AN14969

Using the Built-In Self-Test (BIST) Functionality on S32K3xx

Rev. 1.0 — 26 February 2026 Application note

Document information

Information	Content
Keywords	S32K3xx, built-in self-test
Abstract	This document introduces BIST, exploring its usage in the S32K3xx devices. It provides details of the programming sequences for supported configurations that are available for each device in the family, all while providing BIST bare-metal register-level configuration details and detailing the software/drivers available. Module-specific BIST features, such as the ADC self-test, are not within the scope of this document.

1 Introduction

As vehicles become increasingly software-defined and reliant on complex electronics, ensuring the reliability and safety of automotive systems has become more important than ever before. In the automotive industry, this can be achieved through compliance with the ISO 26262 functional safety standard, which provides a framework for developing automotive systems that can detect and respond to random hardware faults, thereby reducing the risk of hazardous failures. One key mechanism that supports this goal on the chip hardware level is the Built-In Self-Test (BIST) manager functionality, included as part of the S32K3xx family of microcontrollers.

The BIST manager is designed to operate as part of a broader safety architecture, supporting features such as single point and latent fault detection, safety boot modes, and fault recovery strategies. These capabilities are essential for achieving an Automotive Safety Integrity Level (ASIL) up to ASIL D, the highest level defined by ISO 26262. By detecting permanent faults in the chip logic through the Logic Built-In Self-Test (LBIST) submodule and its memory blocks through the Memory Built-In Self-Test (MBIST) submodule, BIST contributes significantly to the robustness and availability of the aforementioned safety-critical automotive systems.

2 Objective

After reading this application note, all in the context of S32K3xx, you should:

- Understand the basics behind the functionality and intent of BIST features available.
- Understand the difference between MBIST and LBIST.
- Learn which programming sequences for supported BIST configurations are available.
- Be able to invoke the available BIST sequence using the STCU2 controller.
- Be able to develop application strategies for deploying BIST testing.
- Learn about safety software offerings from NXP that can be used for BIST programming.

This application note also provides a BIST bare-metal code example for self-test configurations that you can directly implement.

3 Related documentation

To aid in the understanding of this application note, you can make use of these additional NXP documents and resources:

Table 1. Other NXP documentation

Document	Purpose
S32K3xx Data Sheet	Specifies the values of the S32K3xx device technical characteristics (ratings, operating requirements, operating behaviors, and attributes).
S32K3xx Reference Manual	Describes the structure, function, and programming model of the STCU2 (BIST manager) of S32K3xx.
S32K3xx Safety Manual	Provides assumptions and supporting information on BIST for the design of safety-related systems based on a referenced NXP automotive chip. The addendum workbook contains a set of generic safety mechanisms defined for the S32K3xx device family that is referred within the safety analysis work products, the set of Safety Element out of Context (SEooC) assumptions assigned to an external element, and a module classification worksheet specifying whether each module on the chip is safety related.
S32 Safety Software Framework (S32 SAF) and/or S32 Safety Peripheral Driver (S32 SPD) documentation	Describes functionality, interface, and assumptions for integrating S32 SAF/SPD, which include the BIST driver, into applications that conform to the ISO 26262 standard.

4 Overview of built-in self-test (BIST)

The term Built-In Self-Test (BIST) is used to describe the on-chip hardware-integrated mechanisms designed to support or fully perform the testing of MCU features, used to detect permanent faults. The BIST allows the MCU to conduct self-tests to identify faults. The results of these self-tests can then be used by the MCU to handle the faults and ensure that the device remains in a safe state.

4.1 LBIST and MBIST

There are two different types of BISTs implemented on S32K3xx devices:

- Memory Built-In Self-Test (MBIST) – for memory testing purposes
- Logic Built-In Self-Test (LBIST) – for logical testing purposes

MBIST is implemented for each of the SRAM, cache and TCM memories, as well as for peripheral memories on the MCU such as eDMA, FlexCAN, EMAC, HSE and QSPI. For MBIST testing purposes, each of the memories is segmented into individual MBIST partitions.

LBIST tests operate on the digital logic of the device and use scan test techniques to provide high coverage defect detection. The logic is divided into multiple partitions, with each partition containing user-recognizable logic modules (CPU, XBAR, eDMA, and so on).

It is important to note that not all members of the S32K3xx family include both types of BISTs. These BIST types are device-specific and are controlled by Self-test Control Unit (STCU2). The specific BISTs supported and the elements they check vary from device to device. For detailed information about a specific device, see the Reference Manual.

4.2 Device BIST execution

It is anticipated that a user application would require two standard configurations for BIST testing:

- Vehicle start-up/shutdown
- Vehicle on runtime

The BIST tests can be triggered by software via writing to the STCU2 registers, and are configured to execute after the MCU boots or gets a destructive reset. This procedure is performed after BootROM execution and after the application startup, but before the main application software executes. The hardware automatically executes the tests, and upon successful BIST execution, a functional reset is performed at the end of the sequence and the application software execution can proceed; results are provided in the STCU2 registers that software must check. This configuration can also be used for vehicle shutdown. When running the testing at start-up/shutdown, BIST contributes to increasing the diagnostic coverage of Latent Point Fault Metric (LPFM) within MCU.

In addition to being able to run at start-up/shutdown, you can trigger the BIST execution at any time. A key consideration for this case is the BIST resource usage. Typically, any resource being tested or required by a BIST cannot be used by other functions during the test. This poses less of a challenge for MBIST, as memory elements and their control logic are usually naturally isolated. However, this separation is not present in LBIST, which greatly limits its ability to run concurrently with other operations. To address this, a common approach is to schedule BIST during periods when it does not interfere with other operations (for example, for failure diagnostics and quality control within a manufacturing environment).

Both presented cases are considered on-line (runtime) testing because they run after the BootROM execution and application startup, where the MCU allows software to write to the STCU2 module during runtime to configure and trigger the execution of BIST. S32K3xx does not support off-line execution from the DCD records.

5 Self-Test Control Unit (STCU2)

STCU2 is a programmable hardware module that controls the self-test sequence applied. It is able to manage the device's LBIST and MBIST blocks. To configure BIST testing via software, an IPS interface allows access to the STCU2 registers. Using this interface, software can configure the STCU2 registers for execution of the tests or checking the results.

STCU2 has a programmable watchdog timer that specifies the maximum time allowed for the execution of a BIST sequence. In case the time specified by the watchdog has exceeded, the current BIST execution is interrupted and a failure flag is generated; recoverable and non-critical faults are mapped to FCCU, and unrecoverable and critical faults are mapped to MC_RGM.

There are three phases in STCU2:

- Software configuration - program the test configuration to be executed.
- Start of BIST execution - the configuration is executed.
- End of BIST - the execution results are available and actions can be taken according to them.

It is important to note that, after the self-test sequence is triggered, the chip becomes unavailable for software application during the self-test run. After BIST execution, the results are available and remain after functional reset. If the BIST runs during shutdown, its results must be saved for the next power cycle and verified during startup.

Before running BIST, it is required to stop all the communications by software and disable all the peripheral controllers (PCTLs), as they are unusable in self-test, and it is important to have a safe stating of inputs of non-listed peripheral modules. It is also recommended to disable SWT, JTAG, HSE_SWT, FCCU_RST, and DEBUG_FUN in addition to any module that could generate a functional reset, to avoid self-test abortion. The following additional requirements to run BIST must also be met to guarantee a successful execution:

- Only a single core should be active and used to configure the MCU for the self-test. The main safety core is the recommended core (CM7_0 for most S32K3xx devices).
- Ensure that software-initiated functional/destructive resets are not triggered during self-test.
- All system and peripheral clocks should be configured to generate maximum functional frequency as defined in the S32K3xx Reference Manual and Data Sheet.
- LBIST_CLK should be set to 40 MHz.
- The system clock should be based on PLL.

6 MBIST and LBIST testing partitions

A BIST partition is a part of the chip for which a BIST has been defined. The STCU2 module supports the following types of BIST partitions.

- MBIST partition: SRAM or ROM block
- LBIST partition: One or more digital modules

The S32K3xx family only has up to 1 LBIST partition and up to 18 MBIST partitions. As mentioned in a previous chapter, the specific BISTs supported and the elements they check vary from device to device. For detailed information about a specific device, see the S32K3xx Reference Manual.

7 STCU2 BIST configuration

To perform a self-test successfully, correct configuration of the STCU2 module must be applied prior to its execution. The STCU2 BIST configuration consists of the following steps:

1. Unlock STCU2.

2. Program the STCU2_CFG register.
3. Program the STCU2_WDG register.
4. Program the STCU2_MB_CTRL registers.
5. Program the STCU2_LB_CTRL registers.
6. Start the BIST.

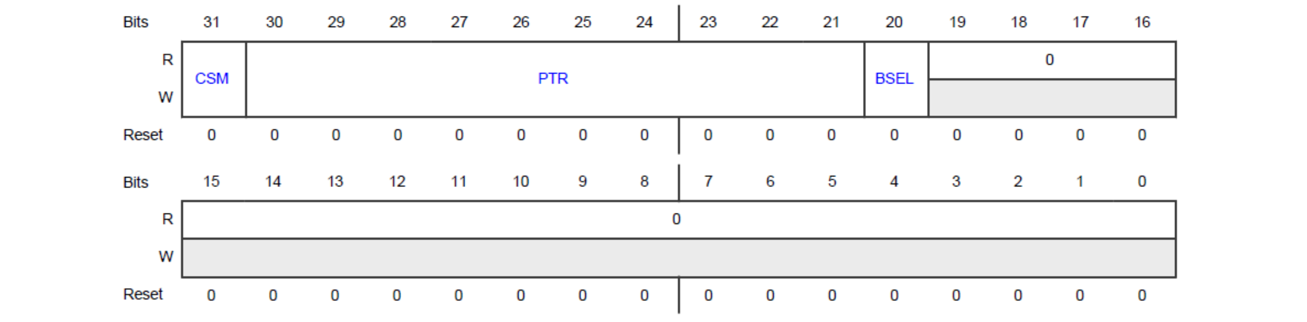
7.1 Unlock STCU2

The STCU2_SKC register implements the security key code mechanism needed to access the write mode for write protected STCU2 registers. In order to unlock the STCU2 access after an STCU2 asynchronous reset, and at the end of the STCU2 run, the software or the SSCM interfaces apply the following sequence:

1. Write Key1 (0x753F924E) into the STCU2_SKC register.
2. Write Key2 (0x8AC06DB1) into the STCU2_SKC register.

7.2 Program the STCU2_MB_CTRL registers

The STCU2_MB_CTRL register defines the control setting of MBIST controller. Program the STCU2_MB_CTRL registers of each NMCUT to be executed.



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

Figure 1. STCU2_MB_CTRL registers

- CSM - Concurrent/sequential mode.
- PTR - PTR defines the logical pointer to the next LBIST or MBIST to be scheduled.
- BSEL - BIST select.

An example of MBIST and STCU2_MB_CTRL register configuration is shown in Section 12.

Note: *The last MBIST control register must be configured to sequential mode. Otherwise, STCU2 reports a pointer configuration error via the error status register bit: invalid linked pointer list.*

7.3 Program the STCU2_WDG register

Program the STCU2_WDG register to define the time budget assigned for the LBIST/MBIST execution.

Figure 2. Programming the STCU2_WDG register

```
STCU2.WDG = 0xFFFFFFFF;  /* Watchdog timeout to Max value */
```

7.4 Program the STCU2_CFG register

Program the STCU2_CFG register to define the core and LBIST/MBIST clock prescaling factor setting the CLK_CFG bits, and set the pointer to the first LBIST/MBIST to be executed.

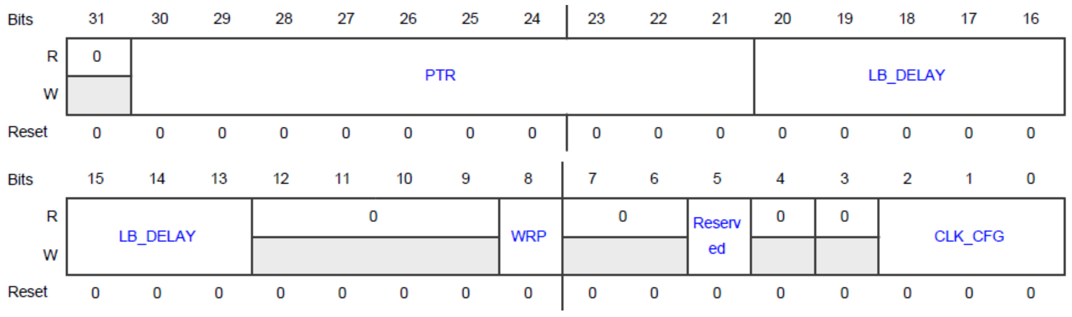


Figure 3. STCU2_CFG register

Figure 4. Programming the STCU2_CFG register

```
STCU2_CFG.CLK_CFG =0;  /* LBIST/MBIST + STCU2 CLK configuration */
STCU2_CFG.LB_DELAY = 0; // Delay LBIST run */
STCU2_CFG.PTR = 0x80;  // Start with MBIST_0 pointer (0x80) */
```

7.5 Program the STCU2_LB_CTRL registers

The STCU2_LB_CTRL register defines the control setting of each LBIST controller.

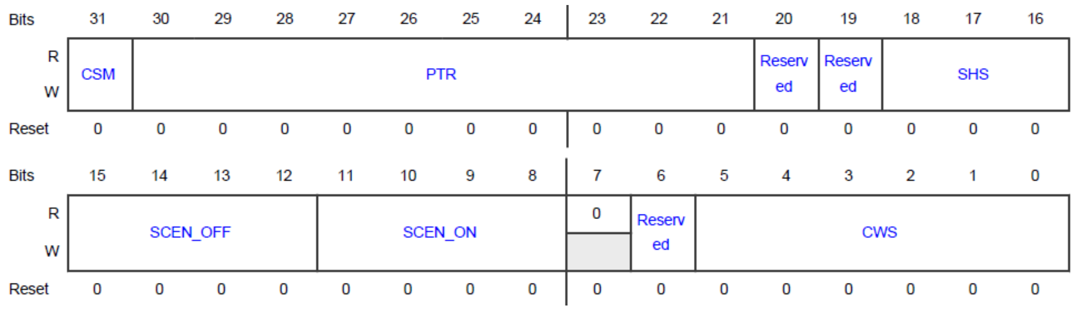


Figure 5. STCU2_LB_CTRL register

An example of LBIST and STCU2_LB_CTRL register configuration is shown in Section 12.

Note: *STCU2_LB_CTRL[SCEN_ON] and STCU2_LB_CTRL[SCEN_OFF] must be programmed to a value >=1.*

7.6 Start BIST

Program the RUNSW bit into the STCU2_RUNSW register to enable the Self-Test execution.

Figure 6. Programming for self-test execution

```
STCU2.RUNSW = 0x1;  /* Start the BIST sequence */
```

8 Supported BIST configurations

S32K3xx's BIST manager allows you to run two NXP validated LBIST and MBIST configuration sets (BIST_SAFETYBOOT_CFG and BIST_DIAGNOSTIC_CFG). Any modification to these configurations changes the coverage levels achieved and is therefore not recommended or guaranteed.

Table 2. BIST configuration sets

Name	Description
BIST_SAFETYBOOT_CFG	Safety Boot configuration is supposed to be triggered from the custom bootloader/ startup code that follows BootROM - HSE firmware execution. It affects the main reset domain (RD0), meaning end of the execution is followed by a functional reset.
BIST_DIAGNOSTIC_CFG	Diagnostic configuration contains every single BIST available on the platform. This configuration is meant to be running only when no other (or just limited number of jobs) are in progress on the chip. The results of the DIAGNOSTIC_CFG self-test can be evaluated after the STCU2 triggered functional reset.

The LBIST and MBIST partition sequences for both supported configurations of the chip are detailed in the following table.

Table 3. LBIST-supported configuration sequences

BIST ID (NLBIST)	BIST Instance Name	Safety Boot	Safety Diagnostic
0	LBIST	✓	✓

Note: *LBIST is not available for S32K312, S32K311, and S32K310.*

Table 4. MBIST-supported configuration sequences for S32K314/S32K324/S32K344

BIST ID (NMCUT)	BIST Instance Name	Safety Boot	Safety Diagnostic
0	SYS0_RAMS	✓	✓
1	SYS1_RAMS	✓	✓
2	DMA_TCD_RAM	✓	✓
3	CM7_0_TOP	✓	✓
4	CM7_1_TOP	✓	✓
5	FLEX_CAN_RAMS	✓	✓
6	QSPI_PERI_RAMS	✓	✓
7	EMAC_TSN_RAM	✓	✓
8	EMAC_RAMS	✓	✓
9	b03_ETF_RAMS	✓	✓
10	HSE_RAMS	✓	✓
11	HSE_ROMS		✓

Table 5. MBIST-supported configuration sequences for S32K342/S32K341/S32K322

BIST ID (NMCUT)	BIST Instance Name	Safety Boot	Safety Diagnostic
0	HSE_ROMS		✓
1	HSE_RAMS	✓	✓

Table 5. MBIST-supported configuration sequences for S32K342/S32K341/S32K322*...continued*

BIST ID (NMCUT)	BIST Instance Name	Safety Boot	Safety Diagnostic
2	SYS0_DMA_RAMS	✓	✓
3	CM7_0_TOP	✓	✓
4	FLEX_CAN_RAMS	✓	✓
5	CM7_1_TOP	✓	✓
6	QSPI_PERI_RAMS	✓	✓
7	EMAC_TSN_RAM	✓	✓
8	EMAC_RAMS	✓	✓

Table 6. MBIST-supported configuration sequences for S32K310/S32K311/S32K312

BIST ID (NMCUT)	BIST Instance Name	Safety Boot	Safety Diagnostic
0	HSE_ROMS		✓
1	HSE_RAMS	✓	✓
2	SYS0_DMA_RAMS	✓	✓
3	CM7_0_TOP	✓	✓
4	FLEX_CAN_RAMS	✓	✓

Table 7. MBIST-supported configuration sequences for S32K358/S32K348/S32K338/S32K328

BIST ID (NMCUT)	BIST Instance Name	Safety Boot	Safety Diagnostic
0	HSE_ROMS		✓
1	HSE_RAMS	✓	✓
2	SYS0_DMA_RAMS	✓	✓
3	SYS1_RAMS	✓	✓
4	SYS2_RAMS	✓	✓
5	CM7_0_TOP	✓	✓
6	CM7_1_TOP	✓	✓
7	CM7_2_TOP	✓	✓
8	FLEX_CAN_RAMS	✓	✓
9	QSPI_PERI_RAMS	✓	✓
10	GMAC_TSN_RAM	✓	✓
11	GMAC_RAMS	✓	✓
12	ETF_RAMS	✓	✓

Table 8. MBIST-supported configuration sequences for S32K388

BIST ID (NMCUT)	BIST Instance Name	Safety Boot	Safety Diagnostic
0	HSE_ROMS		✓

Table 8. MBIST-supported configuration sequences for S32K388*...continued*

BIST ID (NMCUT)	BIST Instance Name	Safety Boot	Safety Diagnostic
1	HSE_RAMS	✓	✓
2	SYS0_RAMS	✓	✓
3	SYS1_RAMS	✓	✓
4	SYS2_DMA_RAMS	✓	✓
5	CM7_0_TOP	✓	✓
6	CM7_1_TOP	✓	✓
7	CM7_2_TOP	✓	✓
8	FLEX_CAN_RAMS	✓	✓
9	QSPI_PERI_RAMS	✓	✓
10	GMAC_0_TSN_RAM	✓	✓
11	GMAC_0_RAM	✓	✓
12	ETF_RAMS	✓	✓
13	HTM_ETF	✓	✓
14	GMAC_1_TSN_RAM	✓	✓
15	GMAC_1_RAM	✓	✓
16	CM7_3_TOP	✓	✓
17	DMA_ACE	✓	✓

Table 9. MBIST-supported configuration sequences for S32K389

BIST ID (NMCUT)	BIST Instance Name	Safety Boot	Safety Diagnostic
0	HSE_ROMS		✓
1	HSE_RAMS	✓	✓
2	SYS0_RAMS	✓	✓
3	SYS1_RAMS	✓	✓
4	SYS2_RAMS	✓	✓
5	SYS3_DMA_RAMS	✓	✓
6	CM7_0_TOP	✓	✓
7	CM7_1_TOP	✓	✓
8	CM7_2_TOP	✓	✓
9	FLEX_CAN_RAMS	✓	✓
10	QSPI_PERI_RAMS	✓	✓
11	GMAC_0_TSN_RAM	✓	✓
12	GMAC_0_RAM	✓	✓
13	ETF_RAMS	✓	✓
14	HTM_ETF	✓	✓
15	GMAC_1_TSN_RAM	✓	✓

Table 9. MBIST-supported configuration sequences for S32K389*...continued*

BIST ID (NMCUT)	BIST Instance Name	Safety Boot	Safety Diagnostic
16	GMAC_1_RAM	✓	✓
17	CM7_3_TOP	✓	✓
18	DMA_ACE	✓	✓

9 BIST best and worst case execution times

BIST execution time varies depending on the device being used to execute the test (running at different clock speeds) in addition to whether the MBIST is running with all SRAMs in parallel or in serial mode. The following table provides a detailed information about NXP's silicon-level validation measurements for both the best case and worst case scenarios per device. This duration does not include the MBIST partition self-test on the HSE ROM (that is, execution is run using BIST_SAFETYBOOT_CFG).

Table 10. BIST execution time summary

Device	Category	Execution Time	Comments
S32K344	LBIST+MBIST	3.4 + 2.89 = 6.29 ms	Best case with all SRAMs running in parallel.
	LBIST+MBIST	3.4 + 13.747 = 17.15 ms	Worst case with all SRAMs running in serial.
S32K312 (No LBIST)	LBIST+MBIST	0 + 2.180 = 2.180 ms	Best case with all SRAMs running in parallel.
	LBIST+MBIST	0 + 5.839 = 5.139 ms	Worst case with all SRAMs running in serial.
S32K311 (No LBIST)	LBIST+MBIST	0 + 2.180 = 2.180 ms	Best case with all SRAMs running in parallel.
	LBIST+MBIST	0 + 4.455 = 4.455 ms	Worst case with all SRAMs running in serial.
S32K342	LBIST+MBIST	3.4 + 1.633 = 5.033 ms	Best case with all SRAMs running in parallel.
	LBIST+MBIST	3.4 + 6.864 = 10.264 ms	Worst case with all SRAMs running in serial.
S32K358	LBIST+MBIST	3.4 + 2.341 = 5.741 ms	Best case with all SRAMs running in parallel.
	LBIST+MBIST	3.4 + 14.109 = 17.509 ms	Worst case with all SRAMs running in serial.
S32K388	LBIST+MBIST	3.4 + 3.509 = 6.909 ms	Best case with all SRAMs running in parallel.
	LBIST+MBIST	3.4 + 17.627 = 21.027 ms	Worst case with all SRAMs running in serial.

10 Handling BIST detected permanent faults

When any BIST failure is detected, the chip needs to go into a degraded mode. Since S32K3xx devices don't have any HW support for a degraded mode, this needs to be covered by the application; S32 SAF premium software from NXP provides support for this feature.

When any MBIST fails, the impacted memory should not be used anymore in the application in the degraded mode. The behavior of modules, when BIST fails, should be described in the safety requirement for the particular application.

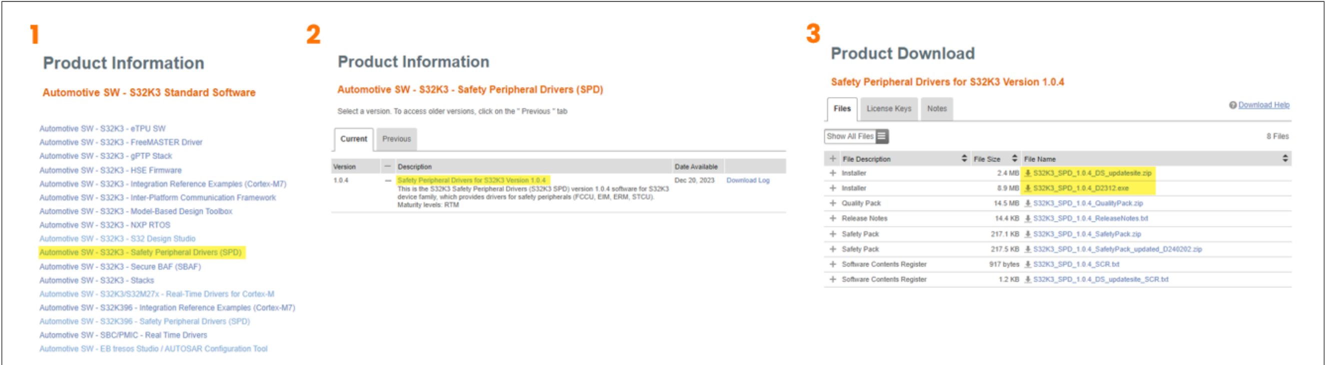
11 Available software and recommendations

NXP offers two main safety software packages for S32K3xx: the S32 Safety Software Framework (SAF) and the S32 Safety Peripheral Driver (SPD). SPD is a standard software package which includes two drivers for safety peripherals, while SAF is a premium software package consisting of the fault detection and reaction components for enabling the safety concept flow of S32K3 Automotive Platform device in addition to including the same two drivers for the safety peripherals included in SPD.

BIST execution and functionality is part of the Safety Peripheral Driver, so its execution and functionality is included as part of the two SW packages. The information included in this section focuses on using BIST as part of the SPD SW package, given this is provided free of charge, but the steps and configurations also apply and can be easily translated to SAF in case you have sourced the premium package.

The BIST manager abstracts the STCU2 and MTR, SELFTEST_GPR, and SELFTEST_GPR_TOP HW modules and provides an API to configure and run, read results, and monitor the status of the BIST tests. It also provides the result analysis of the LBIST/MBIST execution initiated by itself.

The Safety Peripheral Driver (SPD) package is downloadable through NXP's Software Licensing and Support portal, and installed to be configured and used through S32 Design Studio (using the updatesite.zip installer) as well as through any AUTOSAR configuration tool of your choice (using the standalone .exe installer).



Product Information

Automotive SW - S32K3 Standard Software

- Automotive SW - S32K3 - eTPU SW
- Automotive SW - S32K3 - FreeMASTER Driver
- Automotive SW - S32K3 - gPTP Stack
- Automotive SW - S32K3 - HSE Firmware
- Automotive SW - S32K3 - Integration Reference Examples (Cortex-M7)
- Automotive SW - S32K3 - Inter-Platform Communication Framework
- Automotive SW - S32K3 - Model-Based Design Toolbox
- Automotive SW - S32K3 - NXP RTOS
- Automotive SW - S32K3 - S32 Design Studio
- Automotive SW - S32K3 - Safety Peripheral Drivers (SPD)
- Automotive SW - S32K3 - Secure BAF (SBAF)
- Automotive SW - S32K3 - Stacks
- Automotive SW - S32K3/S32M27x - Real-Time Drivers for Cortex-M
- Automotive SW - S32K396 - Integration Reference Examples (Cortex-M7)
- Automotive SW - S32K396 - Safety Peripheral Drivers (SPD)
- Automotive SW - SBC/PMIC - Real Time Drivers
- Automotive SW - EB tresos Studio / AUTOSAR Configuration Tool

Product Information

Automotive SW - S32K3 - Safety Peripheral Drivers (SPD)

Select a version. To access older versions, click on the " Previous " tab

Current | Previous

Version	—	Description	Date Available	
1.0.4	—	Safety Peripheral Drivers for S32K3 Version 1.0.4 This is the S32K3 Safety Peripheral Drivers (S32K3 SPD) version 1.0.4 software for S32K3 device family, which provides drivers for safety peripherals (FCCU, EIM, ERM, STCU). Maturity levels: RTM	Dec 20, 2023	Download Log

Product Download

Safety Peripheral Drivers for S32K3 Version 1.0.4

Files | License Keys | Notes — Download Help

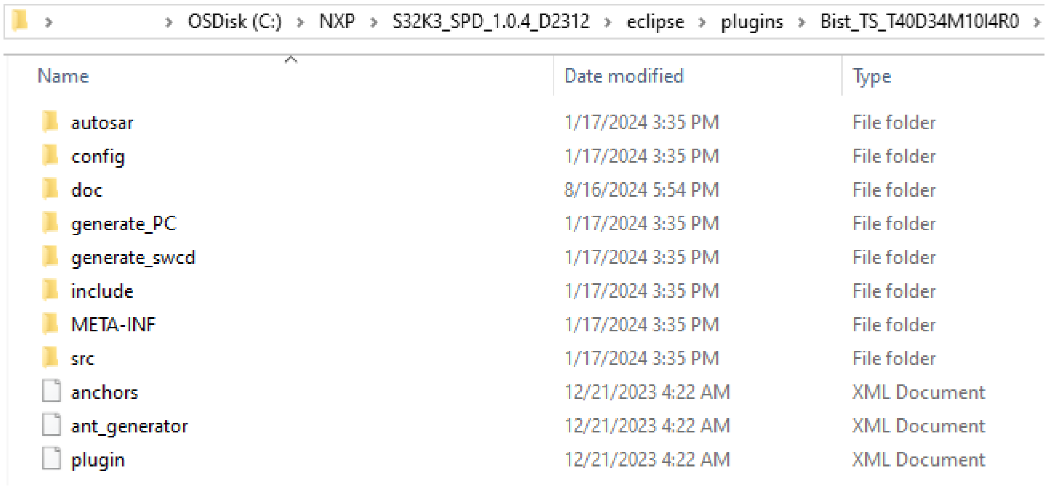
Show All Files — 8 Files

File Description	File Size	File Name
Installer	2.4 MB	S32K3_SPD_1.0.4_DS_updatesite.zip
Installer	8.9 MB	S32K3_SPD_1.0.4_D2312.exe
Quality Pack	14.5 MB	S32K3_SPD_1.0.4_QualityPack.zip
Release Notes	14.4 KB	S32K3_SPD_1.0.4_ReleaseNotes.txt
Safety Pack	217.1 KB	S32K3_SPD_1.0.4_SafetyPack.zip
Safety Pack	217.5 KB	S32K3_SPD_1.0.4_SafetyPack_updated_D240202.zip
Software Contents Register	917 bytes	S32K3_SPD_1.0.4_SCR.txt
Software Contents Register	1.2 KB	S32K3_SPD_1.0.4_DS_updatesite_SCR.txt

Figure 7. Download steps for S32K3xx SPD installers

11.1 BIST for standalone SPD

To make a standalone installation, download the .exe file and follow the wizard steps. After installing it, you will have the next folder in the root directory:



> OSDisk (C:) > NXP > S32K3_SPD_1.0.4_D2312 > eclipse > plugins > Bist_TS_T40D34M10I4R0 >

Name	Date modified	Type
autosar	1/17/2024 3:35 PM	File folder
config	1/17/2024 3:35 PM	File folder
doc	8/16/2024 5:54 PM	File folder
generate_PC	1/17/2024 3:35 PM	File folder
generate_swcd	1/17/2024 3:35 PM	File folder
include	1/17/2024 3:35 PM	File folder
META-INF	1/17/2024 3:35 PM	File folder
src	1/17/2024 3:35 PM	File folder
anchors	12/21/2023 4:22 AM	XML Document
ant_generator	12/21/2023 4:22 AM	XML Document
plugin	12/21/2023 4:22 AM	XML Document

Figure 8. BIST folder included as part of the SPD software package

This folder includes all the necessary files to execute the BIST using S32K3xx. The doc folder contains the user manual with in-depth steps and information to run the BIST functionality through your desired AUTOSAR configuration tool, such as EB Tresos.

11.2 BIST for S32 design studio SPD

You can install SPD through S32 Design Studio by clicking "Install New Software" from the "Help" menu on S32 Design Studio.

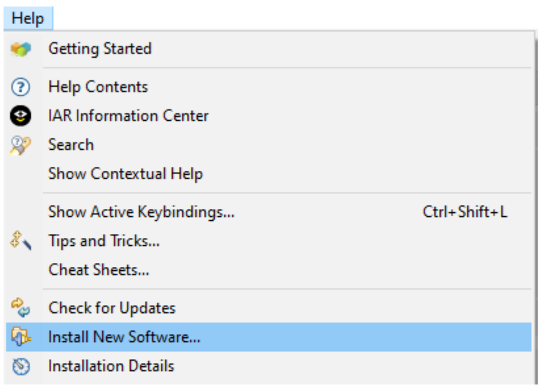
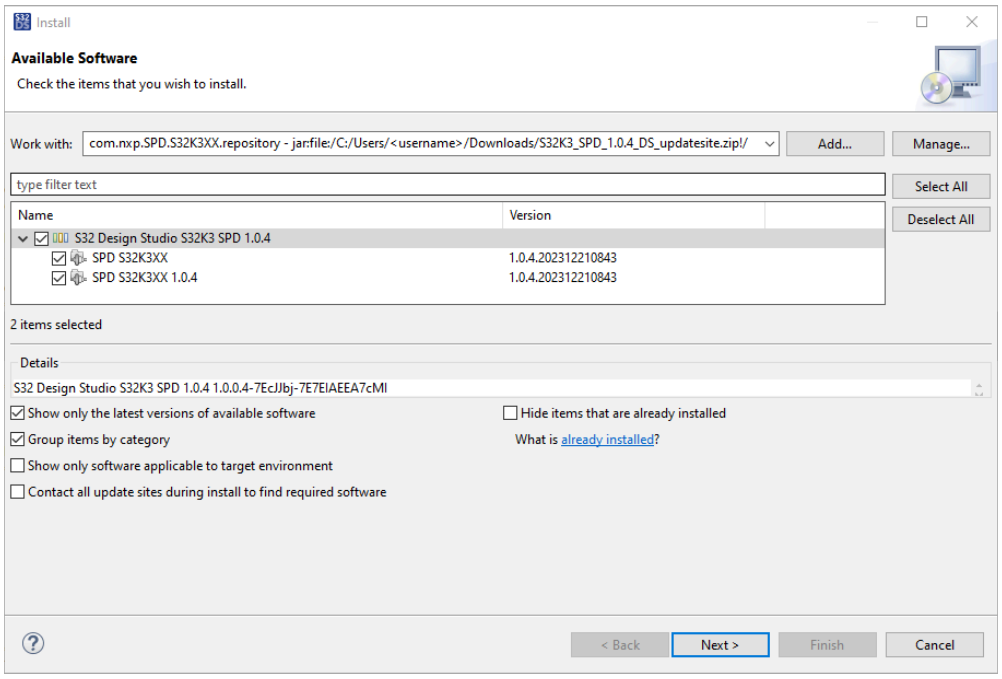


Figure 9. New software installation

Here you need to select the update file for the SPD package:


Install

Available Software

Check the items that you wish to install.

Work with: com.nxp.SPD.S32K3XX.repository - jar:file:/C:/Users/<username>/Downloads/S32K3_SPD_1.0.4_DS_updatesite.zip!/

Name	Version
☑ S32 Design Studio S32K3 SPD 1.0.4	
☑ SPD S32K3XX	1.0.4.202312210843
☑ SPD S32K3XX 1.0.4	1.0.4.202312210843

2 items selected

Details

S32 Design Studio S32K3 SPD 1.0.4 1.0.0.4-7EcJJbj-7E7EIAEEA7cMI

☑ Show only the latest versions of available software

☐ Hide items that are already installed

☑ Group items by category

What is already installed?

☐ Show only software applicable to target environment

☐ Contact all update sites during install to find required software

Figure 10. Updatesite installer for SPD software package

After installation, you will have the BIST driver module available in the "Peripherals Configuration" tool. The following figure shows an example of a configuration that includes the BIST and the eMcem modules from SPD.

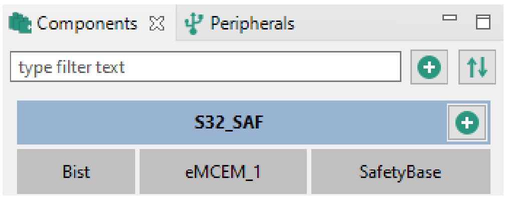
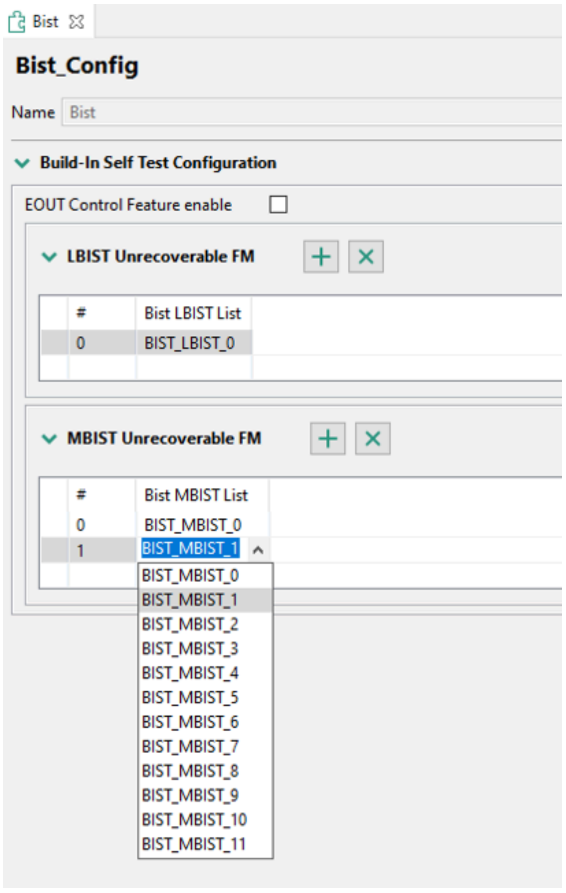


Figure 11. SPD module view in the "Peripheral Configuration" tool

Opening the BIST module reveals the configuration window.



Bist_Config

Name: Bist

Build-In Self Test Configuration

EOUT Control Feature enable ☐

LBIST Unrecoverable FM

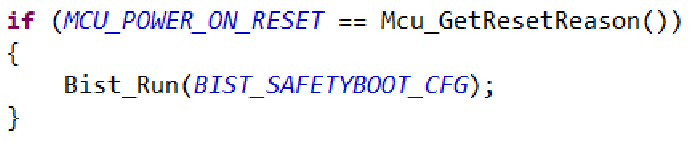
#	Bist LBIST List
0	BIST_LBIST_0

MBIST Unrecoverable FM

#	Bist MBIST List
0	BIST_MBIST_0
1	BIST_MBIST_1

Figure 12. S32 Design Studio BIST manager configuration window

To generate the BIST configuration and the required .c and .h files for later compilation, you need to select "Update Code'" All the BIST functions defined in the User Manual can be called in the main code; *Bist_Run()* is the function that initializes and runs the BIST test.



```
if (MCU_POWER_ON_RESET == Mcu_GetResetReason())
{
    Bist_Run(BIST_SAFETYBOOT_CFG);
}
```

Figure 13. Example of *Bist _Run()* function call

As mentioned in the previous sections, BIST testing can be configured to be performed in one of two different configurations/sequences: BIST_SAFETYBOOT_CFG and BIST_DIAGNOSTIC_CFG. For further SPD details about available API calls, BIST configuration menus, and how to handle unsuccessful attempts to run BIST and read the execution results, please see the S32K3xx SPD BIST User Manual.

11.3 SPD demo application

The S32 SPD installation includes a demo application with BIST configuration and example code, both for the standalone as well as for the S32 Design Studio versions. For further details on the demo installation, configuration and description, see the S32K3xx SPD Demo Application User Manual.

12 BIST bare-metal example

The following code shows a bare-metal example with the minimum requirements to run LBIST and MBIST.

```
#define BIST_STCU_BASE_ADD   ((uint32)0x403A0000UL)
#define BIST_ST_GPR_BASE_ADD   ((uint32)0x403B0000UL)
#define BIST_STCU_LBUFM0_REG   (BIST_STCU_BASE_ADD + 0x7CUL)
#define BIST_STCU_MBUFM0_REG   (BIST_STCU_BASE_ADD + 0x18CUL)
#define BIST_STCU_LB_CTRL0_REG   (BIST_STCU_BASE_ADD + 0x200UL)
#define BIST_STCU_LB_PCS0_REG   (BIST_STCU_BASE_ADD + 0x204UL)
#define BIST_STCU_LB_MISRELSW0_REG  (BIST_STCU_BASE_ADD + 0x220UL)
#define BIST_STCU_LB_MISREHSW0_REG  (BIST_STCU_BASE_ADD + 0x224UL)
#define BIST_STCU_MB_CTRL0_REG   (BIST_STCU_BASE_ADD + 0x2214UL)
/*** @brief    This function configures the LBIST controller. ***/
void LBIST_Config()
{/* Configure LBIST Control registers */
*(volatile uint32*)BIST_STCU_LB_CTRL0_REG/* LBIST CTRL0 Run sequentially,
 pointer to NIL. No next BIST execution. */
*(volatile uint32*)BIST_STCU_LB_PCS0_REG = 0x000005DCUL;/* LBIST 0 pattern count
 */
*(volatile uint32*)BIST_STCU_LB_MISRELSW0_REG = 0x73D053B5UL;/* LBIST MISREL
 Expected Low */
*(volatile uint32*)BIST_STCU_LB_MISREHSW0_REG = 0x253B0C27UL;/* LBIST MISREH
 Expected High */
*(volatile uint32*)BIST_ST_GPR_BASE_ADD = 0xAC55UL; /* Configure the LBIST shift
 cycles.*/
*(volatile uint32*)BIST_STCU_LBUFM0_REG = 0x00000001UL;/*Set LBIST Unrecoverable
 FM*/
}
/*** @brief    This function configures the MBIST controller. **/
void MBIST_Config(Bist_TestType BistTest)
{
uint8_t i;
/* Configure MBIST Control registers */
if(BistTest == BIST_SAFETYBOOT_CFG)
{
for(i=0;i<10;i++)
{
*(volatile uint32*)(BIST_STCU_MB_CTRL0_REG + ii + 1) << 0x15UL); /* MBIST CTRL i
 Run concurrently, next in sequence is MBIST i+1 */
*(volatile uint32*)BIST_STCU_MBUFM0_REG = 0x000000ffUL; /*Set MBIST
 Unrecoverable FM*/
}
*(volatile uint32*)(BIST_STCU_MB_CTRL0_REG + 10*4) = 0x00100000UL; /* MBIST
 CTRL10 Run sequentially, next in sequence is LBIST 0 */
*(volatile uint32*)BIST_STCU_MBUFM0_REG = 0x000000ffUL; /*Set MBIST
 Unrecoverable FM*/
}
if(BistTest == BIST_DIAGNOSTIC_CFG)
{
for(i=0;i<11;i++)
{
*(volatile uint32*)(BIST_STCU_MB_CTRL0_REG + ii + 1) << 0x15UL); /* MBIST CTRL i
 Run concurrently, next in sequence is MBIST i+1 */
*(volatile uint32*)BIST_STCU_MBUFM0_REG = 0x000000ffUL; /*Set MBIST
 Unrecoverable FM*/
}
```

```
*(volatile uint32*)(BIST_STCU_MB_CTRL0_REG + 11*4) = 0x00100000UL; /* MBIST
 CTRL11 Run sequentially, next in sequence is LBIST 0 */
*(volatile uint32*)BIST_STCU_MBUFM0_REG = 0x000000ffUL; /*Set MBIST
 Unrecoverable FM*/
}
}
```

Table 11. S32K344 BIST bare-metal code example configuration sets and their execution times

Name	Example Execution Time
BIST_SAFETYBOOT_CFG	5.8 ms
BIST_DIAGNOSTIC_CFG	22 ms

The following figures show the execution times of BIST_SAFETYBOOT_CFG and BIST_DIAGNOSTIC_CFG configurations using the provided example code, run on the S32K344 Customer Evaluation Board.

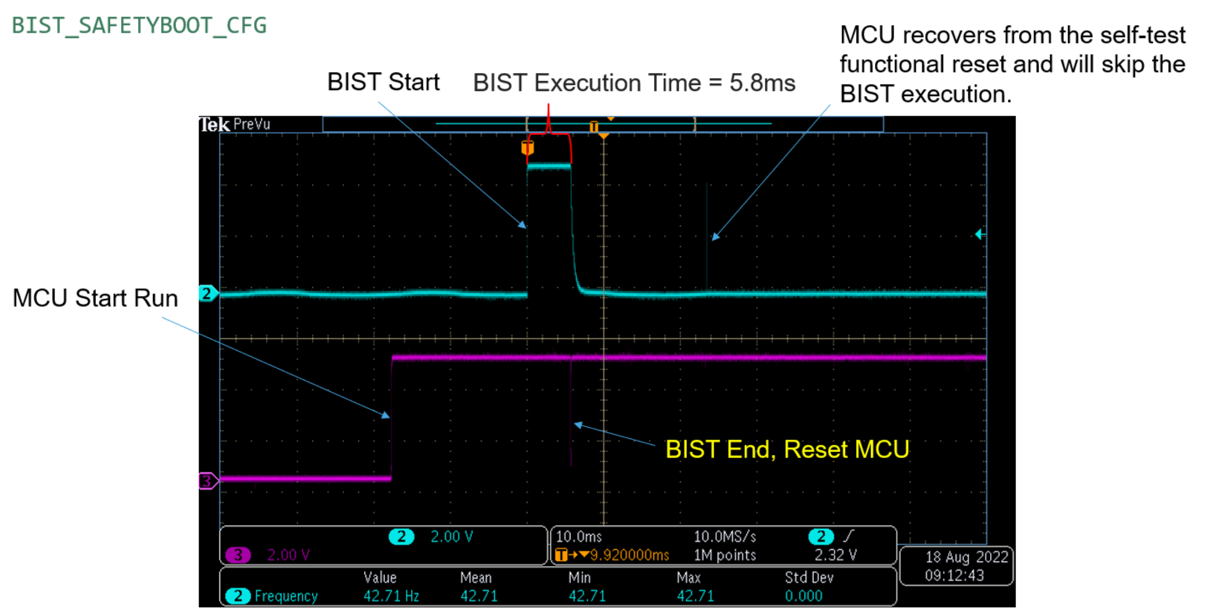


Figure 14. S32K344 BIST bare-metal code example execution time for BIST_SAFETYBOOT_CFG sequence

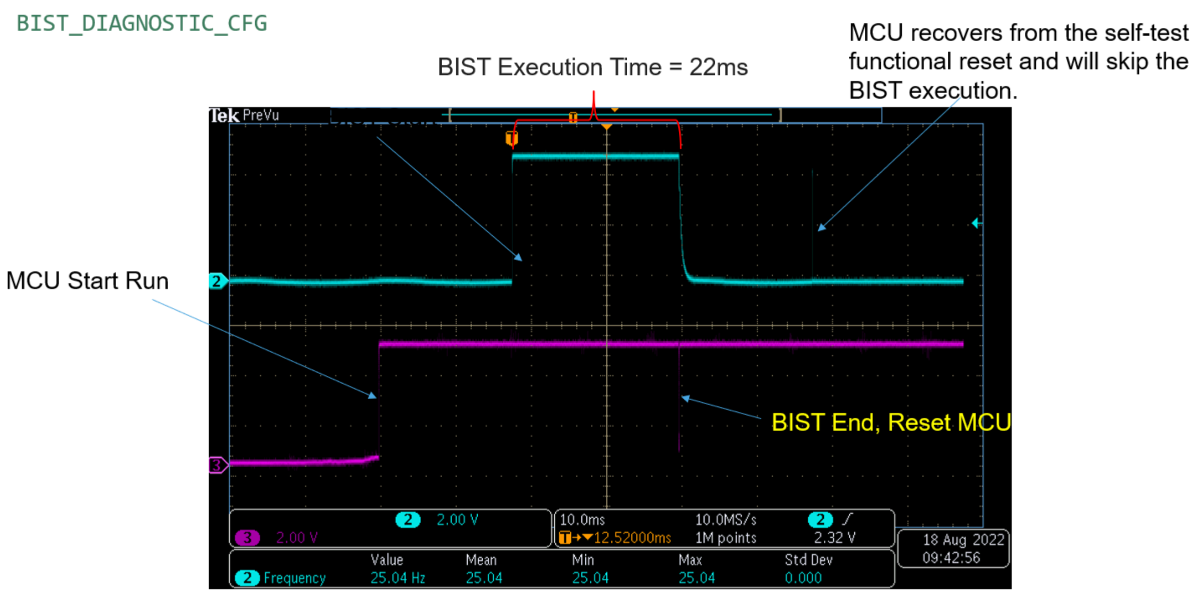


Figure 15. S32K344 BIST bare-metal code example execution time for BIST_DIAGNOSTIC_CFG sequence

13 List of references

Table 12. List of references

#	Item	Version
1	S32K3xx Data Sheet	Rev.11, 04/2025
2	S32K3xx Reference Manual	Rev.10, 04/2025
3	S32K3xx Safety Manual	Rev.7, 03/2025
4	S32 SPD BIST User Manual	UM40BISTSPDR1.0.4, Rev.11.0, 12/2023
5	S32 SPD BIST Demo Application User Manual	UM40IASPDR1.0.4, Rev.9.0, 12/2023
6	S32K344 Customer Evaluation Board	S32K3X4EVB-Q172

14 Note about the source code in the document

The example code shown in this document has the following copyright and BSD-3-Clause license:

15 Abbreviations

Table 13. Abbreviations

Abbreviation	Description
ASIL	Automotive Safety Integrity Level
AUTOSAR	Automotive Open System Architecture
BIST	Built-In Self-Test
BSEL	BIST Select
CAN	Controller Area Network
CLK	Clock
CM7	Cortex-M7
CPU	Central Processing Unit
CSM	Concurrent/Sequential Mode
DMA	Direct Memory Access
EMAC	Ethernet Media Access Controller
FCCU	Fault Collection and Control Unit
HSE	Hardware Security Engine
IPS	Internal Peripheral System
ISO	International Organization for Standardization
JTAG	Join Test Action Group
LBIST	Logic Built-In Self-Test
LPFM	Latent Point Fault Metric
MBIST	Memory Built-In Self-Test
MC_RGM	Reset Generation Module
MCU	Microcontroller Unit
PCTL	Peripheral Controller
PDAC	Peripheral Domain Access Controller
PLL	Phase-Locked Loop
PTR	Pointer
QSPI	Quad Serial Peripheral Interface
ROM	Read Only Memory
SAF	Safety Software Framework
SPD	Safety Peripheral Driver

Table 13. Abbreviations*...continued*

Abbreviation	Description
SRAM	Static Random Access Memory
STCU2	Self-Test Control Unit 2
SWT	Software Watchdog Timer
TCM	Tightly Coupled Memory
XBAR	Crossbar Bus Interface

16 Revision history

Table 14. Revision history

Document ID	Release date	Description
AN14969	26 February 2026	Initial release

Legal information

Tables

Figures

Contents

Date of release: 26 February 2026
Document identifier: AN14969