission marker	-include_marker	Include the marker FEED_FACE_CAFE_BEEFh on the exported image
Export all generated files (to simplify all export commands to one command)	-ExportAll	Export generated files (with source code etc.). Code will be regenerated before export. Includes -ExportBin, -ExportC, -ExportBlob and in framework -ExportMEX. Folder name is expected as argument.
Validate configuration	-ValidateConfiguration	Validates the current loaded configuration and display the problems found.
Automatic alignment for IVT segments	-AutoAlign	Automatically aligns all segments in order to fix alignment problems and segment overlapping problems. It has an optional parameter. The user can specify the start address from which the pointers will be aligned or in case no parameter is specified the auto align start address from the storage will be used.

Figure 54: IVT Tool - command line options

More details and examples are available in the Help documentation.

Support is available for security features in IVT Tool:

- Generate IVT blob image in secure mode. The blob defines the layout of the images that will be loaded at boot time
- HSE Firmware configuration was added in the blob image with:
 - SYS-IMG Pointer Start Address
 - SYS-IMG Pointer (backup) Start Address
 - SYS-IMG External Flash Type

- SYS-IMG Flash Page Size
 - Application BSB External Flash Type
- GMAC (Galois Message Authentication Code) section is generated for each binary input using a specified ADKP (Application Debug Key/Password)

Figure 55: IVT Tool – HSE Configuration

- SYS-IMG pointers do not require a binary to be imported since their content will be generated at runtime by the HSE
- IVT Tool will check that no overlapping occurs between SYS-IMG pointers and other segments
- IVT Tool can generate the GMAC (Galois Message Authentication Code) bytes for each binary input using a specified ADKP (Application Debug Key/Password):

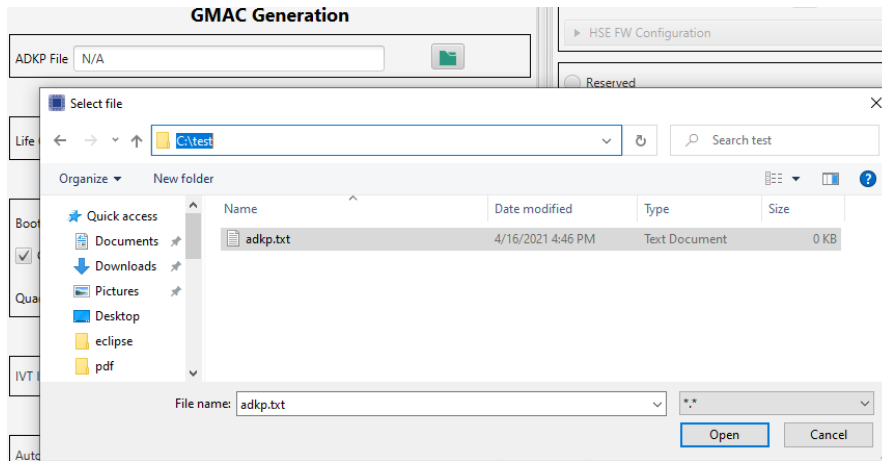


Figure 56: IVT Tool - GMAC generation

User must load a txt file that contains a string of 32 characters for the ADKP value. Using the ADKP value, the IVT, DCD, Self-Test DCD, and Application Bootloader images are signed when the Blob image is exported.

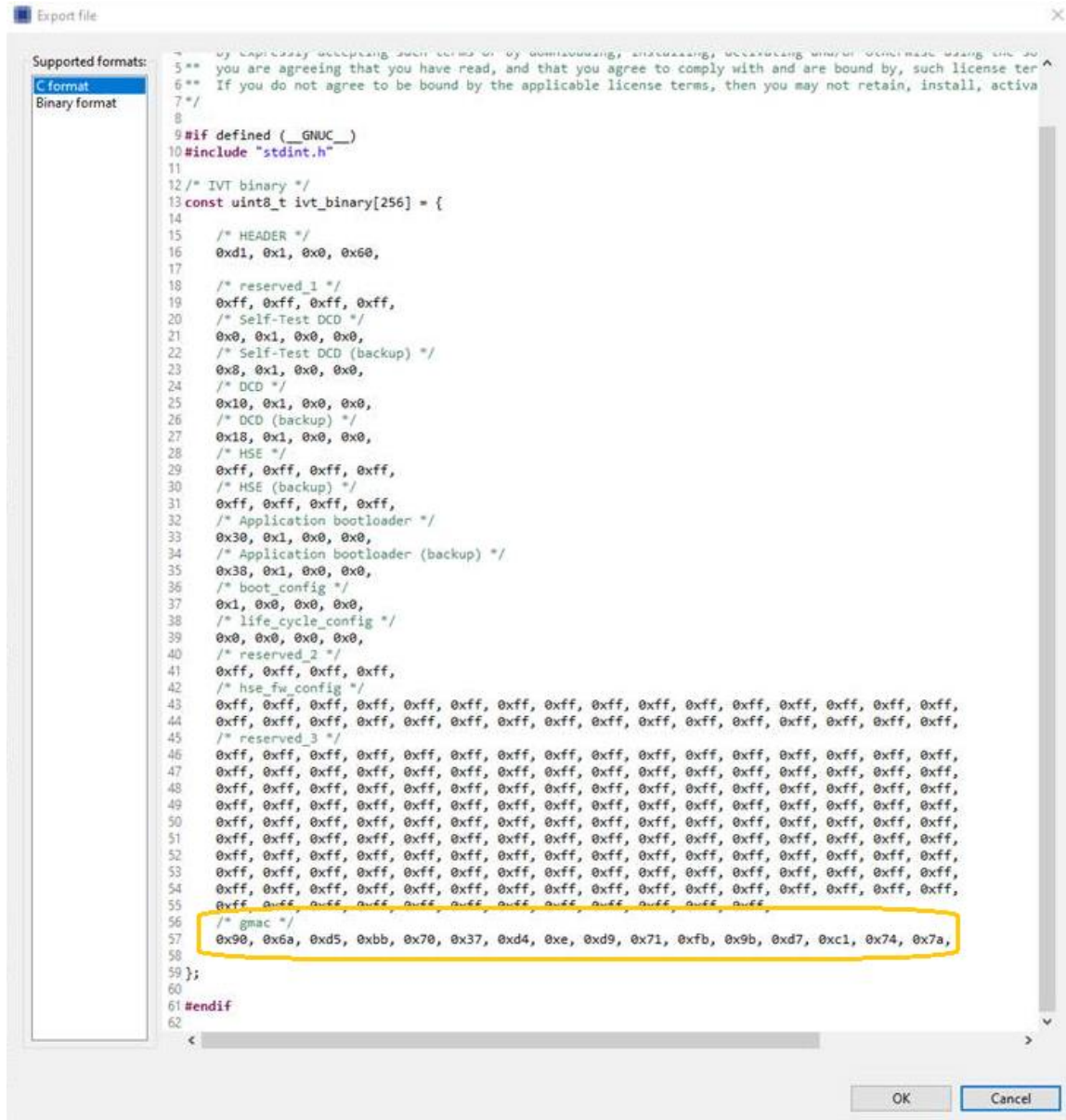


Figure 57: IVT Tool - GMAC generation

Note: Before using the GMAC generation feature, make sure the selected processor supports the offline GMAC generation using the ADKP value.

Context help action is available in the user interface for easy access to documentation:

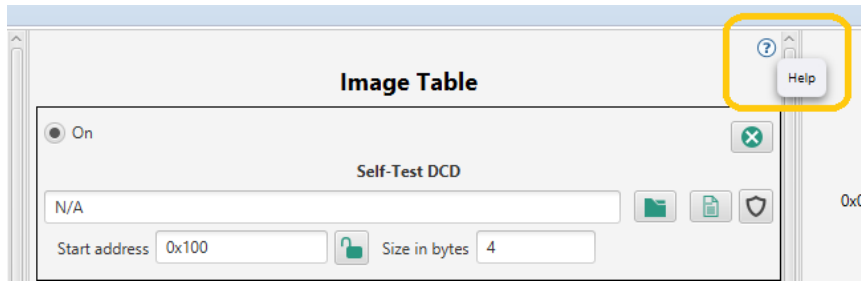


Figure 58: IVT Tool - Context help

The “?” icon is available at the top-right corner on the main panels available in the UI and it opens the same Help documentation available at Help > Contents. It is enabled for IVT, DCD, QuadSPI and DDR tools.

New option added to export the application bootloader image in serial boot format:

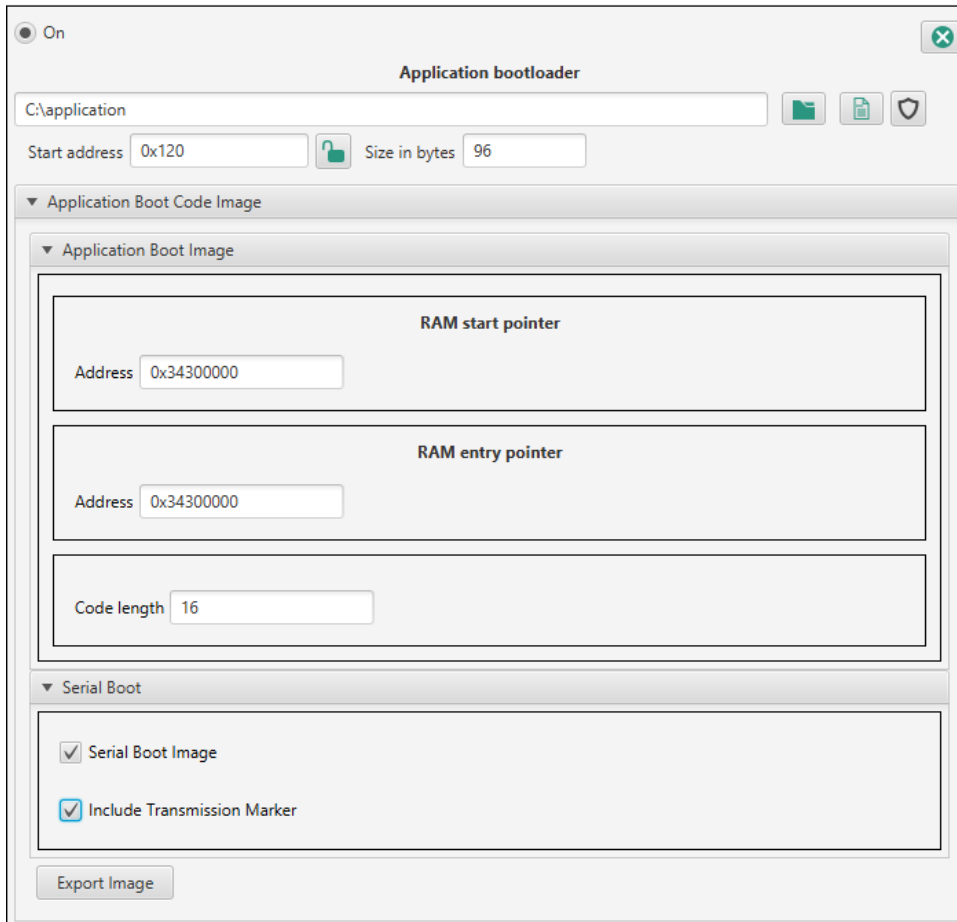


Figure 59: IVT Tool - Serial boot image

User can choose between the classical format of the application bootloader image and the serial boot format. However, please note that the serial boot image is not included in the final blob image as there is no reference to it.

- GMAC generation using Plain ADKP, Hashed ADKP, Wrapped ADKP, and UID calling volkano utils API (libraries and scripts, a subset of secure debug authorization framework).

Before, the IVT, DCD, Self-Test DCD, and Application Bootloader images could be signed based on a Plain ADKP at the time the Blob image was exported. In this release, new ways of generating the GMAC were introduced:

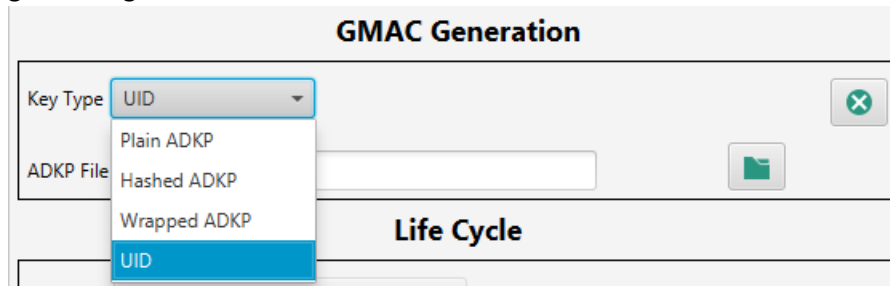


Figure 60: IVT Tool - Generate GMAC

Hashed ADKP is a 64 hex characters file generated based on the Plain ADKP.
 Wrapped ADKP is a 512 hex characters file generated based on the Hashed ADKP.
 UID represents the ID registered in volkano utils database based on the Wrapped ADKP.

- Added EFUSE VDD Marker and EFUSE VDD Word structures in HSE FW Configuration.

The eFuse VDD Marker is a checkbox that sets the bytes related to the VDD marker to a certain pattern and enables the configuration of eFuse VDD Word structure.

The eFuse VDD Word configures the Polarity (1 - when driving the GPIO High, 0 - when driving the GPIO Low), the delay before initiating any fuse writes and the GPIO MSCR value corresponding to the pin used in eFuse process.

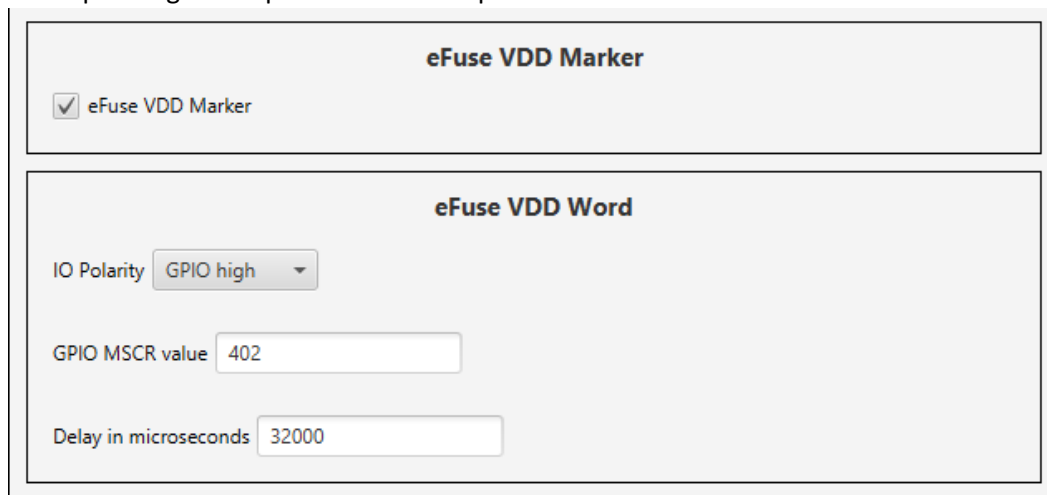


Figure 61: IVT Tool - EFUSE VDD configuration

- Added reserved buttons for SYS-IMG pointers in HSE FW configuration. When a SYS-IMG pointer is set on reserved, the bytes representing the start address of the pointer are set on reserved in the IVT Image.

Figure 62: IVT Tool - SYS-IMG pointers

- Highlight the fields that changed after automatic align process, these are marked with a blue background.

Figure 63: IVT Tool - Automatic align changes

- Added clear button for GMAC generation structure that will clear/reset to default the values from the GMAC structure

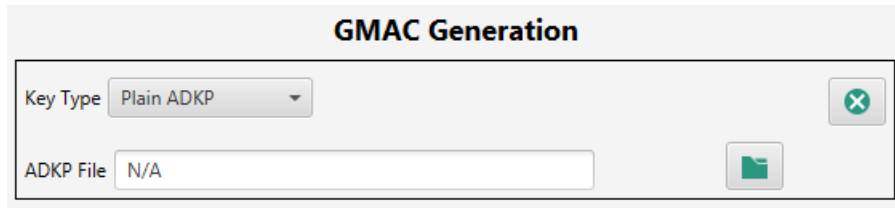


Figure 64: IVT Tool - Clear ADKP file selection

- Improved log messages in command line.
- Improved the Export file dialog for Binary format with a new table header and color highlighting based on the segments:
 - Red bytes represent the header of the image
 - Black bytes represent the segments in the image
 - Grayed out bytes represent the reserved values
 - Brown bytes represent the GMAC value of the IVT Image

Meaningful tooltips were enabled for each segment bytes.

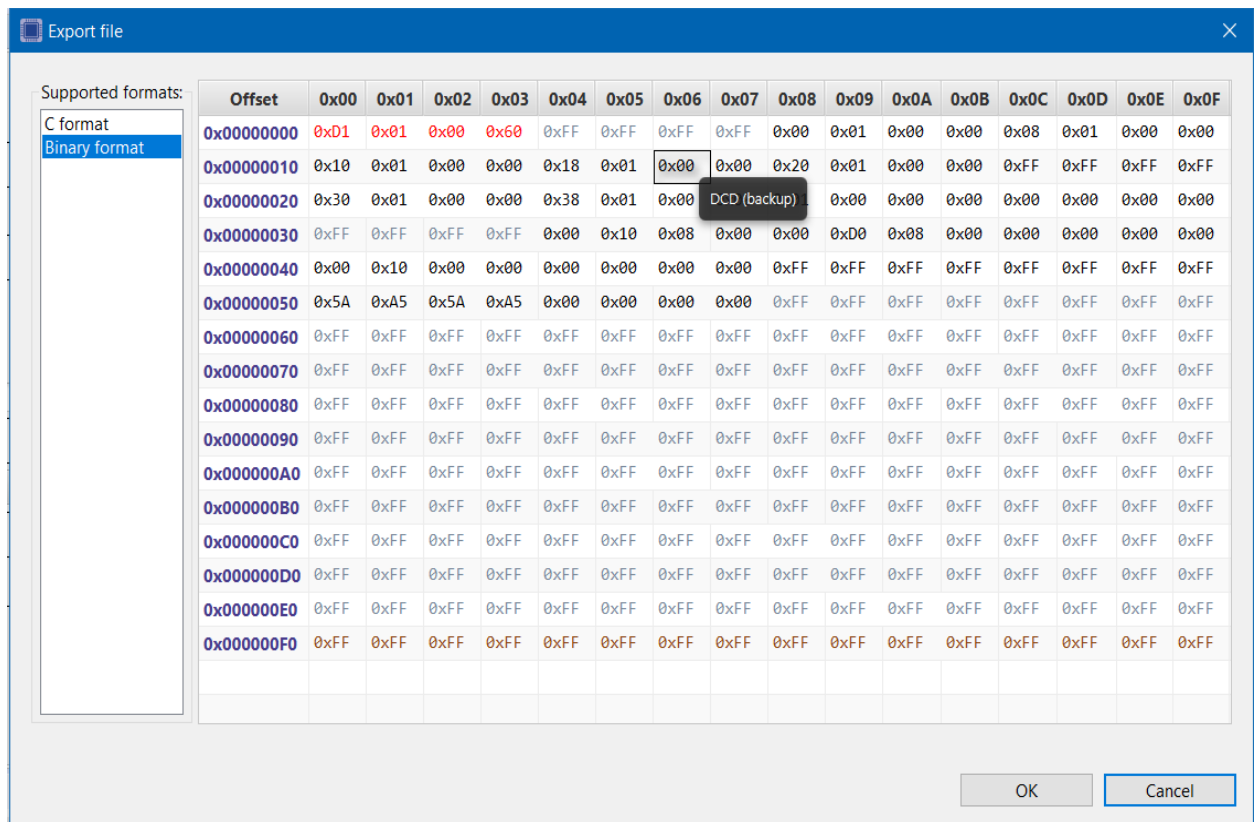


Figure 65: IVT Tool - Export File

This was enabled for all Boot tools – IVT, DCD, and QuadSPI tools.

Import/export functionality for DDRC image

New command line options:

Command name	Definition and parameters	Description
Import application bootloader image	-ImportAB	Imports the complete application bootloader image in the current configuration. The path of the imported complete application bootloader image is expected as argument.
Export DDRC interface init application image	-ExportDDRC	Export the complete DDRC interface init app image. Folder name, start ram pointer address value, entry ram pointer address value, path to the raw binary code file, bootrom timeout value and path to the pre-defined data file are expected as arguments.
Import DDRC interface init application image	-Import DDRC	Import the DDRC interface init app image into the current configuration. The path to the imported binary image is expected as argument.

Figure 66 Extended list of IVT CLI supported commands

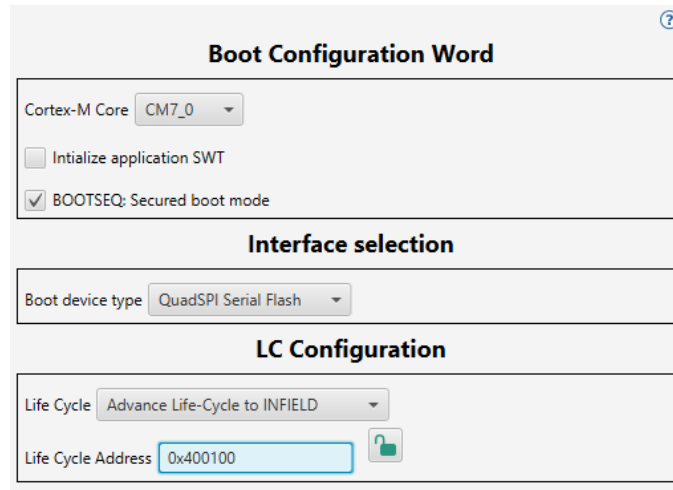
New IVT format supported, following Image Table structure is available for a limited set of processors:

The screenshot shows a configuration window titled "Image Table" with a help icon. It contains three sections, each for a different core:

- CM7_0 core:** The "On" radio button is selected. It has a text field with "N/A", a "Start address" field with "0x400100", and a "Size in bytes" field with "4". There are icons for folder selection and file upload.
- CM7_1 core:** The "Reserved" radio button is selected. It has a text field with "N/A", a "Start address" field with "0x400180", and a "Size in bytes" field with "4". There are icons for folder selection and file upload.
- CM7_2 core:** The "Reserved" radio button is selected. It has a text field with "N/A", a "Start address" field with "0x400200", and a "Size in bytes" field with "4". There are icons for folder selection and file upload.

Figure 67 New Image Table structure

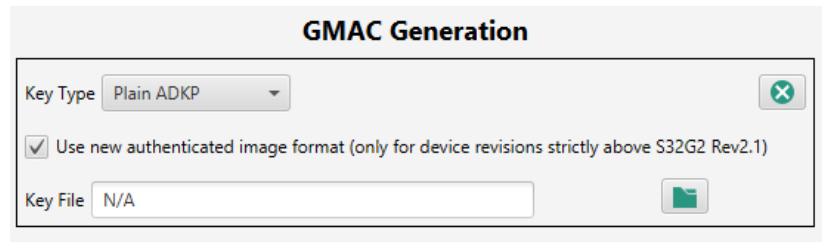
UI option for Life Cycle configuration word address, available for a limited set of processors:



The screenshot shows a software configuration window titled "Boot Configuration Word" with a help icon in the top right corner. It is divided into three sections: "Boot Configuration Word", "Interface selection", and "LC Configuration". In the "Boot Configuration Word" section, "Cortex-M Core" is set to "CM7_0", "Initialize application SWT" is unchecked, and "BOOTSEQ: Secured boot mode" is checked. In the "Interface selection" section, "Boot device type" is set to "QuadSPI Serial Flash". In the "LC Configuration" section, "Life Cycle" is set to "Advance Life-Cycle to INFIELD", and "Life Cycle Address" is set to "0x400100" with a lock icon to its right.

Figure 68 Life Cycle Address configuration field

Enabled new authenticated image format, available for certain processor revisions.



The screenshot shows a software configuration window titled "GMAC Generation". It contains a "Key Type" dropdown menu set to "Plain ADKP" with a close icon to its right. Below this, there is a checked checkbox labeled "Use new authenticated image format (only for device revisions strictly above S32G2 Rev2.1)". At the bottom, there is a "Key File" field set to "N/A" with a folder icon to its right.

Figure 69 Authenticated image format selection

Updated list of available GMAC generation methods to enhance security of the signing process, plain ADKP remains the only recommended option.

Extended list of supported IVT formats, new Image Table structure is available for a limited set of processors.

Improved command line flow for exporting application bootloader and IVT blob images by allowing both to be generated using a single command.

- Added possibility to create custom pointer nodes and include the corresponding images in the exported blob image, available for certain processors.

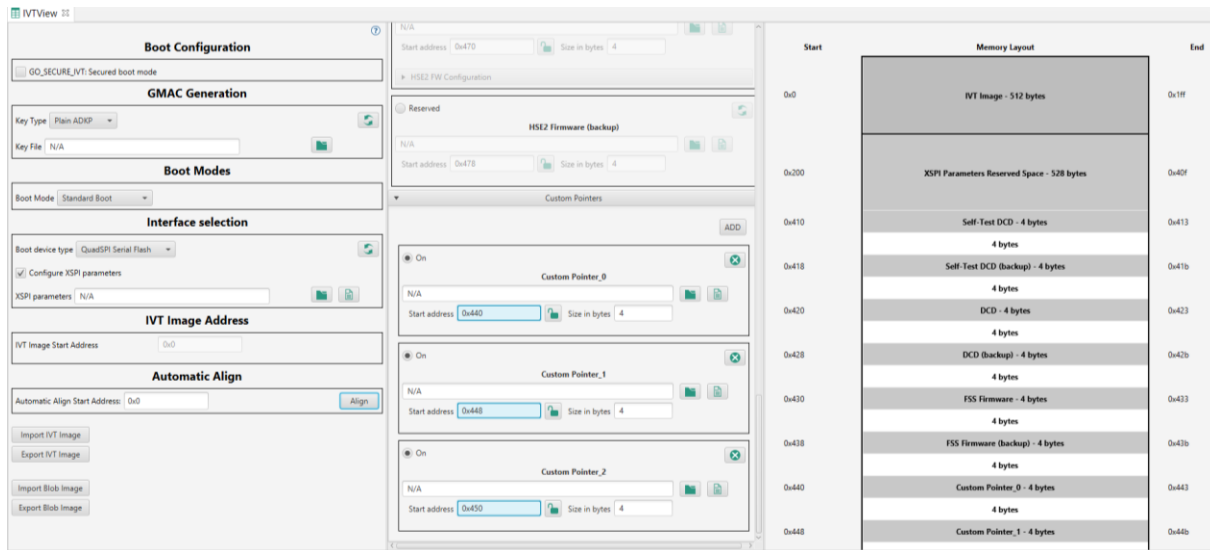


Figure 70 Custom pointers defined in IVT view

- Implemented CRC algorithm (32bit Castagnoli CRC) for ensuring the data integrity of the IVT image, applicable to a limited set of processors
- New command line options:

Command name	Definition and parameters	Description
Import Blob Image	-ImportBlob	Import Blob Image in a binary format. The command allows importing a blob image which includes the IVT image and creates temporary files for the pointers found in blob. The path of the imported blob image is required as argument.
Export pointers to a specific location	-ExportPointers	Exports the pointers to a specific location when importing blob image. The path to a folder where to export the pointers from the imported blob image is required as argument.

Figure 71 IVT CLI options for importing Blob image

New features in S32CT 1.7 U8 R version:

- Added option to keep the input binary files untouched on CLI mode, available on limited set of processors
- Added option to perform auto align during the blob export process
- Allow the import of adkp files that contain the hex prefix
- Added a reset button the Authentication Tag Generation structure

3.6 QuadSPI Tool

BootROM supports boot from external flash memory device over the QuadSPI interface. Booting from QuadSPI provides flexibility for choosing the configuration parameters for which the controller must be programmed during boot. QuadSPI tool allows the configuration of these parameters and generate the QuadSPI image that will be programmed in flash memory.

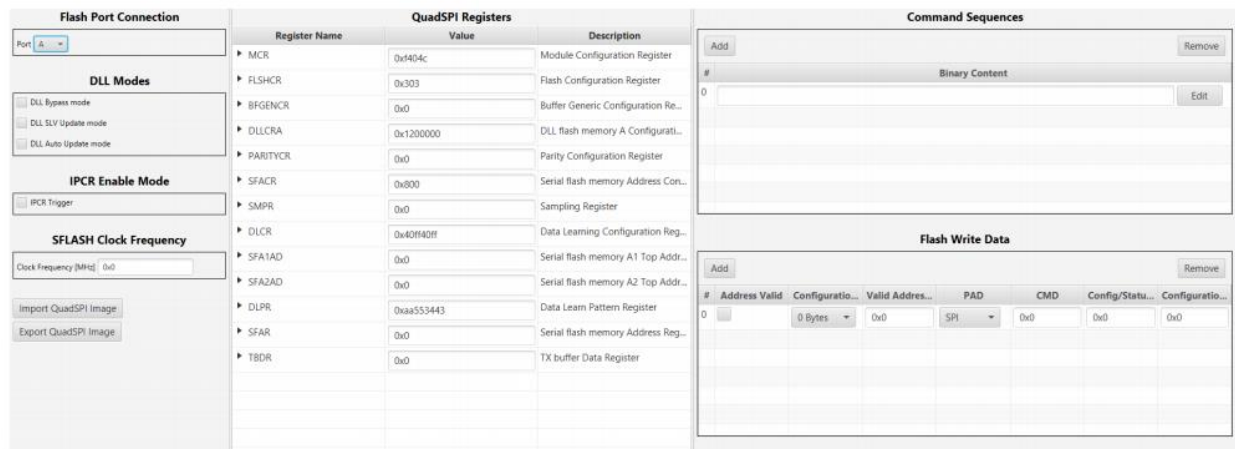


Figure 72: QSPI Tool – Overview

QuadSPI (QSPI) Tool graphical interface consists of four main parts:

- **QuadSPI General Parameters** where general parameters are configured, and final image imported/exported
- **QuadSPI Registers Table** that allow user to view and modify registers' values while checking that no reserved values get modified
- **QuadSPI Command Sequences**
- **Flash Write Data**

After the user obtains the optimal QuadSPI configuration, the result can be exported in a .bin file that will be imported in the QuadSPI parameters field of the IVT tool in order to be integrated in the final blob image.

QuadSPI General Parameters

Flash Port Connection

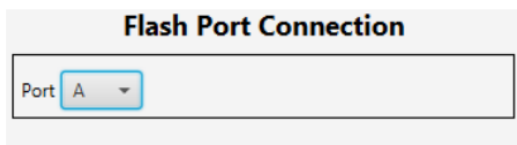


Figure 73: QSPI Tool – Flash Port Connection

Selects the QuadSPI port on which external flash memory is connected to (port A or B). DLLCR, SFLASH_1_SIZE and SFLASH_2_SIZE configured registers are dependent on the port selection (e.g. DLLCRA/DLLCRB).

DLL Modes

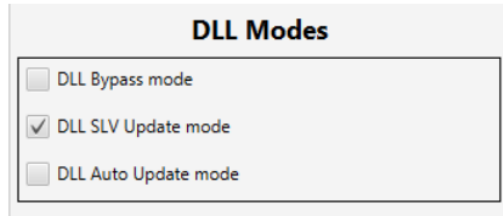


Figure 74: QSPI Tool – DLL Modes

IPCR Trigger

Enables the IPCR register in the QuadSPI Registers Table:

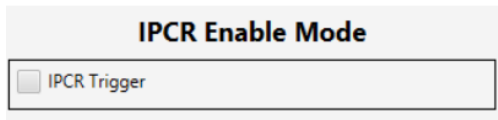


Figure 75: QSPI Tool – IPCR Enable Mode

SFLASH Clock Frequency

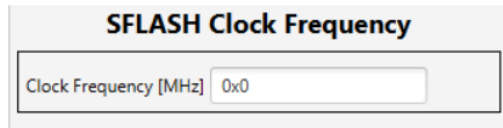


Figure 76: QSPI Tool – Splash Clock Frequency

User-provided frequency (in MHz) which corresponds to the QSPI_1X_CLK clock.

Import/Export button

User can select to import a .bin file that will be applied over the current configuration. User can select to export a .bin or .c file.

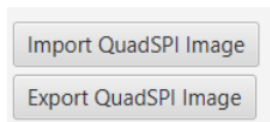


Figure 77: QSPI Tool – Import/Export image

QuadSPI Registers Table

Every QuadSPI register has a size of 4 bytes. Bitfields of the register can be viewed and modified when expanding a register. User can change the register value by setting the bitfields values or directly setting the register's value. The displayed bitfields and register values are always in sync.

QuadSPI Registers		
Register Name	Value	Description
► MCR	0xf404c	Module Configuration Register
▼ FLSHCR	0x303	Flash Configuration Register
Reserved	0000000000000000	(bits 31-18) Reserved bit-field
TDH	00	(bits 17-16) Serial flash memory...
Reserved	0000	(bits 15-12) Reserved bit-field
TCSH	0011	(bits 11-8) Serial flash memory ...
Reserved	0000	(bits 7-4) Reserved bit-field
TCSS	0011	(bits 3-0) Serial flash memory C...
► BFGENCR	0x0	Buffer Generic Configuration Re...
► DLLCRA	0x1200000	DLL flash memory A Configurati...
► PARITYCR	0x0	Parity Configuration Register
▼ SFACR	0x800	Serial flash memory Address Co...
Reserved	0000000000000000	(bits 31-18) Reserved bit-field
BYTE_SWAP	0	(bit 17) Byte swapping
WA	0	(bit 16) Word addressable
Reserved	00	(bits 15-14) Reserved bit-field

Figure 78: QSPI - Registers Table

If the user tries to enter a value that modifies a reserved bitfield, an error will be shown. The erroneous values will be reset to last valid version when the focus on the textfield is lost.

► DLLCRA	0x1200000	DLL flash memory A Configurati...
▼ PARITYCR	0x0FZXDCV	Parity Configuration Register
CRC_WNDW_FB	0	(bit 31) CRC address window co...
CHUNKSIZE_FB	000000	(bits 30-25) Chunk size for flash...
BYTE_SIZE_FB	0	(bit 24) Byte size for flash mem...
CRCEN_FB	0	(bit 23) CRC parity checker logic

Figure 79: QSPI - Registers Table Error

QuadSPI Command Sequence

The Command Sequence represents a Look-Up Table (LUT). The LUT consists of a number of pre-programmed sequences. Each sequence is basically a sequence of instruction-operand pairs, which when executed sequentially, generate a valid serial flash memory transaction. Each sequence can have a maximum of 10 instruction-operand pairs. The LUT can hold a maximum of 16 sequences. User-provided LUT configuration can be used for “read” type operations over the AHB interface. The LUT should be programmed as per requirements of the flash memory connected and the mode of operation selected, including clock, DDR, SDR, 1-bit, 4-bit, or 8-bit operation. The LUT sequence to be invoked during a read is controlled by the configuration provided in BUFGENCR.

#	Binary Content	Edit
0	0x12 0xd 0x23 0x7 0x34 0x32 0x0 0x7 0x0 0x7 0x0 0x7 0x0 0x7 0x0 0x7	Edit
1	0x11 0x7 0x22 0x7 0x33 0x13 0x44 0x1b 0x55 0x2b 0x66 0x17	Edit
2	0x0 0x7 0x0 0x7 0x0 0x7 0x0 0x7 0x0 0x7 0x0 0x7 0x0 0x7 0x0 0x7	Edit
3		Edit

Figure 80: QSPI - Command Sequences

When the Add button is clicked, a new sequence is created. To configure a sequence, click Edit button. A new window is displayed, and user can create the structure of the sequence. A sequence consists of multiple instruction-operand pairs. To add a new pair, click the Add button. Each pair contains Instruction, Pads and Operand.

#	Instruction	Pads	Operand
0	CMD	Eight Pads	0x11
1	CMD	Eight Pads	0x22
2	MODE	Eight Pads	0x33
3	MODE4	Eight Pads	0x44
4	ADDR_DDR	Eight Pads	0x55
5	MODE2	Eight Pads	0x66

Figure 81: QSPI - Command Sequence

QuadSPI Flash Write Data

Represents an array of 10 structures, each of them containing details of a command and related parameters to be sent to flash memory after Phase 1 and before Phase 2.

#	Address Valid	Configuration ...	Valid Address ...	PAD	CMD	Config/Status ...	Configuration ...
0	<input checked="" type="checkbox"/>	2 Bytes	0x11	QUAD	0x22	0x33	0x0
1	<input type="checkbox"/>	3 Bytes	0x1	QUAD	0x44	0x0	0x1
2	<input checked="" type="checkbox"/>	0 Bytes	0x0	DUAL	0x6f	0xaa	0xda

Figure 82: QSPI - Flash Write Data

A Flash Write Data structure has 12 bytes and contains header, configuration status register address and configuration data. Each of these components have a size of 4 bytes. Header contains Address Valid (1 bit), Configuration Data Size (7 bits), Valid Address Bits (6 bits), PAD (2 bits) and CMD (8 bits).

QuadSPI Problems View

Problems view will indicate if the QuadSPI tool has a problem. Tool will perform the following verifications:

- Value in a textfield has expected format (hexadecimal, decimal, binary)
- Textfield value is not empty
- Textfield value is within the width limit
- Register value does not modify a reserved bitfield

Level	Issue	Origin	Target	Resource
Error	Input value width is invalid! Value width is of length 9 and should be of length 1	QuadSPI: Field value 011111111		QuadSPI Registers: CRC_WNDW_FB
Information	No toolchain project detected			Project

Figure 83: QSPI - Problems View

QuadSPI Export Image

This option allows the user to export the configured QuadSPI binary image. There are two exportable formats available:

- Binary format
- C format

The import/export options can be found on the left side of QuadSPI View.

The dialog box is titled "Flash Port Connection". It contains several sections:

- Port:** A dropdown menu showing "A".
- DLL Modes:** Three checkboxes: "DLL Bypass mode", "DLL SLV Update mode", and "DLL Auto Update mode".
- IPCR Enable Mode:** One checkbox: "IPCR Trigger".
- SFLASH Clock Frequency:** A text input field labeled "Clock Frequency [MHz]" with the value "0x0".
- Buttons:** "Import QuadSPI Image" and "Export QuadSPI Image".

Figure 84: QSPI - General Parameters

When clicking export, a window appears on the screen that will allow user to preview the content of the image and export the code in the desired format.

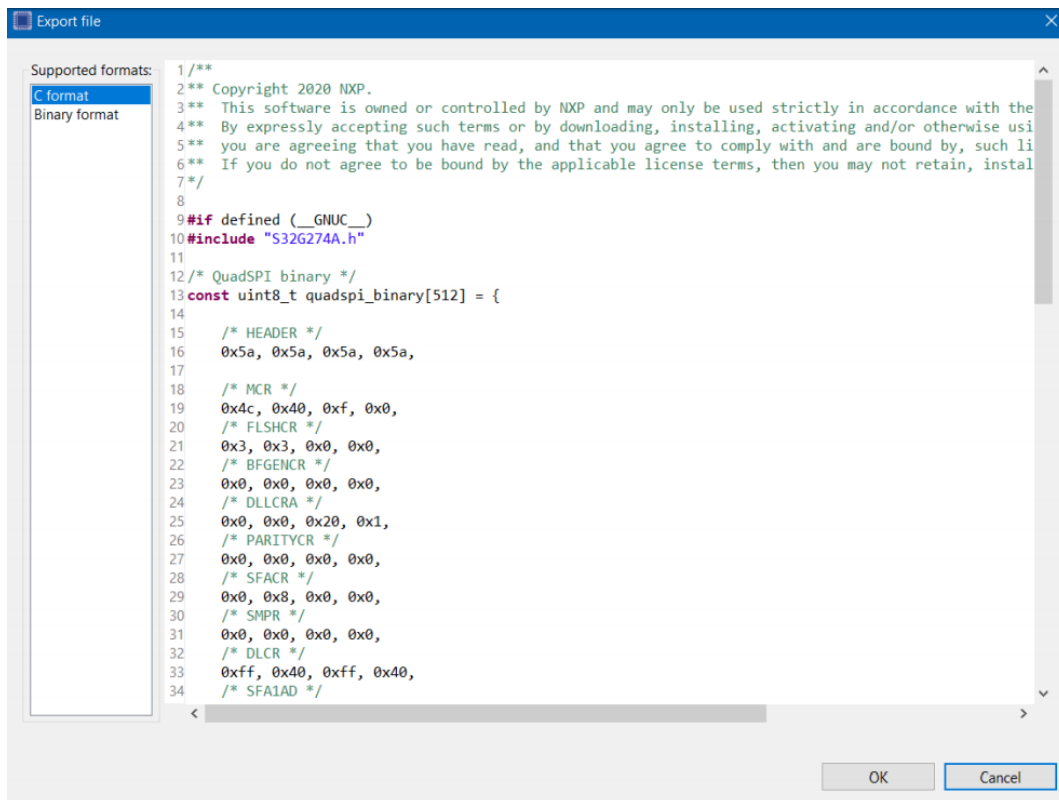


Figure 85: QSPI - Export C format

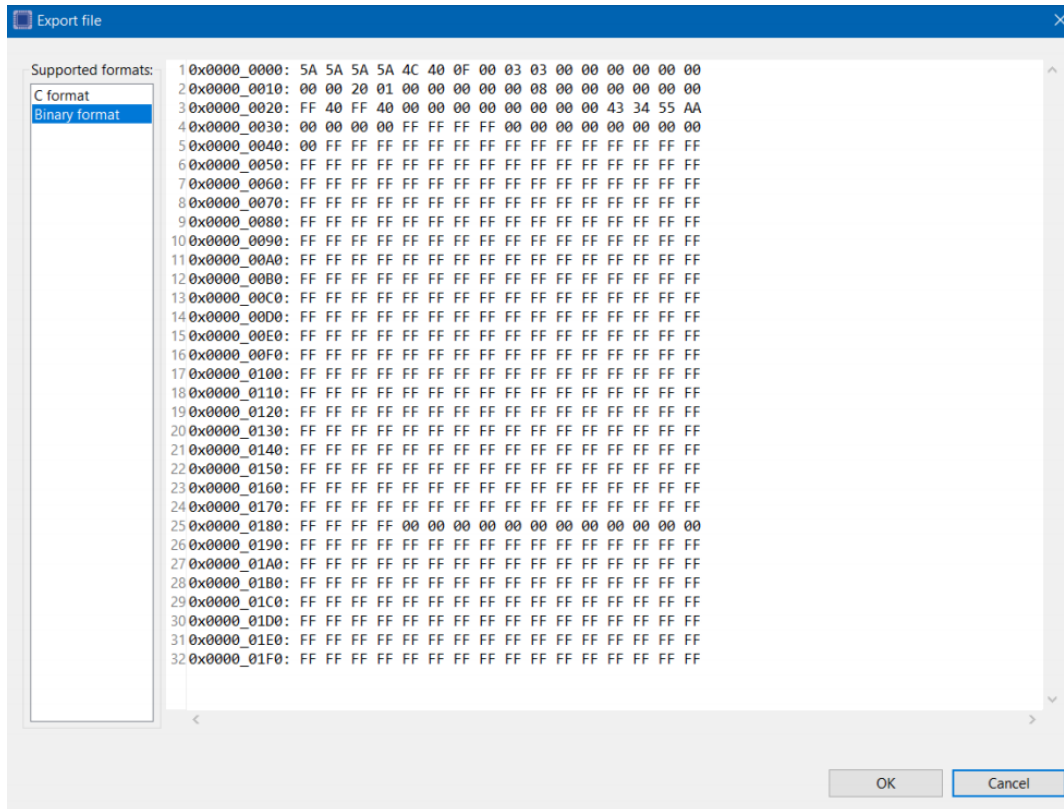


Figure 86: QSPI - Export binary format

After the format is chosen, user can click OK button and a file dialog chooser will appear in order to select the location where the QuadSPI image will be saved.

For some processors, QuadSPI tool contains several default boot images configurations (depending on the device, flash type, clock frequency and mode) that can be imported in the IVT tool for easy configuration. The path where these files can be found is:

mcu_data -> processors -> <processor> -> <SDK version> -> quadspi -> default_boot_images

Note that NXP has only performed functional testing of these configurations in nominal lab settings (room temperature). However, these configurations are aligned to design simulation analysis, and hence are expected to work across PVT. User is however strongly advised to test their system across operating conditions since the behavior is also dependent on QSPI Flash device and PCB layout.

Reset to processor default functionality

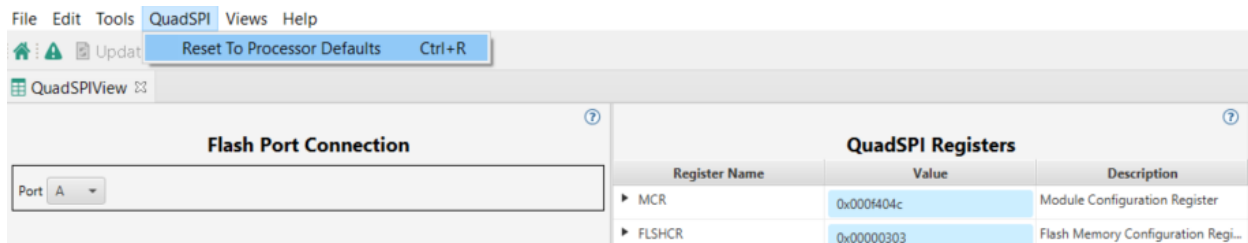


Figure 87 QuadSPI Reset to processor default option

Command line support for QSPI Image generation

All commands listed below require *-HeadlessTool QuadSPI*

Command name	Definition and parameters	Description
Import QuadSPI binary image	-ImportBin	Import the binary QuadSPI image in the current configuration. The path of the imported binary image is expected as argument.
Export QuadSPI image in binary format	-ExportBin	Export the QuadSPI image in binary format. Folder name is expected as argument.
Export QuadSPI image in C format	-ExportC	Export the QuadSPI image in C format. Folder name is expected as argument.
Export all generated files	-ExportAll	Export generated files. Code will be regenerated before export. Includes -ExportBin,-ExportC and in framework -ExportMEX. Folder name is expected as argument.

Improved text field error reporting

- Implemented CRC algorithm (32bit Castagnoli CRC) for ensuring the data integrity of the QuadSPI image, applicable to a limited set of processors

3.7 DDR Tool

DDR Tool help users to execute LPDDR4/DDR3L training, stress test and DDR initialization. Generated code can be used within u-boot as it follows u-boot coding style.

Currently supported memories are:

- LPDDR4 (JEDEC 209-4A)
- DDR3 (JEDEC JESD79-3F)

DDR Tool has two main views:

1. DDR tab - will provide configurable fields for PHY training
2. Validation tab - will run a set of tests in order to ensure stable DDR functionality

Additional tabs that will provide information:

1. Problems view - will display any problems that may occur during DDR initialization
2. Code Preview - will display generated code after DDR training

DDR view

DDR view offers basic configuration for: memory type, frequency, number of channels and others. All options are available as drop-down menus.

DDR Type: LPDDR4
Preset Default Configuration

Device Information

Clock Cycle Freq (MHz)	800 MHz
DDR_CLK frequency	400MHz
Density per channel (Gb)	8G
Number of ROW Addresses	16
Number of Chip Selects used	2
Number of Channels	2
Number of COLUMN Addresses	10
Number of BANK addresses	3
Number of BANKS	8
Total DRAM density (Gb)	32
Bus Width	32
Clock Cycle Time (ns)	1.25

Initial Board Settings

PHY ODT Impedance	60 Ohm
PHY Drive Strength	40 Ohm
PHY Vref Quotient	0x18
PHY Vref	0.206 V
DRAM ODT Impedance	40 Ohm
DRAM Drive Strength	40 Ohm

ECC options

Enable Inline ECC	yes
Note	Only 7/8 of total DDR memory is available when ECC mode is enabled. (3,758,096,384 bytes)
Train 2D	yes
PHY Log Level	Firmware complete
Static Refresh Rate [0.25x]	no
Per Bank Refresh	no
Auto Derating Errata	yes

DQ Swapping

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 88: DDR Tool - DDR view

DDR view helps users to configure a DDR controller depending on the DDR memory placed on the board. The DDR memory may differ by various parameters such as number of chip selects, memory size, or ranks interleaving.

Following options are available:

- **DDR type** - can be LPDDR4 or DDR3L and is selected according to DDR memory on the board
- **Clock Cycle Frequency** - user can select between predefined frequencies for DDR; the following table may help in choosing a frequency:

Clock Cycle Frequency (MHz)	Data Transfer Rate (MT/s)
800	1600
1066	2133
1333	2666
1600	3200

Figure 89: DDR Tool - DDR mapping

DDR_CLK frequency - is the frequency that DDR subsystem should consume; the following table may help to configure correct clock for DDR PLL:

DDR input clock (MHz)	Data Transfer Rate (MT/s)
400	1600
533.3	2133
666.6	2666
800	3200

Figure 90: DDR Tool - DDR_CLK

For more details please consider microcontroller datasheet.

Density per channel - Amount of Gb per channel; information can be extracted from memory datasheet. Below is an example of LPDDR4 memory specifications extracted from datasheet. For memory example, Density per channel is 8Gb:

Configuration		1024M32 (32Gb)
Die per package		4
Device density (per die)		8Gb
Device density (per channel)		8Gb
Configuration		64Mb x 16 DQ x 8 banks x 2 channels x 2 ranks
Number of channels (per die)		1
Number of ranks per channel		2
Number of banks (per channel)		8
Array prefetch (bits) (per channel)		256
Number of rows (per bank)		65,536
Number of columns (fetch boundaries)		64
Page size (bytes)		2048
Channel density (bits per channel)		17,179,869,184
Total density (bits per die)		8,589,934,592
Bank address		BA[2:0]
x16	Row addresses	R[15:0]
	Column addresses	C[9:0]
Burst starting address boundary		64-bit

Figure 91: DDR Tool – Example of LPDDR4 Memory Specifications

- **Number of ROW Addresses** - is calculated based on Density per Channel selection
- **Number of chip selects used** - can be extracted from DDR memory datasheet; in the example is mentioned as 2 chip selects
- **Number of channels** - can be extracted from DDR memory datasheet; in the example is mentioned as 2 channels
- **Number of Column addresses** - can be extracted from DDR memory datasheet; in the example is mentioned as C[9:0]
- **Number of Banks addresses** - can be extracted from DDR memory datasheet; in the example it can be extracted from Number of banks - $2^3=8$ hence Number of Banks addresses is [2:0]
- **Number of Banks** - is calculated based in Number of Banks addresses; in this example there are 8 banks
- **Total DRAM Density** - can be extracted from DDR memory datasheet; in the example it is mentioned as 32 Gb
- **Bus Width** - is calculated based on Number of Channels: $\text{number_of_channels} * 16 \text{ DQ}$
- **Clock Cycle Time** - is time representation of Clock Cycle Frequency selected
- **Train 2D** - option to select if training should perform optional 2D training; 2D training is the DDR reference voltage training and is available for LPDDR4 and DDR4; due to I/O construction DDR3 can't have 2D training as it's voltage reference is fixed; more details on this feature are available in Diag test training
- **PHY log level** - is log level provided by PHY which may contain additional information during training: MR register values, CA training, etc.
- **DQ Swapping** - due to various possibilities of routing of DDR memory, DQ lines can be swapped. General rules that apply to DQ swapping:
 1. All values must be unique
 2. It is allowed swapping at bit lane level. For example: 1 and 4 can be mapped inside [7:0] lane but cannot be mapped inside [15:8] lane.

To be noted that mapping is done at memory level, i.e. upper line is DDR memory and lower line is MCU.

Validation view

Validation view has three embedded tabs:

- **Init** - gives possibility to init ddr and doesn't execute any tests
- **Diags** - runs TxEye and RxEye diag tests; TxEye and RxEye will run init automatically and will output eye diagram; for more information please refer to Diag margin test chapter
- **Operational** - operational test table has DDR tests that will perform basic operations like writing logical '1' and reading for given DDR memory ranges or stress testing a DDR region; each test contains a possibility to set a timeout for how long should a test be running; if target is still running and timeout occurs then a pop-up will appear and test will end.

In order to begin testing please configure in DDR tab required settings, configure performed test and configure connection by choosing connection type and connection settings. Connection type can be one of two types: Serial or S32 Debug. Serial connection is using BootROM communication to upload DDR training data so ensure that serial boot is available.

Current progress can be viewed in **Summary** tab. After test is finished Fail or Pass will be displayed in **Result** column. For Fail a status will be displayed in **Fail reason** column. For current progress and more details on execution of tests script consider opening **Logs** tab.

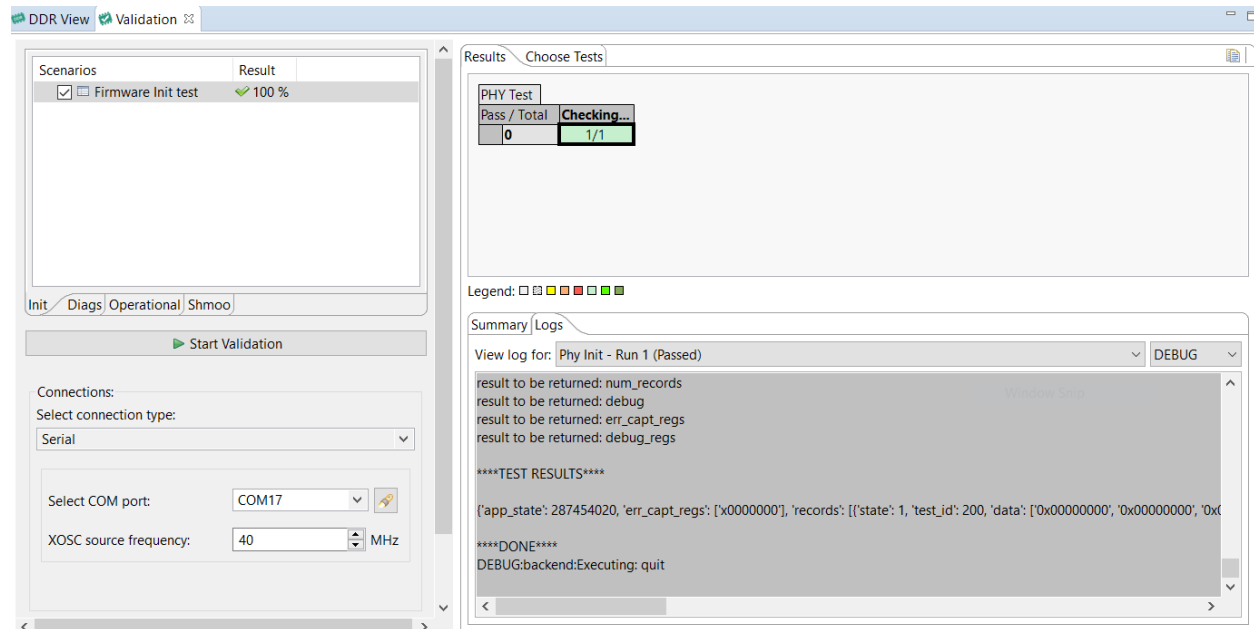


Figure 92: DDR Tool - Validation View

Operational test contains a Stress Test suite that is meant to stress test DDR memory. As all other tests, it contains start address and size. In addition, there is a test timer for how long should stress test run.

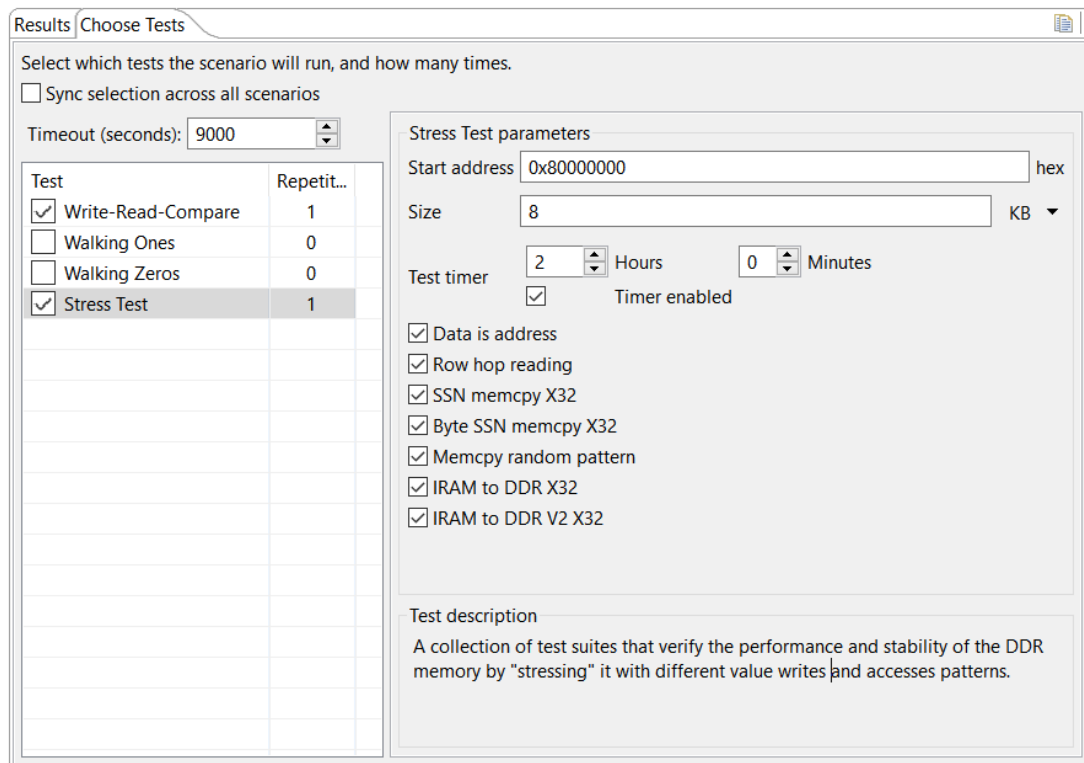


Figure 93: DDR Tool – Operational Tests

Available tests:

- **Data is address** - This test basically programs the source buffer's data words with its address (and immediately read-verifies), then copies (memcpy) the source buffer to a destination buffer and verifies the data transferred correctly. For example, if the source address being written to is "0x80000008", the data value of that address is "0x80000008". Once the source buffer is completely written, the data is transferred to the destination where it is verified. A basic test to ascertain that simple read/writes work.
- **Row hop reading** - This test performs single word reads by hopping from one DRAM row to the next, reading the first column in each row and in each bank. Once all rows are read from the first column, the reads start again from the first row, and begin at the second column. The intent is to perform non-sequential reads and to force pre-charge and activate commands before each read access.
- **SSN memcpy x32** - This test utilizes a custom written memory copy function that issues load and store pair (LDP, STP) instructions, to test the bursting behavior of the DDR interface. This test uses data patterns to help root out simultaneous switching noise (SSN). This test also breaks up the total DDR density into four "banks" or memory regions, where each bank contains a different SSN data pattern. The test uses various stress patterns such as walking ones and walking zeros, and 0xA's followed by 0x5's.
- **Byte SSN memcpy x32** - The purpose of this test is to root out any SSN within byte lanes. It accomplishes this by writing byte-wise patterns to one location and the inverse of each byte to the subsequent location. Each of the four bytes values are equal to one another and the test increments the byte pattern as follows (with the inverse value in brackets: 0x00000000 [0xFFFFFFFF]; 0x01010101 [0xFEFEFEFE]; 0x02020202 [0xFDFDFDFD]; ... 0xFFFFFFFF [0x00000000]).
- **Memcpy random pattern** - This test utilizes a custom written memory copy function that issues load and store pair instructions to test the bursting behavior of the DDR interface. The data pattern used is pseudo-random. This test also breaks up the total DDR density into four "banks" or memory regions, where each bank contains a different "seed" for each pseudo-random data pattern.
- **IRAM to DDR x32** - The purpose of this test is to root out any SSN and isolates the DDR read and write accesses by using the Internal RAM (IRAM) as an intermediate data storage location. This test moves data from DDR to IRAM and then from IRAM to a different DDR location, then compares DDR location 1 and location 2. This test is similar to IRAM_to_DDRv1 test (described next), but instead transfers the data 1000 times per test to ensure that the data never changes to root out random glitches. Also, the test uses various data patterns to root out SSN.
- **IRAM to DDR x32 v2** - The purpose of this test is to root out any SSN and isolates the DDR read and write accesses by using the IRAM as an intermediate data storage location. This test moves data from DDR to IRAM and then from IRAM to a different DDR location, then compares DDR location 1 and location 2.

Diag Margin Test

Validation view has two set of tests scenarios in Diags tab: Tx Eye and Rx Eye.

- **TxEye** test collects the transmit eye associated with a specific byte and bit. The eye is measured by running traffic while varying the DRAM VREF and Phy's TxDq delay settings. The test records

how many errors occur at each delay and voltage setting and returns a matrix of encoded error counts.

- **RxEye** test collects the receive eye associated with a specific byte and bit. The eye is measured by running traffic while varying the Phy's VREF and RxClkDly delay settings. The test records how many errors occur at each delay and voltage setting and returns a matrix of encoded error counts.

Chose Test tab has two dropdown options available to configure Rx/Tx Eye test:

- Byte lane
- Bit lane

After configuring connection, **Start Validation** can be performed. After successful run, eye diagram will be displayed in Result tab.

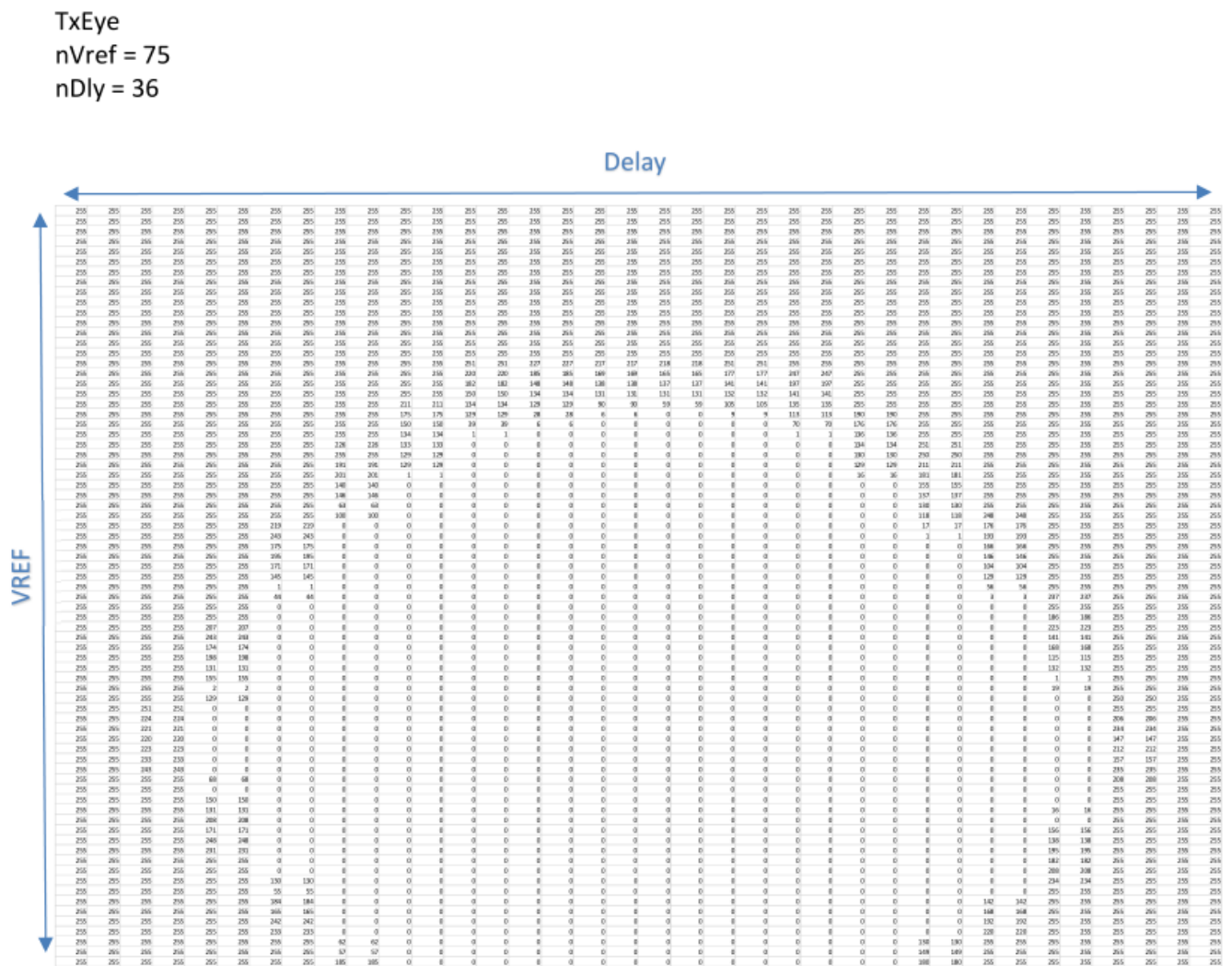


Figure 94: DDR Tool - Example Messageblock ErrorCount Output

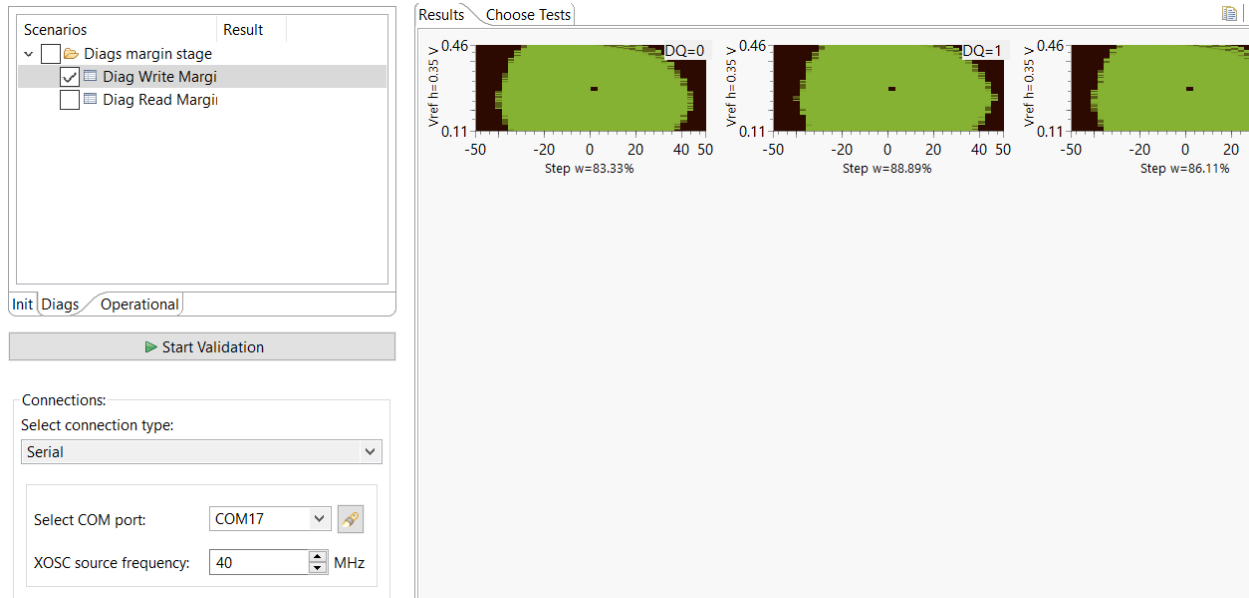


Figure 95: DDR Tool – Start Validation

DDR Code Preview

DDR Tool generates source code which can be incorporated into an application to initialize the DDR subsystem. The source code uses an *array-based* format and is generated after validation on target is finished.

1. Configure DDR controller from DDR View
2. Select Connection type and configure parameters
3. Select test, e.g. Firmware Init test
4. Click on Start Validation
5. Wait until execution finishes
6. The generated code is shown in the Code Preview tab on the right window. It displays all generated files, each in its own tab

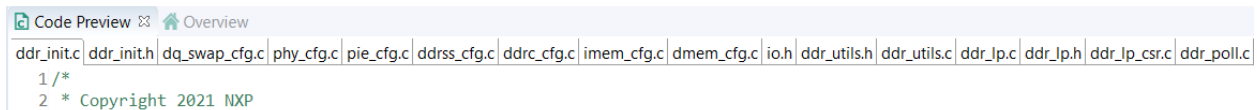


Figure 96: DDR Tool – Generated files

Generated Files:

- **ddr_init.c** – main source file and the entry point of ddr initialization. The method which needs to be called from the user application in order to initialize the DDR subsystem is `ddr_init()`. It configures the DDRC controller based on the selected UI values from DDR View and executes the PHY training algorithm

```

33/* Main method needed to initialize ddr subsystem. */
34uint32_t ddr_init(void)
35{
36    uint32_t ret = NO_ERR;
37    size_t i = 0;
38
39    init_image_sizes();
40
41    for (i = 0; i < ddrss_config_size; i++) {
42        /* Init DDR controller based on selected parameter values */
43        ret = ddrc_init_cfg(&configs[i]);
44        if (ret != NO_ERR)
45            return ret;
46
47        /* Setup AXI ports parity */
48        ret = set_axi_parity();
49        if (ret != NO_ERR)
50            return ret;
51
52        /* Init PHY module */
53        ret = execute_training(&configs[i]);
54        if (ret != NO_ERR)
55            return ret;
56
57        /* Execute post training setup */
58        ret = post_train_setup();
59        if (ret != NO_ERR)
60            return ret;
61    }
62    return ret;
63}

```

Figure 97: DDR Tool – Generated code example

- **ddr_init.h** – main header file, contains method prototypes and global variable definitions
- **dq_swap_cfg.c** – contains DQ swapping options, generated based on selection from UI

DQ Swapping

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 98: DDR Tool – DQ Swapping example

- **phy_cfg.c** – contains phy module initialization sequence, based on the selected parameters from UI
- **pie_cfg.c** – contains PIE initialization sequence
- **ddrss_cfg.c** – initializes global data structure used in ddr_init() method
- **ddrc_cfg.c** – DDRC controller initialization sequence, generated based on selected values in UI (e.g. Clock Cycle Freq (MHz))
- **imem_cfg.c** – IMEM initialization sequence, covering both training stages:
 - struct regconf imem_1d[];
 - struct regconf imem_2d[];
- **dmem_cfg.c** – DMEM initialization sequence:
 - struct regconf dmem_1d[]; - mandatory
 - struct regconf dmem_2d[]; - optional, depending on the selected value from UI

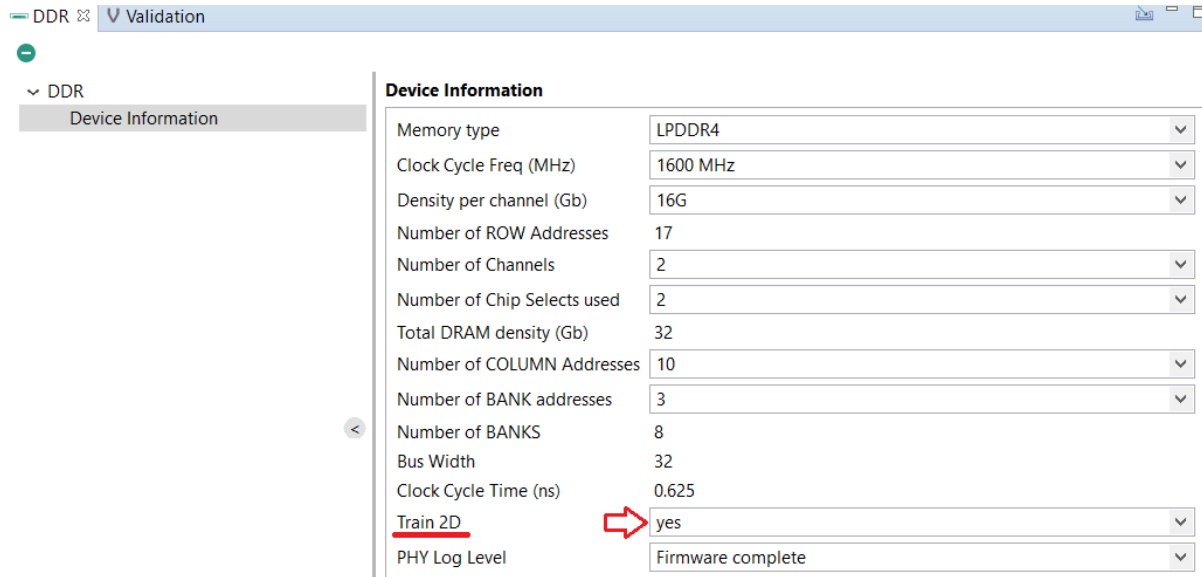


Figure 99: DDR Tool – Train 2D option

- **io.h** – io access utility methods
- **ddr_utils.c** – utility methods used by the phy training algorithm
- **ddr_utils.h** – utility methods header file
- **ddr_lp.c** – low power utility methods for transitioning the DDR subsystem from normal more to low power (and back); see more details in Low Power section below.
- **ddr_lp.h** – low power utility methods leader file
- **ddr_lp_csr.c** – list of registers which need to be stored/restored while transitioning the DDR system
- **ddr_poll.c** - contains an example of polling method which must be periodically called from user application in case Auto Derating Errata option is enabled. The method will revert changes made by errata workaround if they are no longer needed.

A subset of the files above is dynamically generated based on the output log file obtained from the target. If the log file is missing, a custom message is displayed, and code generation is stopped.

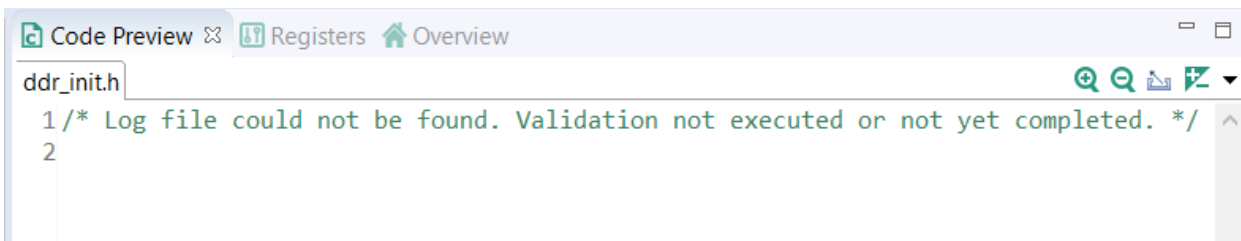


Figure 100: DDR Tool – Log file is missing example

Code generation is supported for Firmware Init and Operational scenarios. Running Read Margin Test or Write Margin Test scenarios will display the eye diagram but will not generate initialization code in Code Preview. A custom message is displayed instead:

```
1 /* Code generation is not supported for currently selected scenario. */
2
```

Figure 101: DDR Tool – Code generation not supported example

Other important capabilities:

- Support for DDR max frequency 1600 MHz, data rate 3200 MT/s

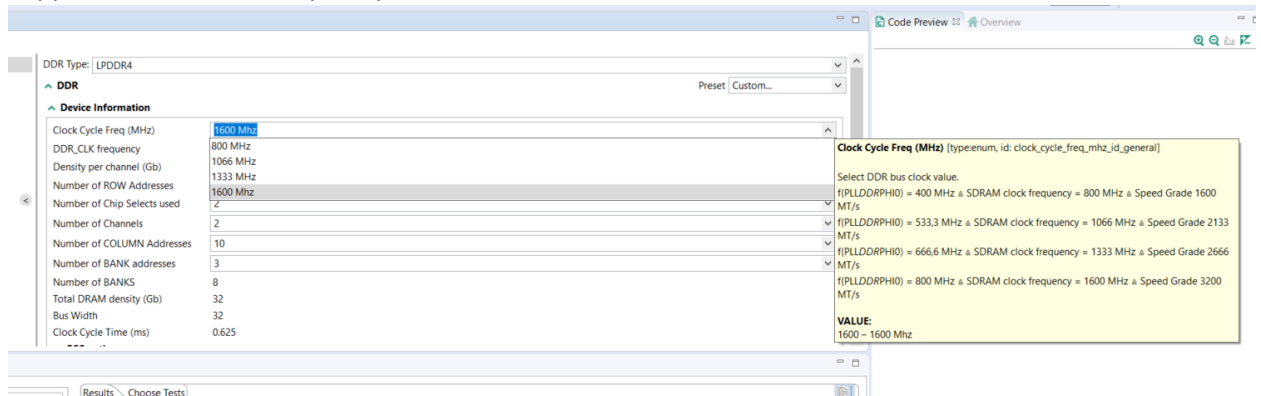


Figure 102: DDR Tool – Max frequency 1600MHz

- LPDDR4 Inline Error Correcting Code – includes the configuration of ECC scrubber

Bus Width 32

Clock Cycle Time (ms) 1.25

ECC options

Enable Inline ECC

Note Only 7/8 of total DDR memory is available when ECC mode is enabled. (3,758,096,383 bytes)

Train 2D

PHY Log Level

DQ Swapping

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4

Figure 103: DDR Tool – LPDDR4 Inline ECC

- Support for 2020.06 DDR PHY Firmware
- Code size optimization for LPDDR4 configuration code
- Static Refresh Rate [0.25x] - Disables automatic refresh rate and adjust register settings for static refresh rate of 0.25X. Auto Derating Errata is not applicable if this option is enabled.

- Auto Derating Errata – If the system is hot or cold prior to enabling derating, temperature update flag (TUF) might not be set in MR4 register, causing incorrect refresh period and timing parameters being used (tRCD, tRAS, tRP, tRRD). Software workaround requires reading MR4 during initialization, disabling auto-derating and adjusting timing parameters, if necessary. When this option is enabled, poll_derating_temp_errata method needs to be periodically called from user application, in order to monitor TUF and enable auto-derating logic when possible. In case DDR traffic can be halted, timing parameters can also be restored (see traffic_halted parameter). This option is not applicable if Static Refresh Rate is used

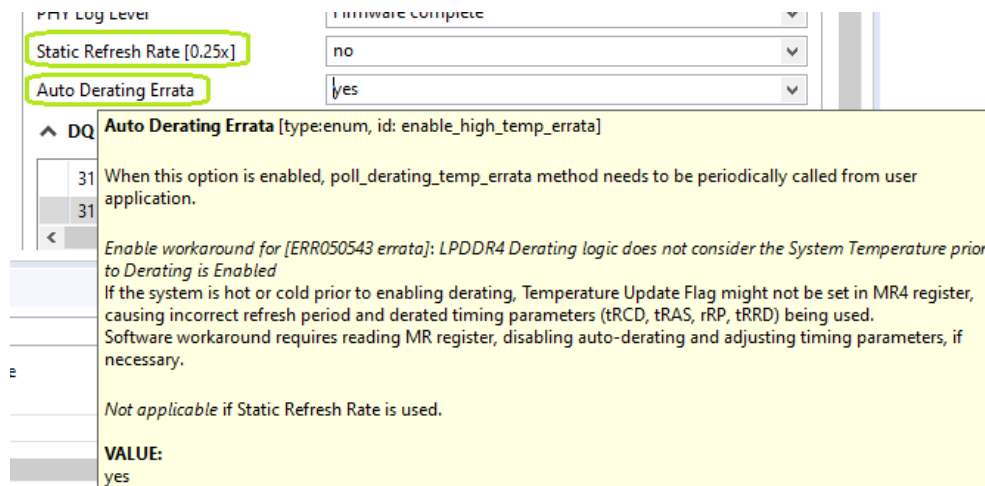


Figure 104: DDR Tool – Static Refresh Rate & Auto Derating Errata

- UI option that allows user to enable Per Bank Refresh (PBR) mechanism:

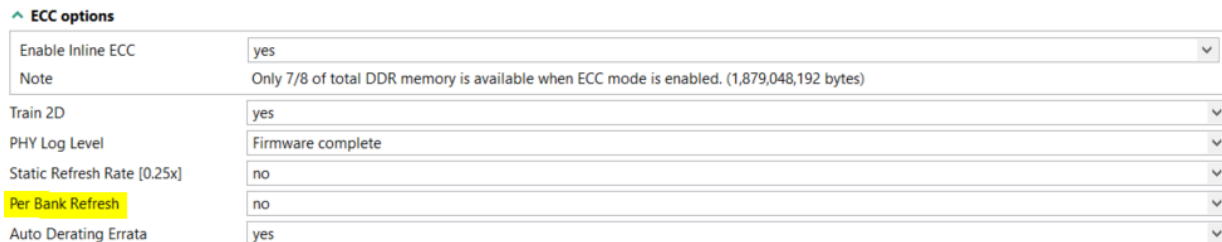


Figure 105: DDR Tool - Per Bank Refresh option

- Per bank refresh (PBR) is an alternative refresh scheme in which the refresh is executed one memory bank at a time, allowing the other banks to be used for read/write accesses, thus increasing performance. If Per bank refresh is disabled, all memory accesses are stopped until all banks finish the refresh job, resulting in a longer period of inactivity (lower performance).
- There is a known issue (ERR050336: "LPDDR4 Per-bank Refresh Issued to the Same Bank within tRFCpb) which applies for memories with density per channel higher than 4Gb. As such, DDR Tool will only support PBR if density per channel value is <= 4Gb.
- Static Refresh Rate and Per Bank Refresh can't be enabled simultaneously (setting either of them to "yes" will disable the UI element of the other).

- DDR Tool generates a html report of the training parameters and mode registers settings that can be stored by the customer along with the generated code, using Update Code:

General Information

DDR Type : LPDDR4

Firmware version : 2020_06

UI Configuration

^ Device Information :

Clock Cycle Freq (MHz) : 800
DDR_CLK frequency : 400MHz
Density per channel (Gb) : 8G
Number of ROW Addresses : 16
Number of Chip Selects used : 2
Number of Channels : 2
Number of COLUMN Addresses : 10
Number of BANK addresses : 3
Number of BANKS : 8
Total DRAM density (Gb) : 32
Bus Width : 32
Clock Cycle Time (ms) : 1.25

^ Board settings :

PHY ODT Impedance : 60
PHY Drive Strength : 40
PHY Vref : 0x11
Vref : 0.146 V
DRAM ODT Impedance : 40
DRAM Drive Strength : 40

^ ECC options :

Enable Inline ECC : yes
Note : Only 7/8 of total DDR memory is available when ECC mode is enabled. (3,758,096,384 bytes)

Train 2D : yes
PHY Log Level : firmwareComplete
Static Refresh Rate [0.25x] : no
Per Bank Refresh : no
Auto Derating Errata : yes

^ DQ Swapping :

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Mode Registers

#	value	OP [7]	OP [6]	OP [5]	OP [4]	OP [3]	OP [2]	OP [1]	OP [0]
MR1	0x24	RPST = 0x0	nWR (for AP) = 0x2			RD-PRE = 0x0	WR-PRE = 0x1	BL = 0x0	
MR2	0x12	WR Lev = 0x0	WLS = 0x0	WL = 0x2			RL = 0x2		
MR3	0x33	DBI-WR = 0x0	DBI-RD = 0x0	PDDS = 0x6			PPRP = 0x0	WR PST = 0x1	PU-CAL = 0x1
MR4	0x00	TUF = 0x0	Thermal Offset = 0x0		PPRE = 0x0	SR Abort = 0x0	Refresh Rate = 0x0		
MR11	0x66	Reserved = 0x0	CA ODT = 0x6			Reserved = 0x0	DQ ODT = 0x6		
MR12	0x4D	CBT Mode for Byte Mode = 0x0	VR-CA = 0x1	Vref(CA) = 0xd					
MR13	0x8	FSP-OP = 0x0	FSP-WR = 0x0	DMD = 0x0	RRO = 0x0	VRCG = 0x1	VRO = 0x0	RPT = 0x0	CBT = 0x0
MR14	0x4F	Vref(DQ) = 0x0	VR(DQ) = 0x1	RFU = 0xf					
MR16	0x00	PASR Bank Mask = 0x0							
MR17	0x00	PASR Segment Mask = 0x0							
MR22	0x6	ODTD for x8_2ch(Byte) mode = 0x0		ODTD-CA = 0x0	ODTE-CS = 0x0	ODTE-CK = 0x0	SOC ODT = 0x6		
MR24	0x00	TRR Mode = 0x0	TRR Mode BAn = 0x0			Unlimited = 0x0	MAC Value = 0x0		

Figure 106: DDR Tool - Detailed report

- “Board settings” section was added in DDR View allowing configuration of On Die Termination (ODT) impedances, drive strengths and PHY Vref values, depending on the client’s custom board layout:
 - PHY ODT Impedance
 - PHY Drive Strength
 - PHY Vref Quotient
 - PHY Vref
 - DRAM ODT Impedance
 - DRAM Drive Strength

^ Board settings

PHY ODT Impedance	60 Ohm	▼
PHY Drive Strength	40 Ohm	▼
PHY Vref Quotient	0x18	
PHY Vref	0.206 V	
DRAM ODT Impedance	40 Ohm	▼
DRAM Drive Strength	40 Ohm	▼

Figure 107: On Die Termination (ODT) impedance

- PHY ODT Impedance - On Die Termination impedance value (in Ohms) used by PHY during reads
- PHY Drive Strength - Select the driver impedance value (in Ohms) used by PHY during writes for all DBYTE drivers (DQ/DM/DBI/DQS)
- PHY Vref Quotient - Should be programmed with the Vref level to be used by the PHY during reads. The units of this field are a percentage of VDDQ according to the following equation:

$$\text{PHY Vref} = \text{VDDQ} * \text{PhyVrefQuotient}[6:0] / 128$$
- PHY Vref – Vref value calculated with above formula, based on PHY Vref Quotient value.
- DRAM ODT Impedance – Select On Die Termination impedance value (in Ohms) used by DRAM during write

- DRAM Drive Strength - Select the driver impedance value (in Ohms) used by DRAM during reads
- DDR Tool can generate code with reference methods for reading or writing to a mode register (MR).
- DDR Tool Log View is now available when running Margin tests, displaying real-time information about the test progress:

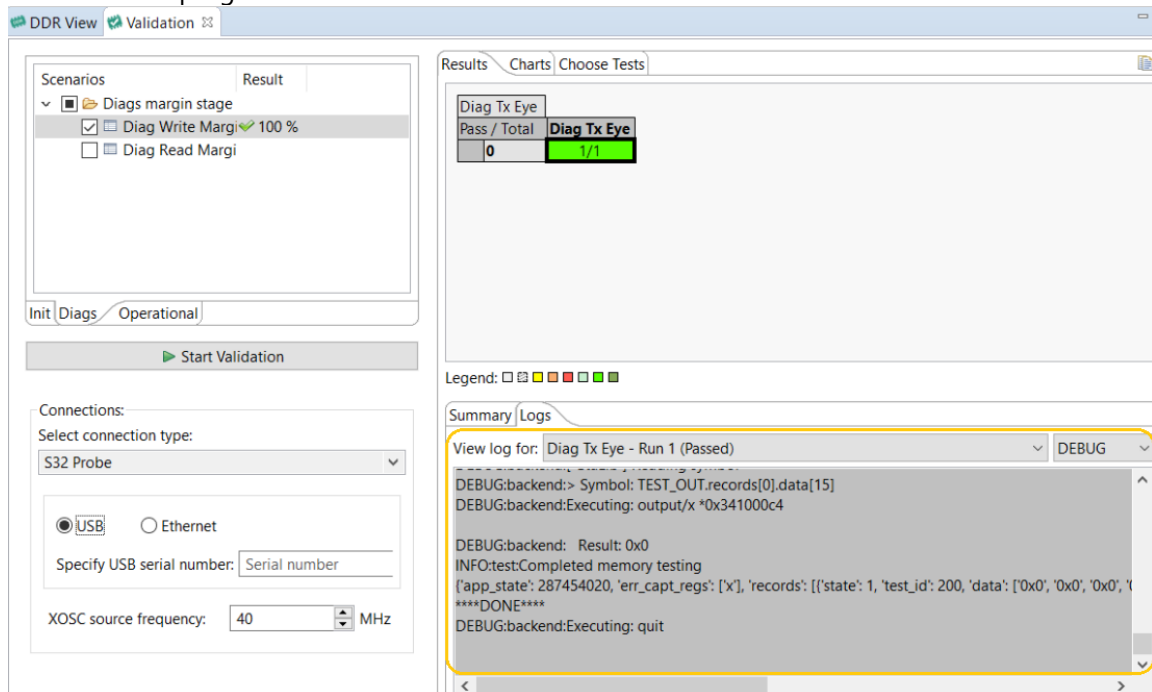


Figure 108: DDR Tool - Log view

- New Shmoo test category introduced in the DDR Tool
Shmoo tab contains a suite of tests designed to experiment with different Initial Board Setting values and display a map of the results. The list of possible values depends on the currently selected memory type, but all memory types offer the following Shmoo tests:
 - **Write test** - PHY Drive Strength and DRAM ODT are varied across the list of possible values. Consecutive trainings are executed, displaying the results for each combination of values:

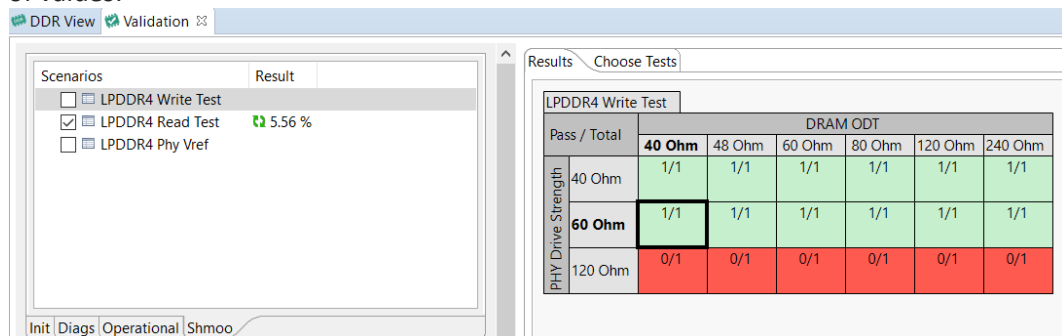


Figure 109: DDR Tool - Shmoo, Write test

- **Read test** - PHY ODT and DRAM Drive Strength are varied across the list of possible values. Consecutive trainings are executed, displaying results for each combination of values:

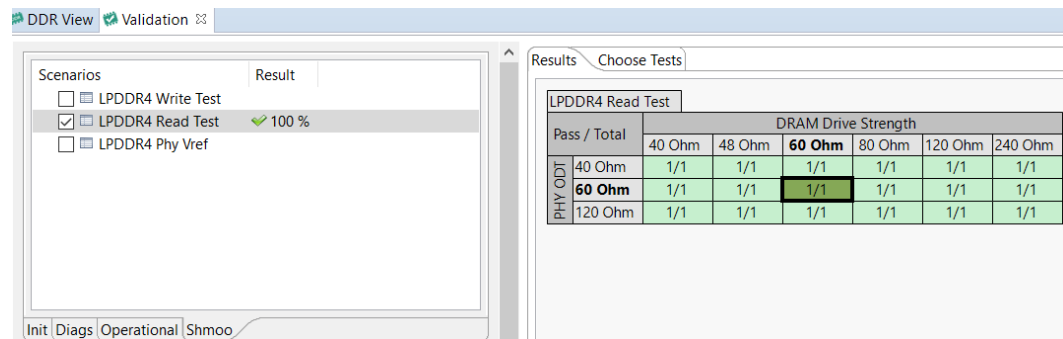


Figure 110: DDR Tool - Shmoo, Read test

- **Phy Vref** - the purpose of this test is to get an optimized Phy Vref value which will be used as starting value for the consequent 1D trainings. This is useful when the default value provided for Phy Vref is not good enough for the custom board to have 1D training passing because if 1D training is already passing, Phy Vref is further optimized during 2D training. The test will display as result the value of Phy Vref Quotient corresponding to the optimized Vref value:

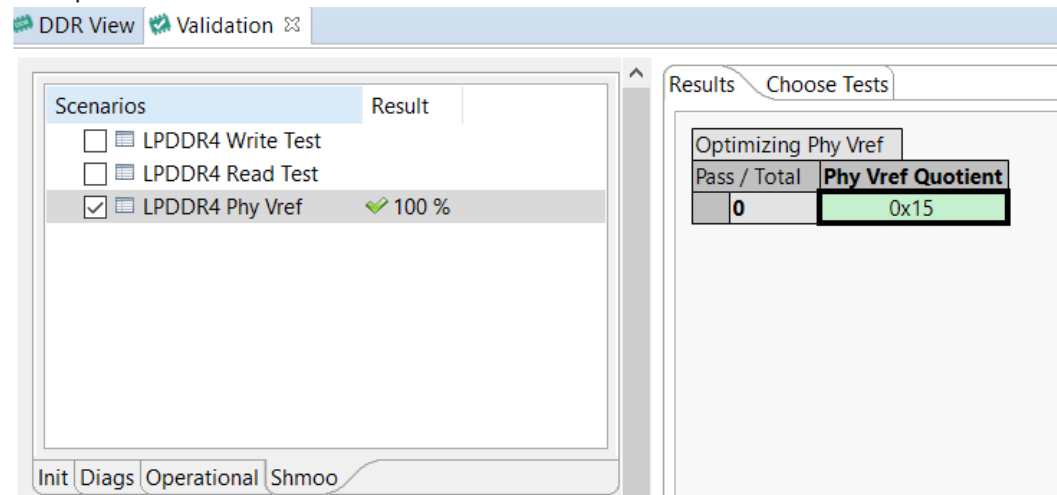


Figure 111: DDR Tool - Shmoo, Phy Vref test

- Added DDR Tool Synopsis firmware version into the generated code

```

46
47 /* Compute the number of elements in the given array */
48 #define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))
49
50 #define FIRMWARE_VERSION "2020_06"
51

```

Figure 112: DDR Tool - Firmware version

- Improved DDR log when tests fail by adding brief human-readable information about the failure

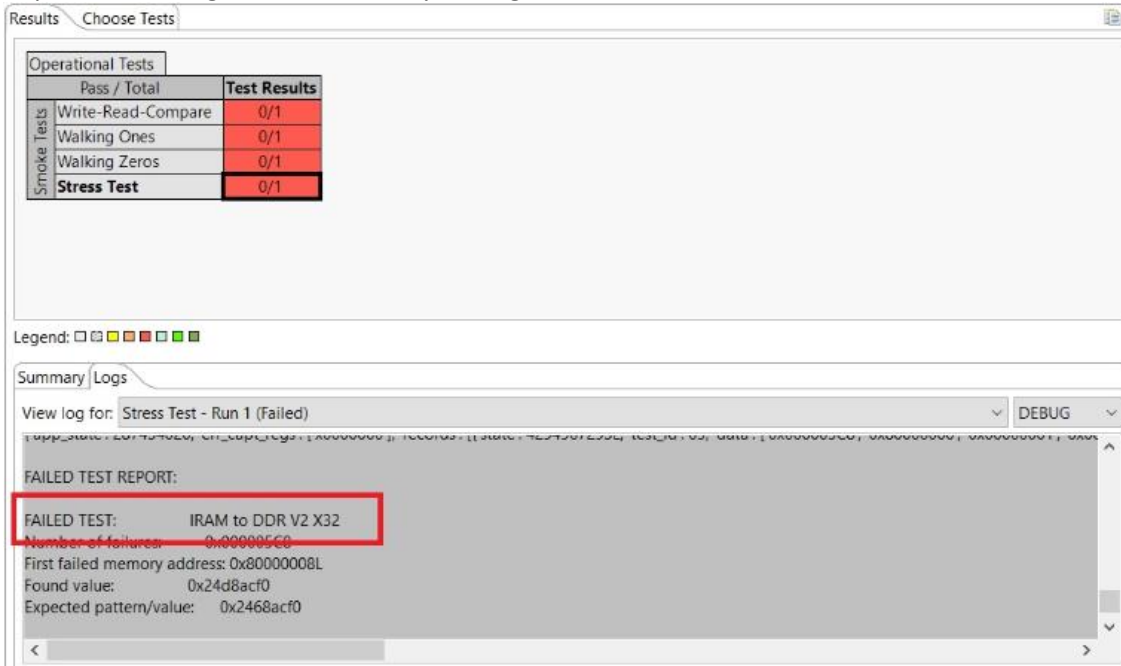


Figure 113: DDR Tool - DDR log with info

- DDR Tool generated code provides reference code to transition the DDR subsystem from normal mode to low power mode (retention mode) as well as the other way around. Low power related methods can be found in:
 - ddr_lp.h, ddr_lp.c
 - utility methods for storing and loading configuration registers to / from a given address
 - ddr subsystem transitioning methods: ddrss_to_io_retention_mode and ddrss_to_normal_mode
 - ddr_csr_lp.c
 - the list with registers which need to be stored / restored
 - generated dynamically, based on the memory type
- Added new UI option to select generated code format. Code Preview content will vary depending on the selected format.

Auto Derating Errata: yes

▼ DQ Swapping

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6

▼ Code Generation

Code format: U-boot

- U-boot
- MISRA
- ATF

Figure 114: DDR Tool - Generate code format

- MISRA-C 2012 compliance for DDR generated code with limited list of exceptions (deviations) which are documented in the files they occur in.

```

31 /*
32 * MISRA-C:2012 violations
33 *
34 * [MISRA 2012 Advisory Directive 4.9]:
35 * A function should be used in preference to a function-like macro where they
36 * are interchangeable.
37 * Violated since functions-like macro are used to generate more efficient
38 * code.
39 *
40 *
41 * [MISRA 2012 Advisory Rule 1.2]:
42 * Language extensions should not be used.
43 * Violated since read/write functions-like macro use assembly instructions for
44 * memory accesses.
45 *
46 *
47 * [MISRA 2012 Advisory Rule 11.4]:
48 * A conversion should not be performed between a pointer to object and an
49 * integer type.
50 * Violated since conversion is required in this particular case for the
51 * read/write memory accesses.
52 *
53 *
54 * [MISRA 2012 Advisory Rule 8.7]:
55 * Functions and objects should not be defined with external linkage if they
56 * are referenced in only one translation unit.
57 * Violated since ddr_init function is provided as API, to be used from
58 * external files.
59 *
60 *
61 * [MISRA 2012 Advisory Rule 15.5]:
62 * A function should have a single point of exit at the end.
63 * Violated to simplify code logic for ddr_init and execute_training functions.
64 */
65
66 #include "ddr_init.h"

```

Figure 115: DDR Tool - MISRA compliant code

```

/*
 * Check if 2d training images have been initialized before executing
 * the second training stage.
 */
if ((config->imem_2d_size > 0U) && (config->dmem_2d_size > 0U)) {
    /* Load 2d imem image */
    writel(UNLOCK_CSR_ACCESS, MICROCONT_MUX_SEL);
    ret = load_phy_image(IMEM_START_ADDR, config->imem_2d_size,
                        config->imem_2d);
    if (ret != NO_ERR) {
        return ret;
    }
    writel(LOCK_CSR_ACCESS, MICROCONT_MUX_SEL);

    /* Load 2d dmem image */
    writel(UNLOCK_CSR_ACCESS, MICROCONT_MUX_SEL);
    ret = load_phy_image(DMEM_START_ADDR, config->dmem_2d_size,
                        config->dmem_2d);
    if (ret != NO_ERR) {
        return ret;
    }
    writel(LOCK_CSR_ACCESS, MICROCONT_MUX_SEL);

    /* Configure PLL optimal settings */
    set_optimal_pll();

    writel(LOCK_CSR_ACCESS, MICROCONT_MUX_SEL);
    writel(APBONLY_RESET_STALL_MASK, APBONLY_MICRORESET);
    writel(APBONLY_STALL_TO_MICRO_MASK, APBONLY_MICRORESET);
    writel(APBONLY_MICRORESET_CLR_MASK, APBONLY_MICRORESET);

    ret = wait_firmware_execution();
    if (ret != NO_ERR) {
        return ret;
    }
}

```

Figure 116: DDR Tool - MISRA deviations

- ATF (Arm Trusted Firmware) format for DDR generated code

```

123  /*
124  * Check if 2d training images have been initialized before executing
125  * the second training stage.
126  */
127  if (config->imem_2d_size > 0U && config->dmem_2d_size > 0U) {
128      /* Load 2d imem image */
129      mmio_write_32(MICROCONT_MUX_SEL, UNLOCK_CSR_ACCESS);
130      ret = load_phy_image(IMEM_START_ADDR, config->imem_2d_size,
131                          config->imem_2d);
132      if (ret != NO_ERR)
133          return ret;
134      mmio_write_32(MICROCONT_MUX_SEL, LOCK_CSR_ACCESS);
135
136      /* Load 2d dmem image */
137      mmio_write_32(MICROCONT_MUX_SEL, UNLOCK_CSR_ACCESS);
138      ret = load_phy_image(DMEM_START_ADDR, config->dmem_2d_size,
139                          config->dmem_2d);
140      if (ret != NO_ERR)
141          return ret;
142      mmio_write_32(MICROCONT_MUX_SEL, LOCK_CSR_ACCESS);
143
144      /* Configure PLL optimal settings */
145      set_optimal_pll();
146
147      mmio_write_32(MICROCONT_MUX_SEL, LOCK_CSR_ACCESS);
148      mmio_write_32(APBONLY_MICRORESET, APBONLY_RESET_STALL_MASK);
149      mmio_write_32(APBONLY_MICRORESET, APBONLY_STALL_TO_MICRO_MASK);
150      mmio_write_32(APBONLY_MICRORESET, APBONLY_MICRORESET_CLR_MASK);
151
152      ret = wait_firmware_execution();
153      if (ret != NO_ERR)
154          return ret;
155  }

```

Figure 117: DDR Tool - ATF compliant

- Integrated errata ERR050759 workaround for multi-rank configurations
- Added sanity check to Firmware Init scenario. Basic write/read memory accesses are performed after executing the training and the result is reported by the tool:

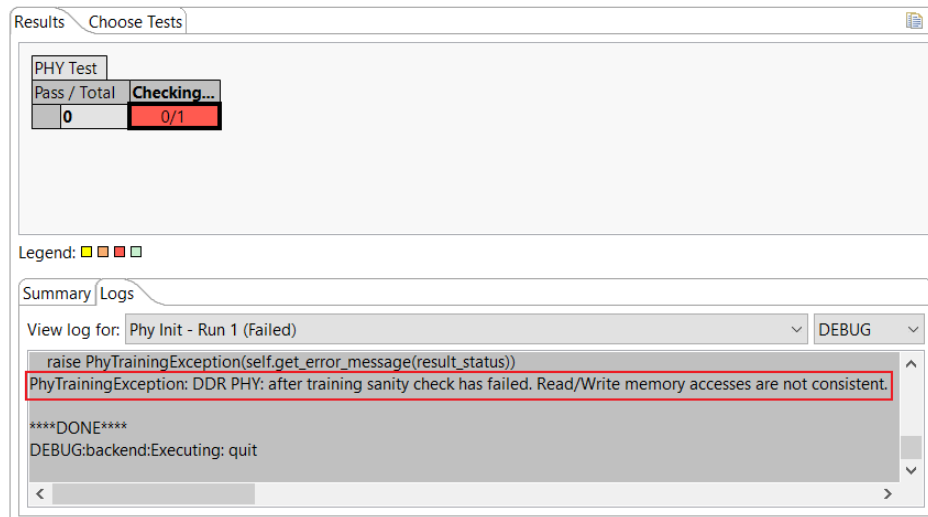


Figure 118: DDR Tool - Firmware init, sanity check

- Improved readability of the generated code by using macro definitions
- Included center coordinates (Vref, delay) in Margin tests eye diagrams

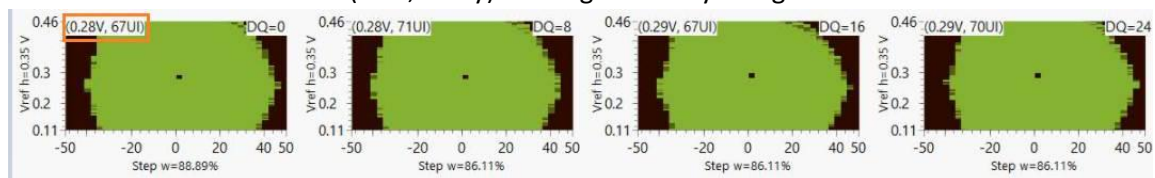


Figure 119: DDR Tool - Center coordinates (Vref, delay) in Margin tests

- Enabled 3032MT/s configuration option on platforms where this configuration is stable

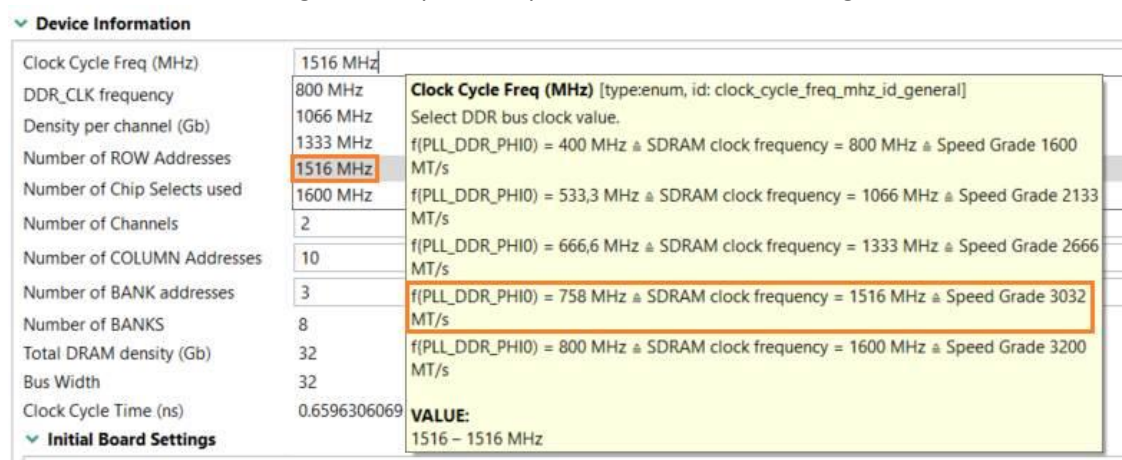


Figure 120: DDR Tool - Option for 3032MT/s configuration

- Eye diagrams can be now exported in ASCII format

```

4 struct dq_info {
44  uint8_t chip_select;
45  double center_vref;
46  uint8_t center_tap_delay;
47  uint8_t dq_index;
48  uint8_t byte_index;
49  uint8_t bit_index;
50  double max_vref_opening;
51  double max_delay_width;
52};
53
54 struct diag_eye {
55  double vrefs[VREF_NUMBER];
56  int delays[DELAY_NUMBER];
57  uint8_t data_eye[VREF_NUMBER][DELAY_NUMBER];
58};
59
60 struct diag_result {
61  const struct dq_info *dq_info_result;
62  const struct diag_eye *diag_eye_result;
63};
64
65 /*
66 * | CS: 0 | Vref: 0.176847 V | TapDelay: 15 UI |
67 * |-----|
68 * | DQ: 0 (Byte: 0 | Bit: 0) |
69 * |-----|
70 * | H: 0.274516 V | W: 83.33 %UI |
71 * |-----|
72 * VREF(V) \ Strobe delay (UI) | -6 | -5 | -4 | -3 | -2 | -1 | 0 | 1 | 2 |
73 * |-----|
74 * | 0.052 | # | # | # | # | # | # | # | # |
75 * | 0.056 | # | # | # | # | # | # | # | # |
76 * | 0.060 | # | # | # | # | # | # | # | # |
77 * | 0.064 | # | # | # | # | # | # | # | # |
78 * | 0.068 | # | # | # | # | # | # | # | # |
79 * | 0.072 | # | # | # | # | # | # | # | # |
80 * | 0.076 | # | # | # | # | # | # | # | # |
81 * | 0.080 | # | # | # | # | # | # | # | # |

```

Figure 121: DDR Tool - Eye diagram in ASCII format

- Updated content retention routines to provide improved CA margins across temperatures after standby exit
- UI option to control whether CA Vref training is executed – static value can be configured if the user chooses to disable the training step.

DRAM ODT Impedance	40 Ohm
DRAM Drive Strength	40 Ohm
CA Vref Training	yes

DRAM Drive Strength	40 Ohm
CA Vref Training	no
VREF CA Range:	Range 1
Range 1:	27.2%

Figure 122 CA Vref training disabled example with static value configured

- Improved COM port scanning action by adding a progress bar in Validation View to inform user about the status

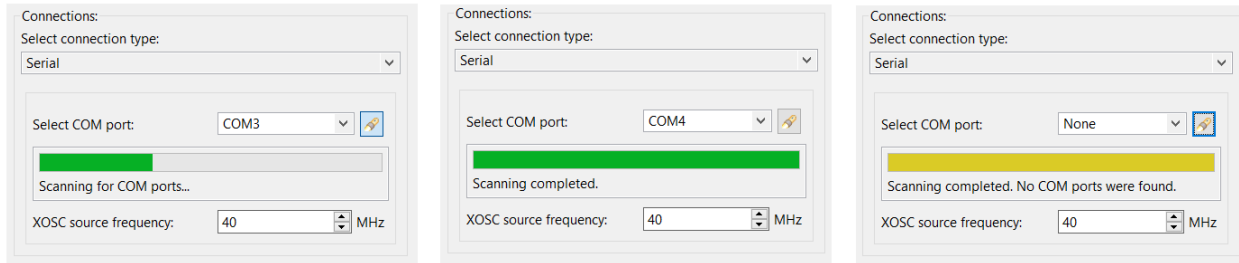


Figure 123 COM port scanning progress

- Enable serial connection for secured parts

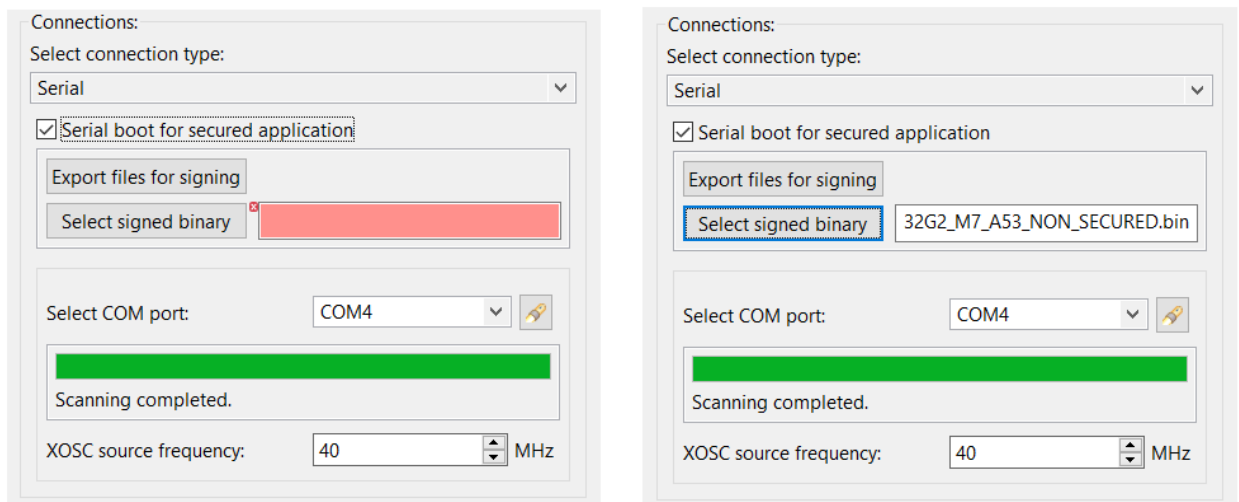


Figure 124 Secured serial boot in 2 steps

- Added possibility to disable the PMIC watchdog before running DDR training



Figure 125 Select DCD image for disabling PMIC watchdog

- Added support for Modified and Legacy modes for the refresh command timing constraints

Figure 126 Refresh command timing constraints

- ▼ Debug

Save debug messages

yes

Start address

0x34410000

Figure 127 Store messages option in DDR View

- [illegible]

Figure 128 Decoded debug messages

- Removed Refresh Command Constraints option from DDR View. Default configuration is working both on Legacy and on Modified mode devices because 2x and 4x refresh rates are disabled by default in MR13 OP[4].
- Add option to synchronize input test parameters across the selected tests
- Add size constraints for Operational test parameters

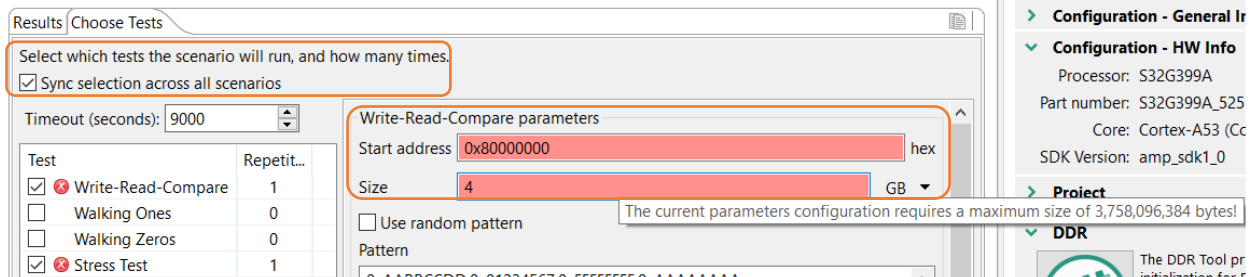


Figure 129 Size validation synchronized across scenarios

- Add option to bypass the default XOSC configuration, allowing the user to configure the oscillator with custom settings. The custom configuration must be done before launching the DDR validation.



Figure 130 XOSC bypass in DDR View

- Improvements on DDR PHY impedance calibration sequence

- Enabled Overlay Charts feature
 - Offers an interactive representation of the Diag margin test results, displaying the overlapped eye contours.
 - The chart can display all or any subset of the available data lines. This can be configured at line level through the interactive legend or at byte level using checkboxes.

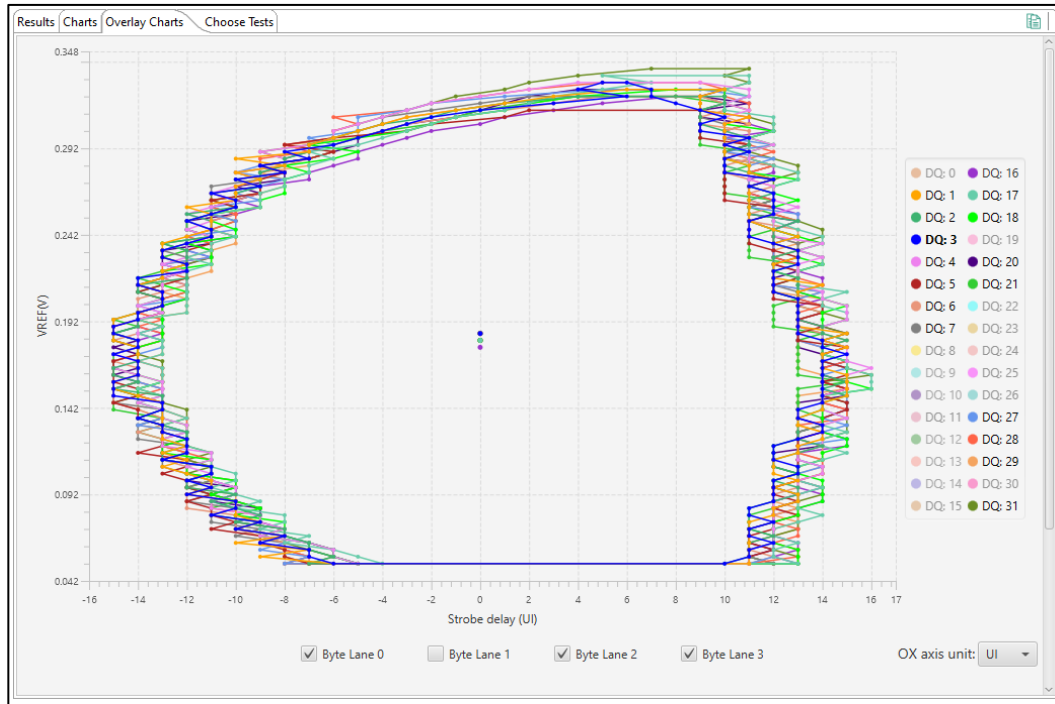


Figure 131 Overlay Charts example displaying a subset of data lines

- Zoom feature is available on the chart and content can be exported in HTML format

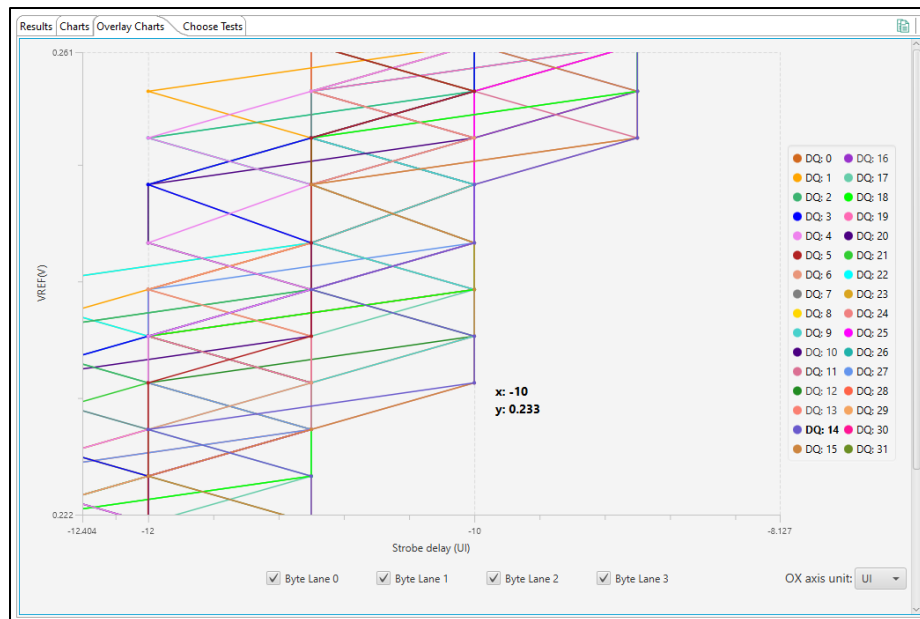


Figure 132 Zoomed-in view in the Overlay Charts

- Added scan for connected S32 Debug probe devices and Test Connection functionality

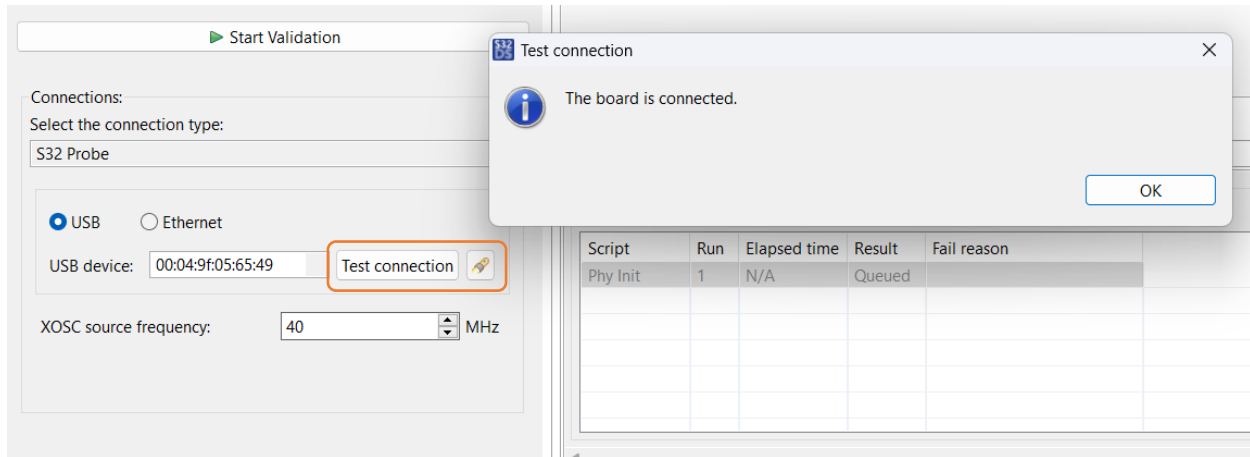


Figure 133 Test Connection feature in ValidationView

New features in S32CT 1.7 U8 R version:

- Enabled Inline ECC support for LPDDR5
- Enabled multiple frequencies for LPDDR5

3.8 eFuse Tool

eFuse tool provides an intuitive graphical interface for writing/programming the OTP (One Time Programmable) bits by software.

Initial support includes a Standard configuration panel where each fuse word can be configured based on the value of its components. Initially all fuse settings have the default values (all bits are 0).

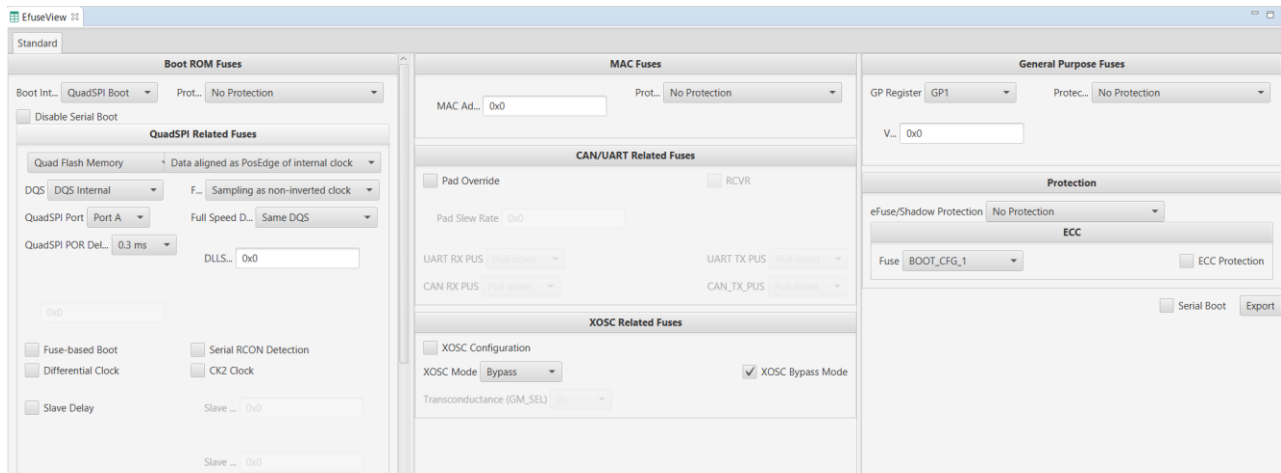


Figure 134 EfuseView - Standard layout

Standard Layout provides the following options:

- Configuration of MAC fuses

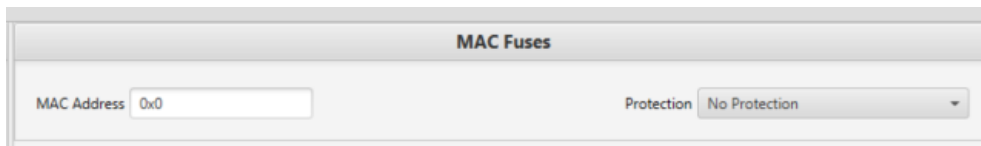


Figure 135 MAC fuses panel

- Configuration of General Purpose fuses

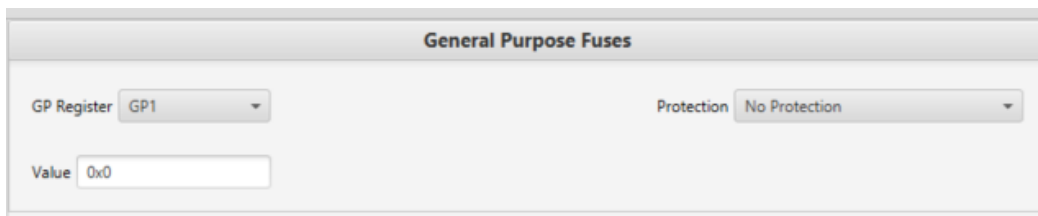


Figure 136 General purpose fuses panel

- Configuration of Boot fuses, with different UI options based on the selected Boot Interface

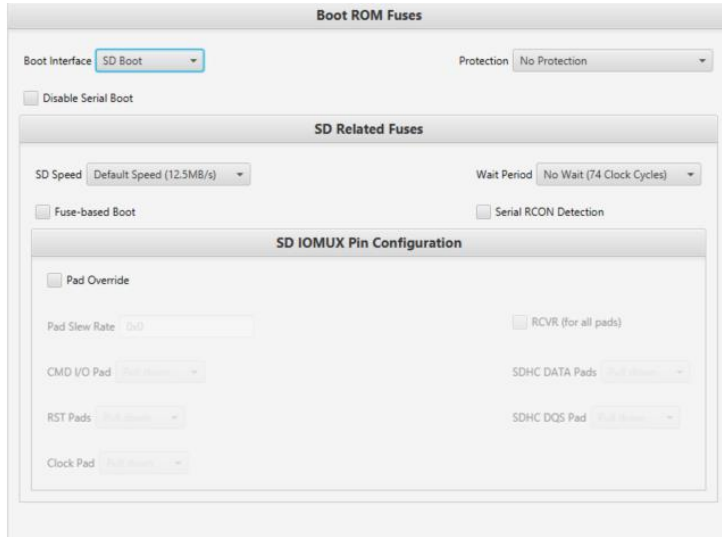


Figure 137 Boot ROM fuses panel for SD Boot

- Control of fuse protection attribute, available for all fuse words

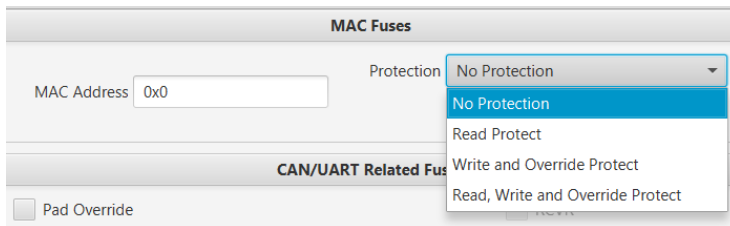


Figure 138 Fuse protection options

- Configuration of ECC protection

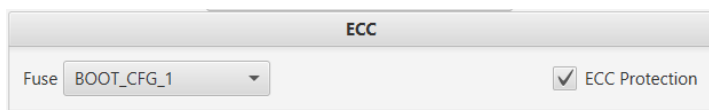


Figure 139 ECC protection panel

- Constraint mechanism aligned with the Reference Manual
- Export functionality

Current fuse configuration can be exported in 2 binary image formats:

- raw binary format
- serial boot format (if Serial Boot checkbox is selected).

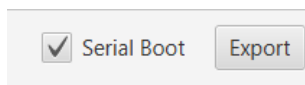


Figure 140 Export panel with Serial boot option

- Added save/restore configuration functionality
- Added import functionality

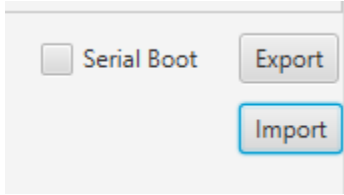


Figure 141 eFuse import button

- Enabled command line support to export the eFUSE binary image

Command name	Definition and parameters	Description
Export eFUSE Configuration	-ExportConfig	Export eFUSE binary configuration. Folder name/path is expected as argument.
Include serial boot header	-serial_boot	Include the serial boot header into the exported eFUSE binary configuration. No arguments are required.

Figure 142 eFUSE Command Line support

- Integrated Problems View and error reporting mechanism, checking the following:
 - Value in a textfield has expected format (hexadecimal, decimal)
 - Textfield value is not empty
 - Textfield value is within the width limit (overflow error is reported)

Level	Resource	Issue	Origin	Target	Type
Error	Value	Input value is invalid! Input is bigger than allowed width.	eFUSE: Field value: 0xffffffff		Tool problem
Error	MAC Address	TextField cannot be empty, please insert a value	eFUSE: Textfield value is empty.		Tool problem

Figure 143 eFuse Problems View example

- Added eFUSE advanced layout, which allows configuration of each fuse word with any custom value which is aligned with the reference manual constraints. It provides the following options:
 - Adding a new fuse word to be configured, “Add Fuse Word” button is located in “Fuse Configuration” area
 - Removal of an existing configured fuse word, “Remove Fuse Word”
 - Changing the value of any available fuse word.
 - Graphical representation of the eFUSE configuration in binary format
 - Same export/import functionalities as standard layout

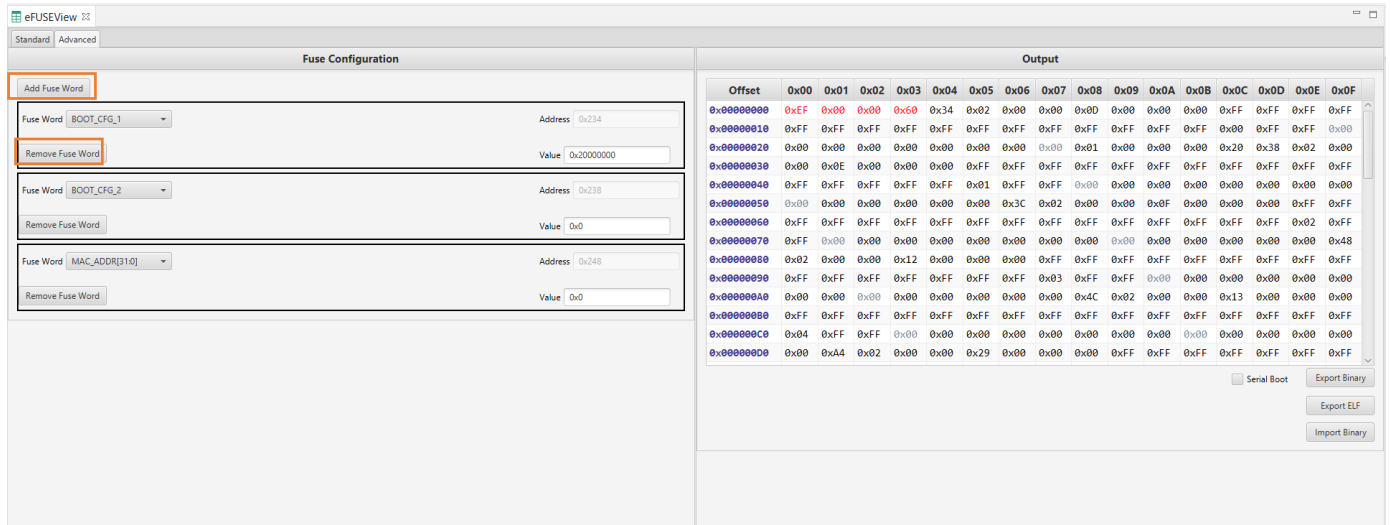


Figure 144 Add/Remove fuse word in eFUSE Advanced layout

- Added HTML report generation – report is exported together with the .bin or .elf files and contains the efuse configuration in a human readable format

eFuse Fuse Words Table

Fuse Word	Value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BOOT_CFG1	0x20000000			DQS : 01	DLSEMPFB : 000			Full Speed Delay : 0	Full Speed Phase : 0	Time Hold Delay : 00				Differential Clock : 0		QuadSPI POR Delay : 000		XOSC Bypass Mode : 0	Column Address Space : 0000			CH2 Clock : 0	QuadSPI Port : 0	Serial RCON Detection : 0	Boot Interface : 000		QuadSPI Mode : 000						
BOOT_CFG2	0x0													Slave Delay Offset : 0000			Slave Delay Coarse : 0000		Slave Delay : 0		Transconductance (GM_SEL) : 0000					Fuse-based Boot : 0	Disable Serial Boot : 0	XOSC Mode : 00		XOSC Configuration : 0			
BOOT_CFG3	0x0							Pad Override : 0		Pad Slew Rate : 000		UART RX PUS : 0	CAN TX PUS : 0	CAN RX PUS : 0	RCVR : 0	UART TX PUS : 0							Pad Override : 0		Pad Slew Rate : 000	CS Pads : 0	Data Pads : 0	DQS Pad : 0	RCVR (for all pads) : 0		Clock Pads : 0		
MAC0_ADDR	0x0																																
MAC0_ADDR	0x0																																
GP1	0x0																																
GP2	0x0																																
GP5[31:0]	0x0																																
GP5[63:32]	0x0																																
GP5[95:64]	0x0																																
GP5[127:96]	0x0																																
GP5[159:128]	0x0																																
GP5[191:160]	0x0																																
GP6[31:0]	0x0																																
GP6[63:32]	0x0																																
GP6[95:64]	0x0																																
GP6[127:96]	0x0																																
GP6[159:128]	0x0																																
LOCK_BITS1	0x0			eFuse/Shadow Protection : 00																		GP5 Fuses Protection : 00		MAC0_ADDR Fuses Protection : 00			BOOT ROM Fuses Protection : 00		GP2 Fuse Protection : 00		GP1 Fuse Protection : 00		
LOCK_BITS2	0x0																															GP3 Fuses Protection : 00	

Generated on Wednesday Oct 2022, 16:34:12 by S32 eFUSE Tool (Processor: S32G274A_Rev2, SDK: amp_sdk1_0, S32 Configurations Tools: 202210060904)

Figure 145 HTML report with the exported eFUSE configuration

- Export eFUSE configuration in ELF format

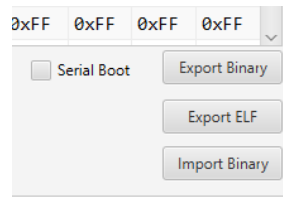


Figure 146 Export Elf format

- Extend command line support

Command name	Definition and parameters	Description
Export eFUSE Configuration in ELF format	-ExportELF	Export eFUSE configuration in elf format. Folder name/path is expected as argument.

- Added Undo/Redo functionality
- Added Reset to processor default functionality
- Enabled Overview tab

3.9 GTM Tool

Allows the configuration of the Generic Timer Module, with the objective of generating source code necessary to output PWM signals or analyze incoming waveforms. The GTM peripheral contains multiple modules, each with its own dedicated function.

GTM Tool is currently offering configuration options for the following components:

- Cluster Clock Management Unit (CCM)
- Clock Management Unit (CMU)
- ARU-connected Timer Output Module (ATOM)
- Timer Output Module (TOM)

The GTM View offers selection via a tree panel with the following main elements:

- Documentation – summary of the working mode of the supported submodules.
- Use-Cases – step by step examples which can be applied in the current configuration.

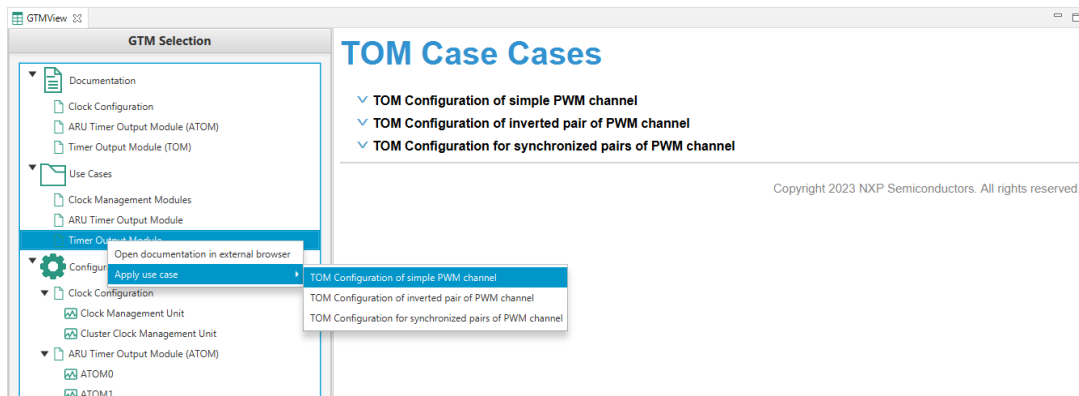


Figure 147 Apply example use-case in current configuration

- Configuration – list with all supported modules and configurable instances

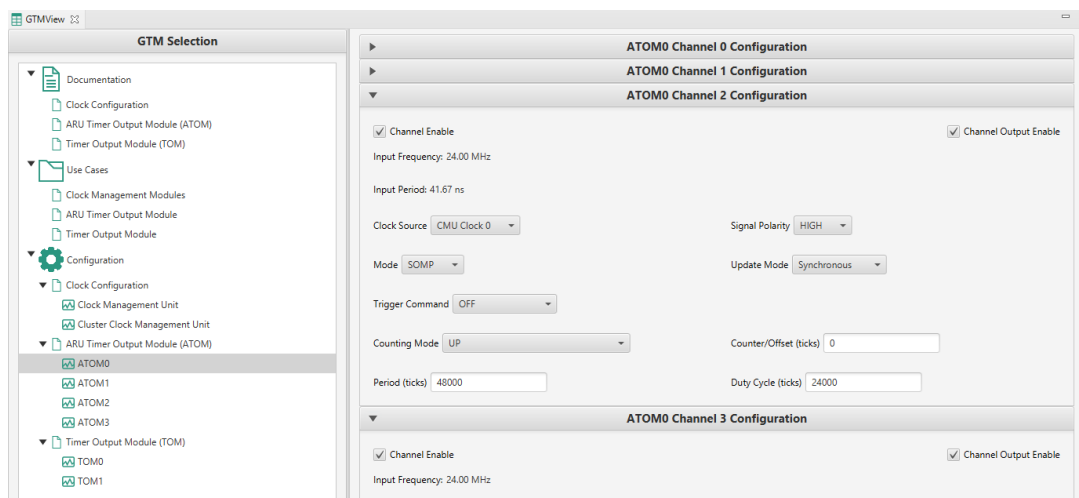


Figure 148 GTM View with ATOM0 module configuration

GTM Tool integrates Problems View and error reporting mechanism, checking for invalid and out of range values.

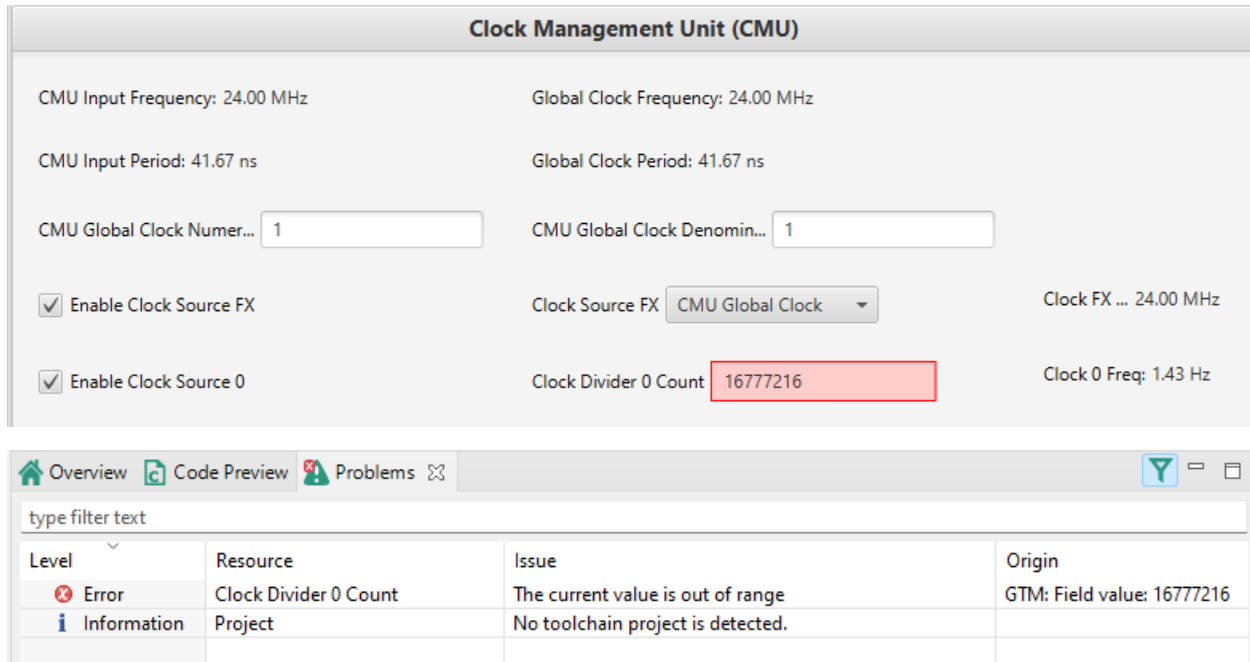
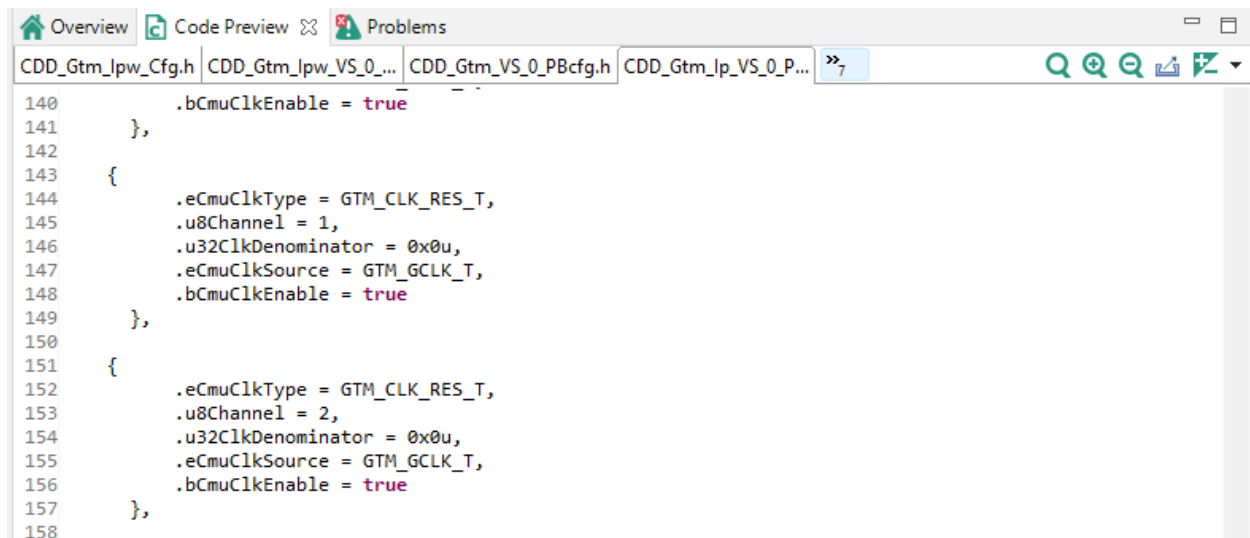


Figure 149 Problems View reporting an out of range value

GTM Tool offers code generation mechanism – each valid change in the configuration triggers a code generation request. By design, the last generated code instance is always valid.



- Support for configuring TIM (Timer Input Module)
- Added Help documentation for all UI options
- Added Undo/Redo support
- Added Import/Export .mex functionality
- Enabled Show Problem feature
- Implementation of output frequency info for ATOM and TOM channels

☒ Channel Enable ☒ Channel Output Enable

Input Frequency: 24.00 MHz Output Frequency: 1.00 kHz Input Period: 41.67 ns Output Period: 1.00 ms

Clock Source: CMU Clock 0 Signal Polarity: HIGH

Mode: SOMP Update Mode: Synchronous

Trigger Command: OFF

Counting Mode: UP Counter/Offset (ticks): 0

Period (ticks): 24000 Duty Cycle (ticks): 12000

Figure 151 Example of Output Frequency Info

New features in S32CT 1.7 U8 R version:

- Interactive diagram view, displaying a graphical overview of the current configuration

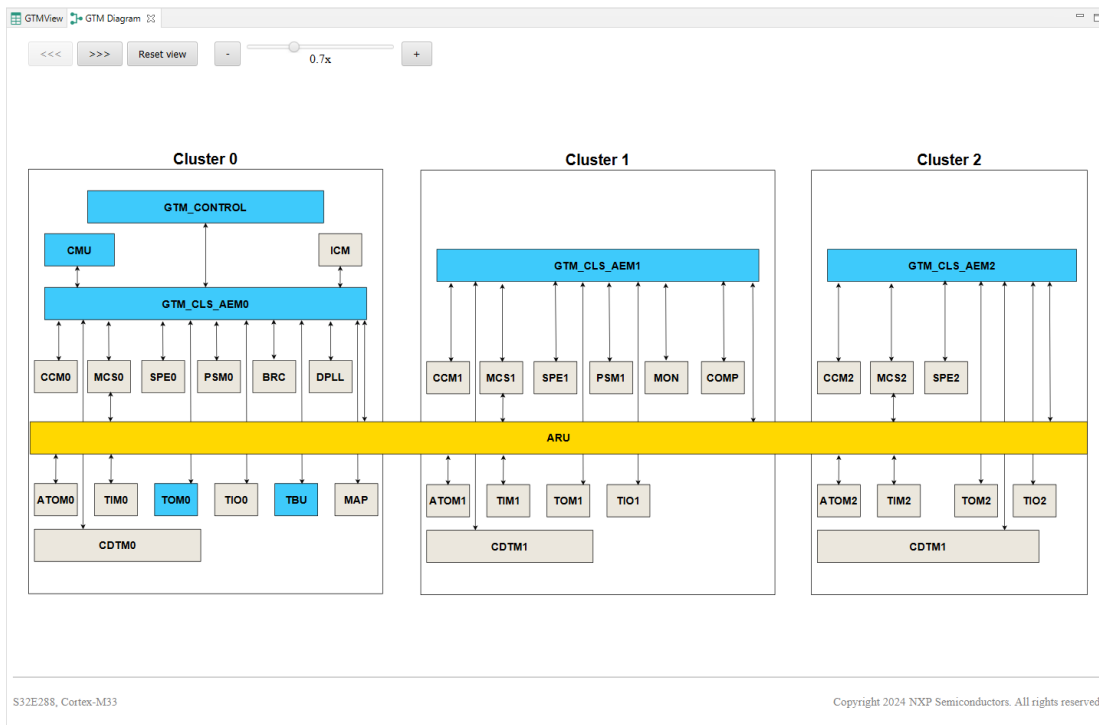


Figure 152 GTM Diagram

- Support for configuring general Clusters clock dividers.

Cluster Clock Management Unit

Input Frequency: 48.00 MHz
Input Period: 20.83 ns

Cluster	Enable	Clock Divider	Frequency
Cluster 0	<input checked="" type="checkbox"/>	Enable	48.00 MHz
Cluster 1	<input checked="" type="checkbox"/>	Enable divide by 2	24.00 MHz
Cluster 2	<input checked="" type="checkbox"/>	Enable divide by 2	24.00 MHz
Cluster 3	<input checked="" type="checkbox"/>	Enable divide by 2	24.00 MHz

Figure 153 Cluster Cock configuration panel

- Support for configuring CCM (Cluster Clock Management)

Cluster Control Management

Cluster No. Cluster 0

Cluster 0

☒ TIM 0 Unit Enable
☒ TOM 0, SPE 0 and TDTM 0 Unit Enable
☒ ATOM 0 and ADTM 0 Unit Enable

Cluster 0 Clock Management

TOM 0 Clocks

CCM 0 FX CLK Clock FX 0

ATOM 0, TIM 0 Clocks

CCM 0 CLK RES 0	CMU Clock 0	CCM 0 CLK RES 4	CMU Clock 4
CCM 0 CLK RES 1	CMU Clock 1	CCM 0 CLK RES 5	CMU Clock 5
CCM 0 CLK RES 2	CMU Clock 2	CCM 0 CLK RES 6	CMU Clock 6
CCM 0 CLK RES 3	CMU Clock 3	CCM 0 CLK RES 7	CMU Clock 7

Figure 154 Example of CCM configuration

- Support for configuring TBU(Time Base Unit)
- Support for configuring TIM (Timer Input Module)
- Added Help documentation for all UI options.
- Updated templates of generated configuration files for CMU, CCM, TBU, TIM, ATOM and TOM.

3.10 S32 Design Studio Integration

3.10.1 Open S32 Configuration Tools

To start using S32 Configuration Tools user must first setup a project in S32 Design Studio. This is done by selecting:

File > New > S32DS Application Project

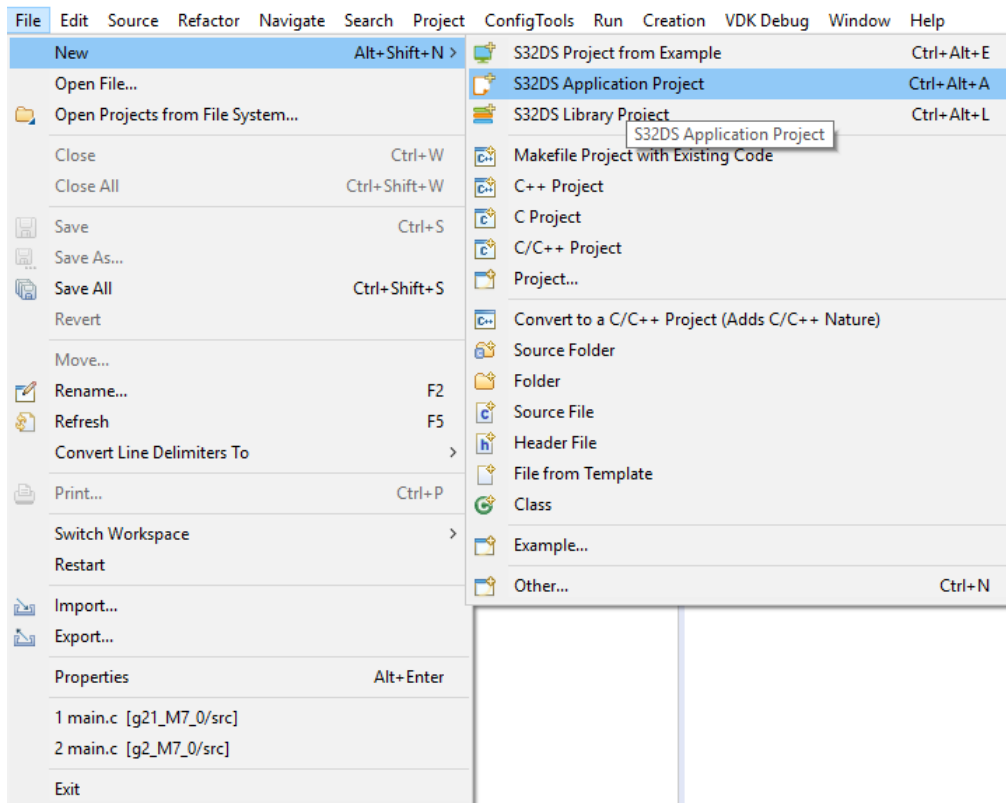


Figure 155: Open S32 Configuration Tools

After that, a new window is prompted to “Create a S32 Design Studio Project”. Follow the wizard by choosing a Project name, Project location and Processor.

Next user must select the required cores, parameters for them, and the preferred SDK/RTD from the available ones. Note that this step is recommended when working with Pins, Clocks, or Peripherals tools as additional enablement is added from SDK/RTD.

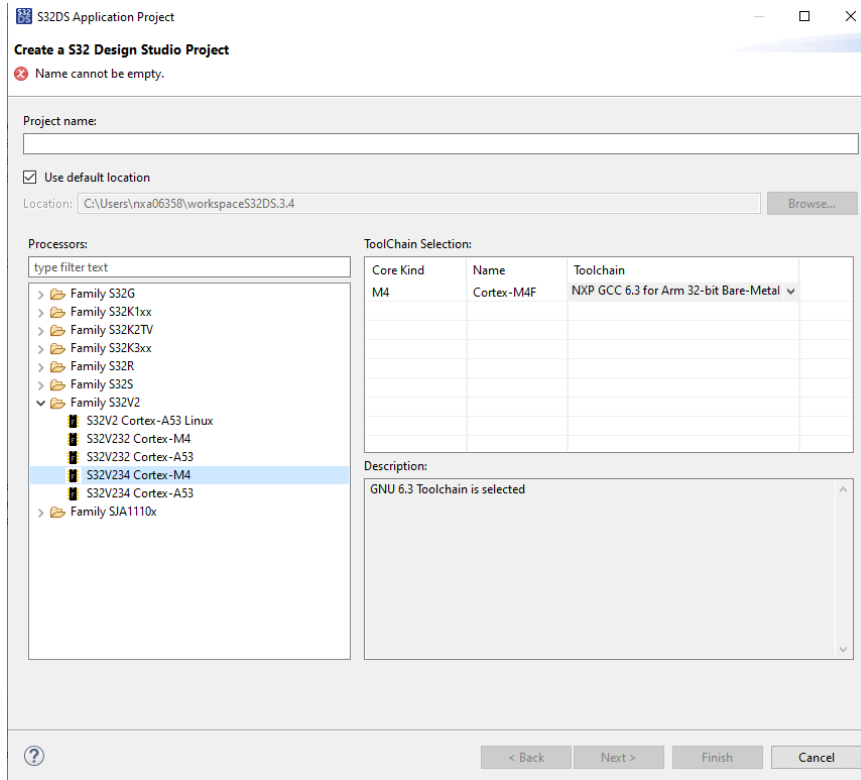


Figure 156: Integration - Create Project

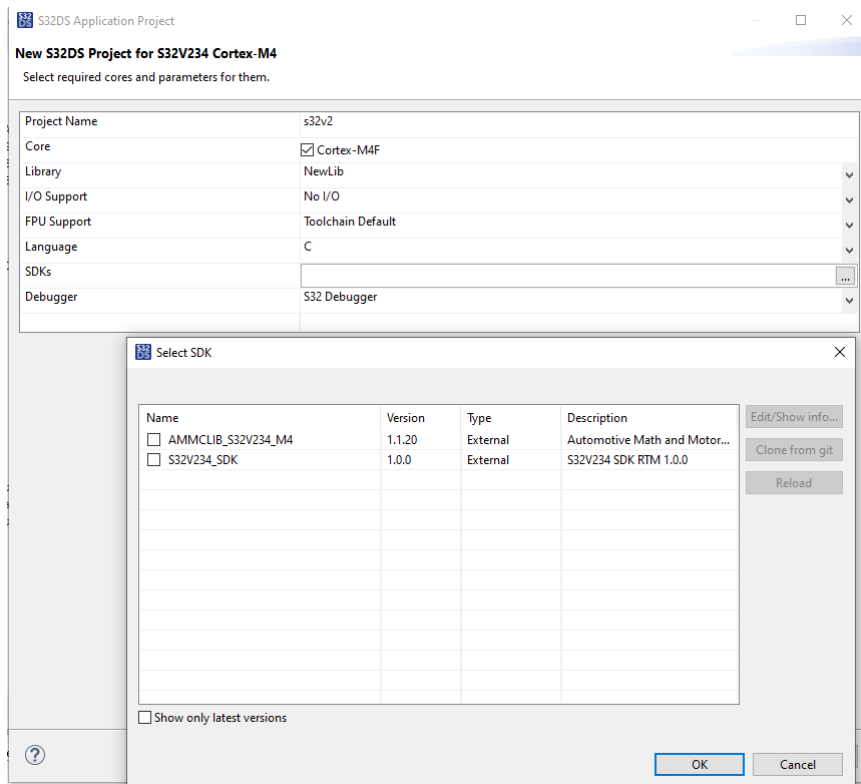


Figure 157: Integration – Select SDK

By clicking on Finish the project is all setup. To open S32 Configuration Tools views right-click on the project and go to S32 Configuration Tools option or select the same option from the upper toolbar.

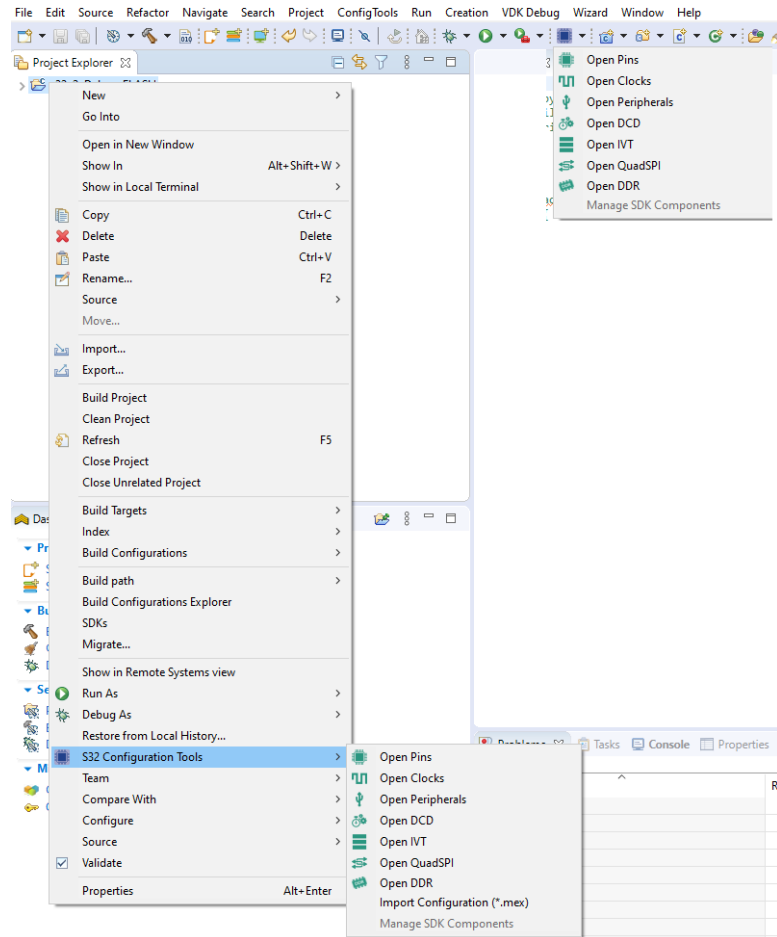


Figure 158: Integration – Open configuration

3.10.2 Add SDK drivers to project

When the configuration is opened in one of the S32 Configuration tools' perspectives, user can just choose "Manage SDK components" button from the toolbar.

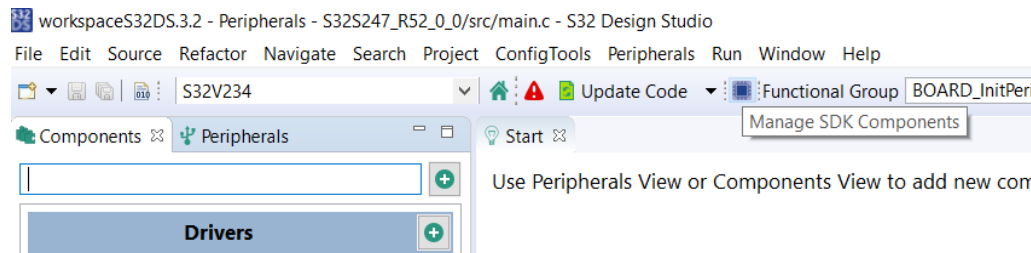


Figure 159: Integration – Open Manage SDK Components

After the option is selected, a dialog with the supported S32 SDK components is displayed:

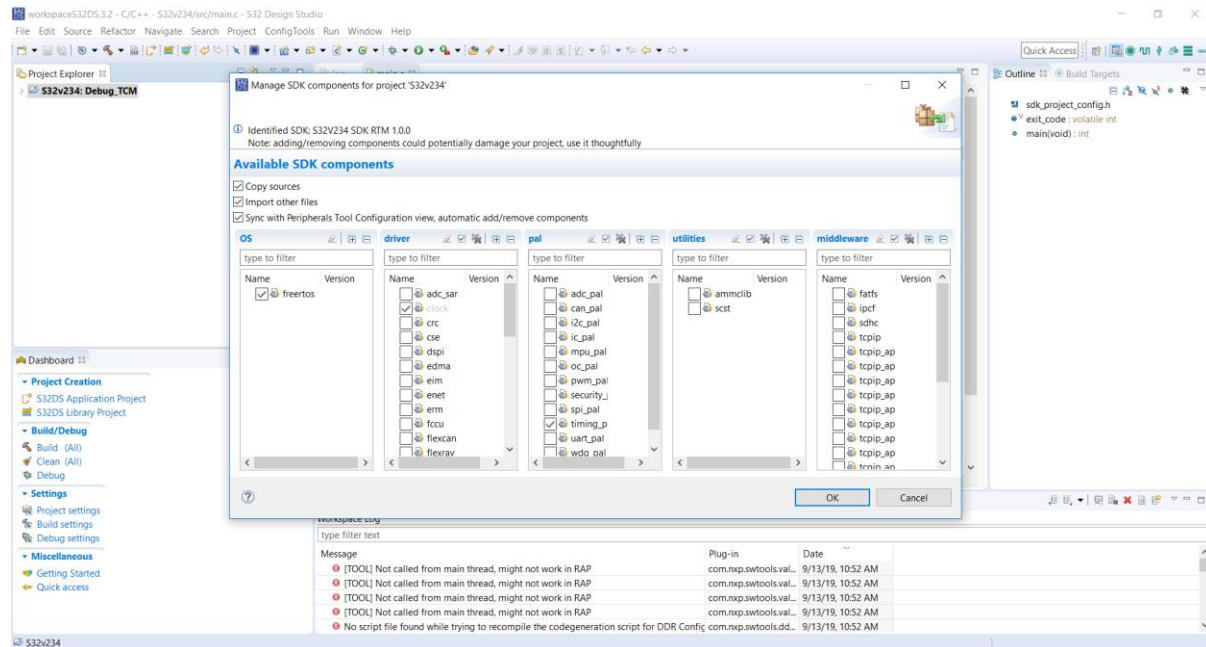


Figure 160: Integration – Manage components

The available options that can be selected are:

- Copy sources - if checked, sources are copied to the workspace project, otherwise they are linked from existing sources in the SDK.
- Import other files - if checked, other files listed in the SDK components and/or example.xml will be imported.
- Sync with Peripherals Tool Configuration view, automatic add/remove components - if true, for each dependency, if available, in Peripherals Tool Configuration the drivers will be added/removed only if the configuration for the selected project is opened.

When “OK” button is pressed, a pop-up with all the drivers’ sources are shown for each operation, add to project or remove from project:

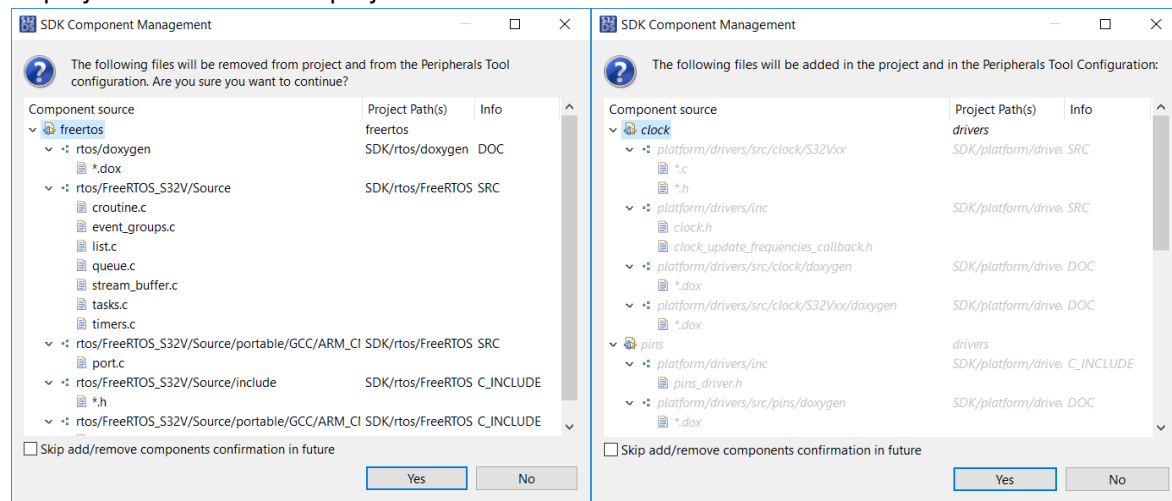


Figure 161: Integration – Add/remove drivers from project

4 Known limitations

IVT Tool:

- For some processors, SYS-IMG pointers reserved status does not properly update when exporting .mex in CLI mode
- Very rarely, the GMAC section is not generated properly using UID
Workaround: re-try the operation in the UI a few times or use directly the volkano utils commands in the console.
- Sporadically, "Problems occurred" shows-up after working in IVT Tool perspective and restarting S32 Design Studio.
Workaround: ignore error as it doesn't impact any functionalities.
- Automatic align is causing overflow when doing automatic align with size of first pointer 0.
Workaround: Insert a valid size for the pointer.
- Tool is not responding when displaying content for AB/DDRC pointer with large code length value.
- There is an error message "The code length value is bigger than the imported image length" when loading a .mex with the raw code from CLI mode.
Workaround: Use the UI mode to generate header for the raw binary and then use the exported binary in CLI process
- For some processors, import blob might not work when importing large blob files (around 1GB)

DCD Tool:

- For registers which have the same address as other registers, the peripheral name in C file and code preview might be incorrect.
Workaround: Can be ignored, no functionality impact.

DDR Tool:

- DDR View interface may be slow when creating content in UI
- Performance degradation when running walking ones/zeros tests
- Validation view is reset to default after closing it

GTM Tool:

- External clock configuration for CMU is currently not supported by the GTM tool.
- Only ATOM SOMP mode is supported in the PWM driver, the rest of the modes will be added in future work.
- Trigger command for ATOM/TOM is currently not supported by the PWM driver.
- Update mode for ATOM/TOM is set by default to synchronous mode, as the PWM driver currently does not support asynchronous mode.
- Counter offset for ATOM/TOM is currently not supported driver-wise.
- TIM configuration supports only Edge counting mode.

S32 Configuration Tools framework:

- Importing new mcu_data with “Data Manager” places the data one level up than expected if no previous data existed before
Workaround: manually move the mcu_data in the correct location
- Command line options are not working outside of installation folder
Workaround: Open .ini file from installation folder and replace the efxclipse.java-modules.dir relative path with the absolute one

S32 Design Studio, integration with RTD/SDK:

- In some cases, there might be warnings with message "Invalid project path: Include path not found ..." after removing an SDK/RTD component
Workaround: remove include path entries manually from Project properties > C/C++ Build > Settings > Includes

How to Reach Us:

Home Page:

nxp.com

Web Support:

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRANCE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale logo, the Freescale logo, Altivec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2023 NXP B.V.