SafeTI™ Diagnostic Library Software Safety Manual for the Hercules™ Processors

User's Guide

Literature Number: SPNU592
January 2015

Contents

User's Guide
SPNU592–January 2015

SafeTI™ Diagnostic Library Software Safety Manual for the Hercules™ Processors

1 Introduction

This document is a safety manual for the SafeTI™ Diagnostic Library for the Texas Instruments Hercules™ safety microcontroller product family to use the safety diagnostics features of this device and provide a configuration driver for functional safety use of the TPS65381 PMIC. This safety manual provides information needed by system developers to assist in the creation of a safety-critical system using a supported Hercules™ microcontroller, the TPS65381 power management IC and the SafeTI™ Diagnostic Library. This document contains:

- An overview of the superset product safety architecture for management of random failures
- An overview of the development process utilized to reduce systematic failures
- Software Quality Metrics
- SafeTI™ Diagnostic Library overview
- Failure modes and effects analysis report for the SafeTI™ Diagnostic Library

The user of this document should have a general familiarity with the Hercules™ product families. More information can be found at http://www.ti.com/hercules. This document is intended to be used in conjunction with the pertinent data sheets, technical reference manuals, and other documentation for the products under development. This partition of technical content is intended to simplify development, reduce duplication of content, and avoid confusion. The Hercules™ MCU product family utilizes a common safety architecture that is implemented in multiple application focused products. Product implementations covered by this safety manual include:

- RM4xx Safety Critical Microcontrollers
  - RM42x
  - RM46x
  - RM48x
- TMS570LSxx Safety Critical Microcontrollers
  - TMS570LS04x/03x
  - TMS570LS12x/11x
  - TMS570LS31x/21x

You, as a system and equipment manufacturer or designer, are responsible to ensure that your systems (and any TI hardware or software components incorporated in your systems) meet all applicable safety, regulatory, and system-level performance requirements. All application and safety related information in this document (including application descriptions, suggested safety measures, suggested TI products, and other materials) is provided for reference only. You understand and agree that your use of TI components in safety critical applications is entirely at your risk, and that you (as buyer) agree to defend, indemnify, and hold harmless TI from any and all damages, claims, suits, or expense resulting from such use.

2 TI Hercules™ MCU Safety Overview

The Hercules™ MCU family of processors share a common safety architecture concept called a *Safe Island* philosophy. The basic concept involves a balance between application of hardware diagnostics and software diagnostics to manage functional safety, while balancing cost concerns. In the safe-island approach, a core set of elements are allocated continuously operating hardware safety mechanisms. This core set of elements, including power and clock, reset, CPU, flash memory, SRAM and associated interconnect, is needed to assure any functionally correct execution of software. Once correct operation of these elements is confirmed, software execution can begin on these elements in order to provide software-based diagnostics on other device elements, such as peripherals.

The Hercules™ architecture also provides various safety mechanisms and technical recommendations for the use of safety mechanisms. The SafeTI™ Diagnostic Library provides interfaces to these safety mechanisms. Based on the final system requirements the system integrator can use these APIs to incorporate appropriate mechanisms in the final system to meet safety requirements.

Figure 1 illustrates the safe-island approach overlaid to superset configuration of the Hercules™ product architecture.

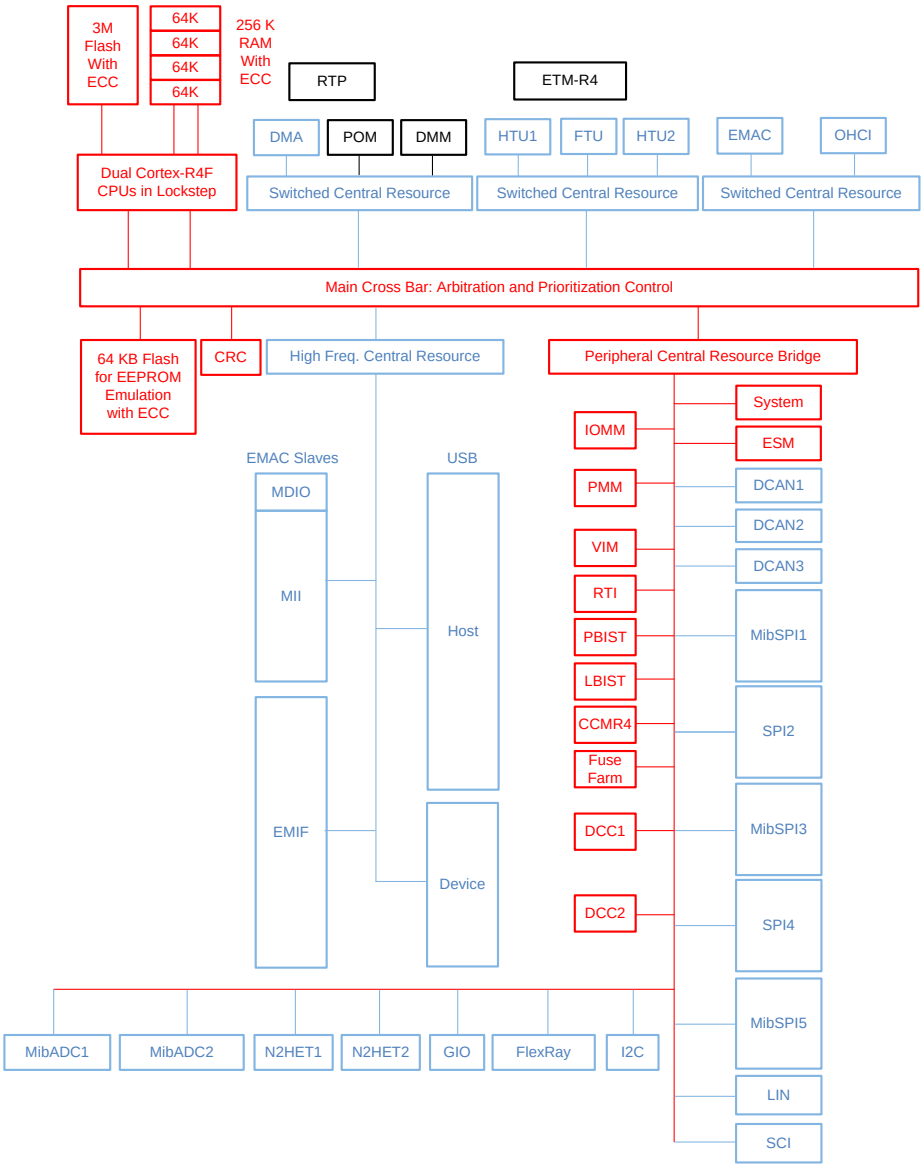

Figure 1. Safe-Island Approach – Hercules™ MCUs

SafeTI, Hercules are trademarks of Texas Instruments.
Cortex is a trademark of ARM Ltd.

Safe-Island Layer (RED): This is the region of logic that is needed for all processing operations. This logic is protected heavily by on board hardware diagnostics and specific assumptions of use to assure a high level of confidence in safe operation. Once this region is safe, it can be used to provide comprehensive software diagnostics on other design elements.

Blended Layer (BLUE): This is the region of logic that includes most safety critical peripherals. This region has less reliance on hardware diagnostics. Software diagnostics and application protocols are overlaid to provide the remainder of needed diagnostic coverage.

Offline Layer (BLACK): This region of logic has minimal or no integrated hardware diagnostics. Many features in this layer are used only for debug, test, and calibration functions; flash is not active during safety critical operation. Logic in this region could be utilized for safety critical operation, assuming appropriate software diagnostics or system-level measures are added by the system integrator.

2.1 TPS65381 Power Management IC Safety Overview

The TPS65381 or TPS65383 device is a multi-rail power supply designed to supply microcontrollers in safety-critical applications, such as those found in automotive. The device supports Texas Instruments' TMS570LS series 16- or 32-Bit RISC flash MCU and other microcontrollers with dual-core lockstep (LS) or loosely-coupled (LC) architectures.

The TPS6538x device monitors undervoltage and overvoltage on all regulator outputs, battery voltage, and internal supply rails. A second band-gap reference, independent from the main band-gap reference, monitors for undervoltage and overvoltage, to avoid any drifts in the main band-gap reference being undetected. In addition, the device implements regulator current limits and temperature protections.

The TPS6538x functional safety architecture features a question-answer watchdog, MCU error-signal monitor, check-mode for MCU error-signal monitor, clock monitoring on internal oscillators, self-check on clock monitor, CRC on non-volatile memory, and a reset circuit for the MCU. A built-in self-test (BIST) allows for monitoring the device functionality at start-up.

2.1.1 TPS Driver Usage in End Application

The TPS library provides driver-level API to interface the Hercules™ device with TPS and make use of the various TPS device features such as voltage monitoring, watchdog monitoring, error monitoring, and so on. The TPS library serves as a special driver library, which helps the application to interface the TPS56381 PMIC with the Hercules™ microcontroller. The library will be released as an additional package along with the SafeTI™ Diagnostic Library package, which will be released as CSP and will help the end customers in the ISO26262 certification of the product. Figure 2 shows the usage of the TPS driver (library) in the end application.

- Driver layer provides necessary APIs to initialize and use the TPS device.
- The send and receive API (MibSPI) function pointers have to be initialized by the application. These APIs will be provided by an interface wrapper which internally uses the MibSPI or SPI APIs. The interface wrapper provides the necessary APIs as needed by the TPS driver.
- These send and receive APIs will be used to send commands to the TPS device and receive contents of the TPS device registers.

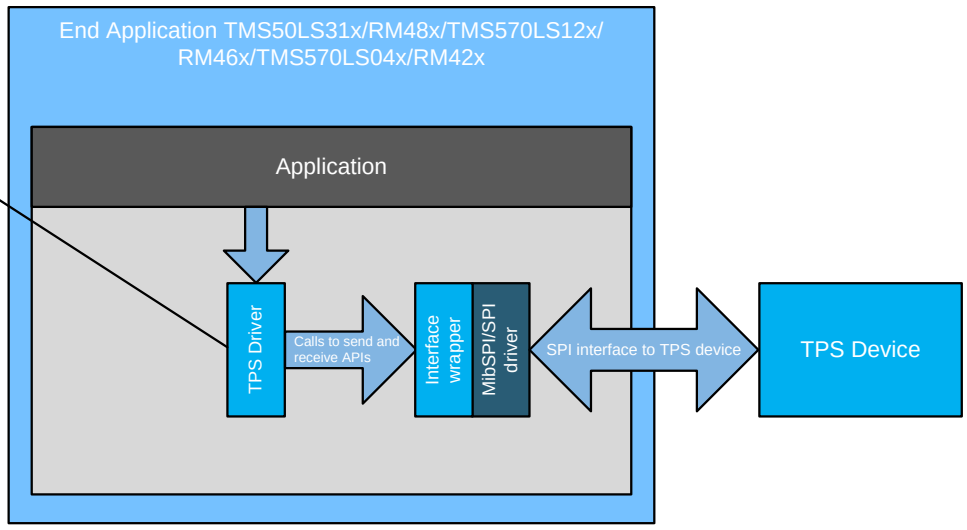

Figure 2. TPS Driver (Library) Usage in End Application

2.1.2 TPS Device Features

The block diagram of the TPS device shown in Figure 3 provides fine details about the features of the TPS device. The TPS library provides extensive API to use all the features mentioned in the diagram of the TPS device.

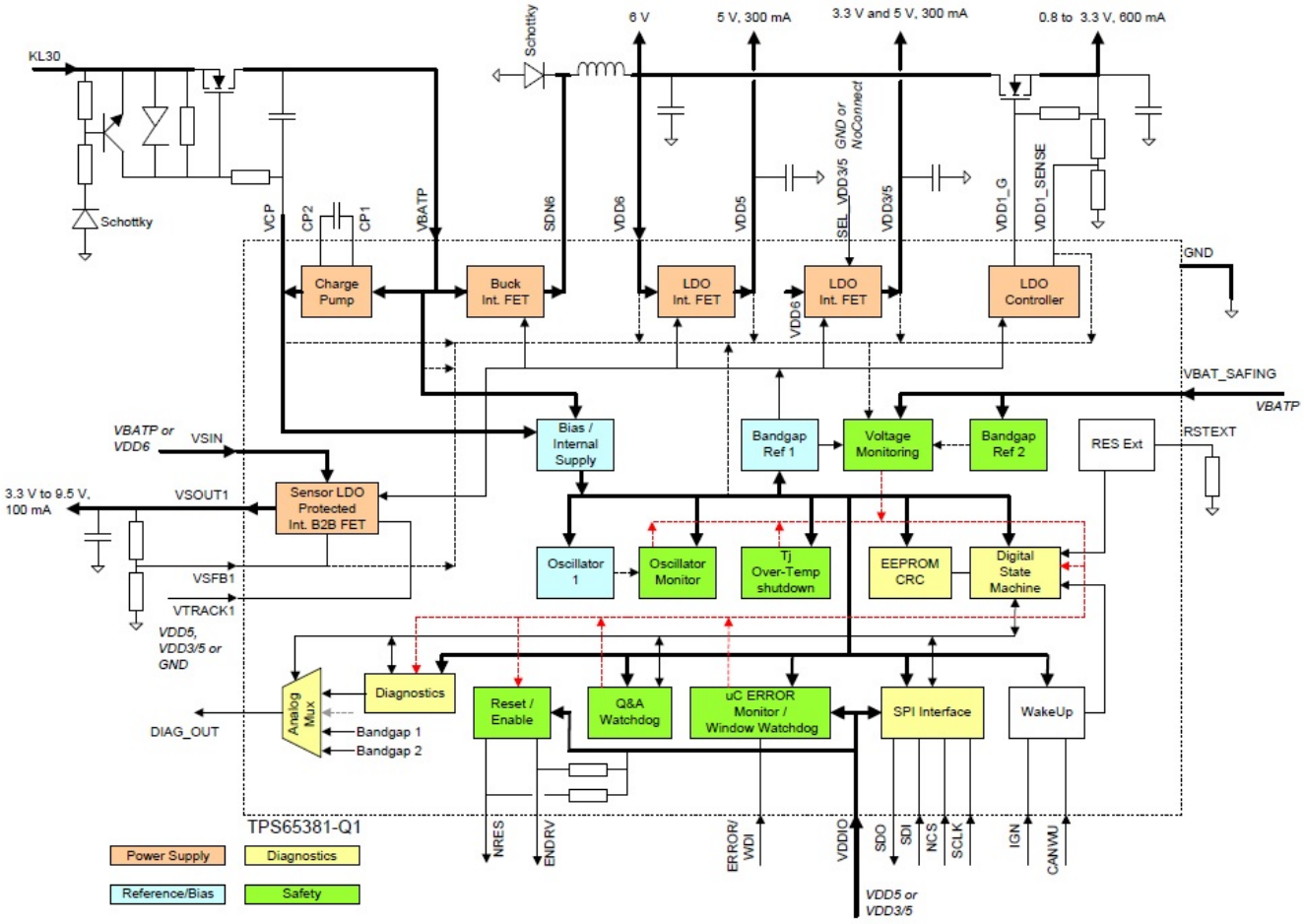

Figure 3. TPS Device Block Diagram

2.1.3 Interfacing TPS Device with the Hercules™ Processors

Figure 4 shows an example of how the TPS device is interfaced with a Hercules™ processor. Figure 4 gives an overview of various connections that must be made to the TPS device from the Hercules™ processors. Shown are the various peripherals and ports that are used for communicating with the TPS device.

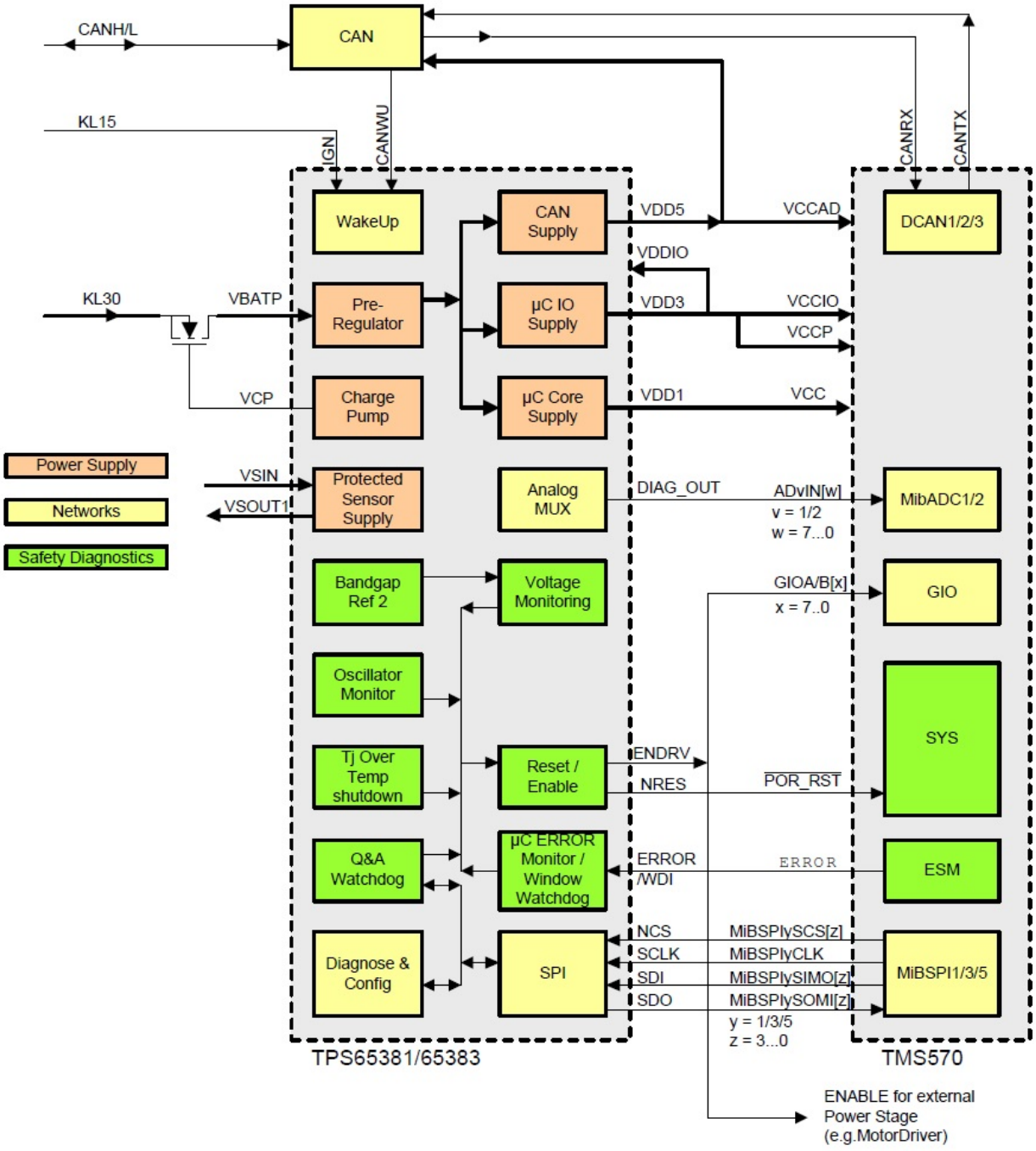

Figure 4. TPS Device Interfaced with a Hercules™ Processor

2.2 Targeted Applications

The Hercules™ MCU family is targeted at general purpose safety applications. Multiple safety applications were analyzed during the concept phase in order to support Safety Element out of Context (SEooC) development, according to ISO 26262-10:2012. Example target applications include:

- Automotive braking systems, including anti-lock braking (ABS), anti-lock braking with traction control (ABS+TC), and electronic stability control (ESC)
- Motor control systems, particularly electronic power steering (EPS) systems and electrical vehicle (EV) power train
- General purpose safety computation, such as integrated sensor cluster processing and vehicle strategy generation in an active safety system
- Industrial automation such as programmable logic controllers (PLCs) and programmable automation controllers (PACs) for safety critical process control

In the case of overlapping requirements between target systems, TI has attempted to design the device respecting the most stringent requirements. For example, the fault tolerant time intervals for timer logic in an ESC application are typically on the order of 100 ms. In an EPS application, the fault tolerant time interval is typically on the order of 10 ms. In such cases, TI has performed timer-subsystem analysis respecting less than 10 ms fault tolerant time interval. While TI considered certain applications during the development of these devices, this should not restrict a customer who wishes to implement other systems. With all safety critical components, rationalization of the component safety concept to the system safety concept must be executed by the system integrator.

2.3 Product Safety Constraints

For safety components developed according to many safety standards, it is expected that the component safety manual will provide a list of product safety constraints. For a simple component, or more complex components developed for a single application, this is a reasonable response. However, the Hercules™ product family is a complex design and is not developed targeting a single, specific application.

Therefore, a single set of product safety constraints cannot govern all viable uses of the product. The *Detailed Safety Analysis Report* for the particular Hercules™ MCU (SPNU570) provides an example implementation of the Hercules™ product in a common system with relevant product safety constraints.

3 SafeTI™ Software Development Process

The software development model adopted here is the *V-Model* depicted in Figure 5 with each life-cycle phase ending with a cross-functional review called Checkpoint (CP) review. In some cases, the releases may have to iterate through the checkpoints multiple times. Approval to proceed to the next checkpoint is obtained at the end of the checkpoint review from identified stakeholders. Following are the six checkpoints covering all the life-cycle phases of the software development project:

- SW CP1: Software Project Commissioning
- SW CP2: Safety Requirements and Planning
- SW CP3a: Software Architecture, Unit Design, and Development
- SW CP3b: Software Unit Testing and Integration Testing
- SW CP4: Safety Software Testing and Release
- SW CP5: Software Project Closure

To ensure functional safety throughout the software life-cycle development, supporting processes like requirements management, configuration management, change management, tool qualification, safety assessment, safety audits, document management, and personnel management are defined and followed. Additional recommended techniques and measures in the targeted functional safety standard for the targeted SIL/ASIL are applied throughout the development life cycle. The rationale for selected techniques and measures are documented in the appropriate planning documents. A project software safety manager is assigned to ensure project-functional safety activities are planned and coordinated.

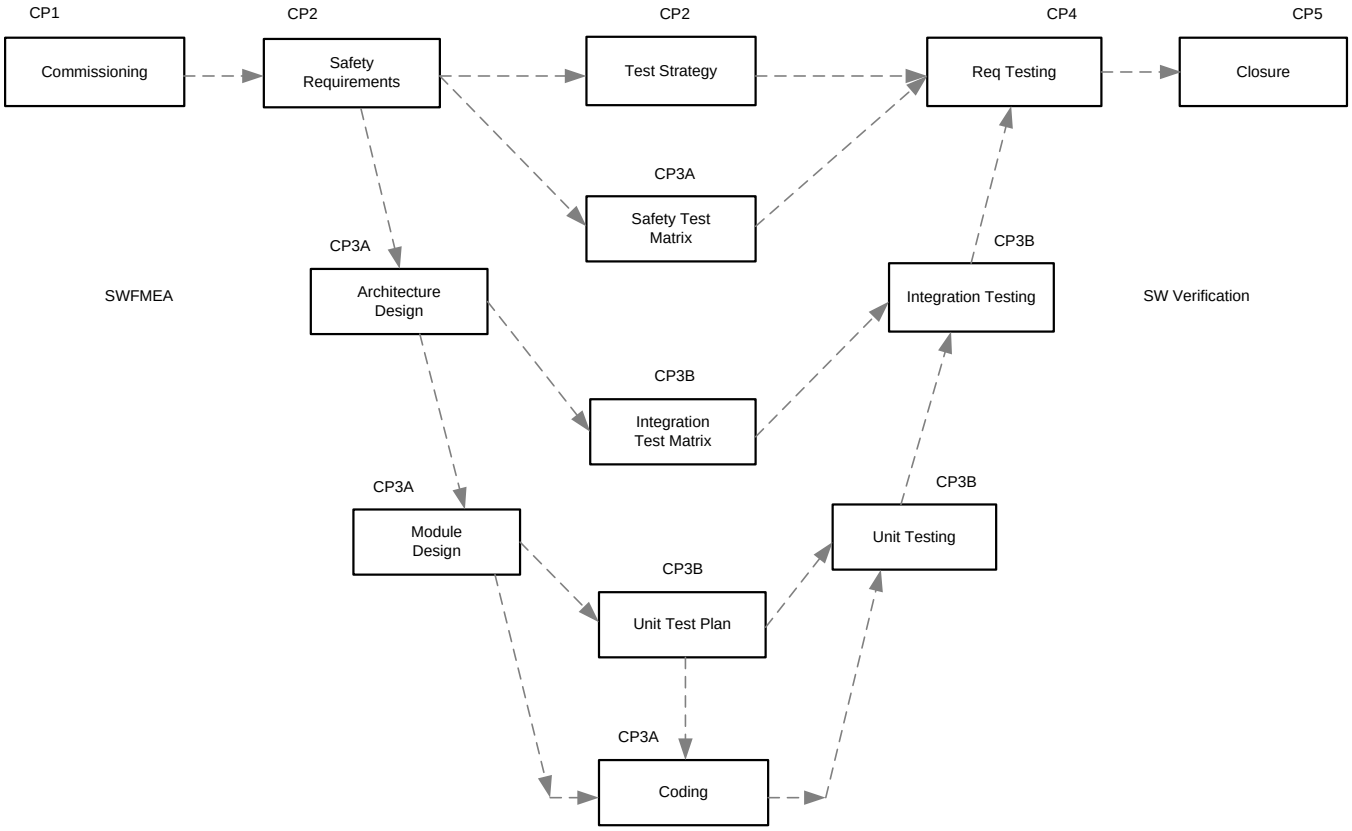

Figure 5. Software Development Process Based on the V-Model

4 Safety Assessment and Certification

For safety-critical development, it is necessary to manage both random and systematic faults. Texas Instruments has created a unique development process for safety-critical software, which reduces the probability of systematic failure. This process is built on top of the standard quality managed system software development process to meet specific requirements of IEC 61508-SIL3 and ISO 26262-ASIL D. The foundation quality managed software development process is compliant to TS 16949: 2009 standards and CMMI 1.3 Level 3 Model. The process implemented in the SafeTI™ Diagnostic Library is based on the software development process and is documented to meet safety standards.

Texas Instruments has been developing automotive microcontrollers and processors for safety critical and non-safety critical automotive applications for over twenty years. Automotive markets have strong requirements on quality management and high reliability of product. Though not explicitly developed for compliance to a functional safety standard, the TI standard new product development process already featured many elements necessary to manage systematic faults. This development process can be considered to be QM (Quality Managed), but does not achieve an IEC 61058 SIL (Safety Integrity Level) or ISO 26262 ASIL (Automotive Safety Integrity Level). The TI standard new product development process is certified compliant to ISO TS 16949 Bureau Veritas certificate USA-13036/1-TS (IATF certificate No 142823) as part of the South Campus support function. It is also certified to ISO 9001:2008 under Bureau Veritas Certificate US-005329-1 as part of the South Campus support function.

5 SafeTI™ Diagnostic Library Overview

Figure 6 shows the software stack in the perspective of the SafeTI™ Diagnostic Library. Hardware Abstraction Layer (HAL) is the lowest software layer. It contains software modules with direct access to the MCU and is responsible for system initialization. Diagnostic Library is a collection of functions for access to safety functions and response handlers for various safety mechanisms. Diagnostic Library runs in the context of the caller's protection environment and all responses are handled in the context of interrupt or exception.

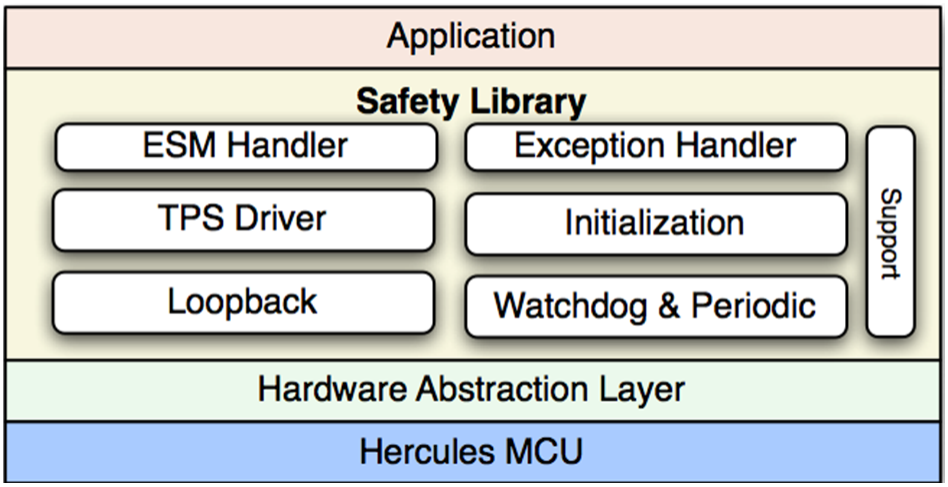

Figure 6. SafeTI™ Diagnostic Library Software Stack

Initialization

Startup block is responsible for configuring the safety mechanisms and detecting any failures at system boot (through Safety Tests). This block has a few Initialization APIs and depends on the APIs of the other blocks to execute various functions at self test (Self-Test, PBIST, LBIST, Configuration through HAL). The application may follow an initialization sequence as described in *Initialization of Hercules™ ARM Cortex-R4F Microcontrollers* (SPNA106).

Exception Handler

The R4/R4F/R5F CPU branches to an exception handler for handling failures at runtime. The following exceptions are handled:

- Prefetch Aborts (Precise)
- Data Abort (Precise and Imprecise)
- Undefined Instructions

These handlers are typically defined by the RTOS/HAL layers and, hence, are not provided by the library.

ESM Handler

ESM handler block is responsible for handling various errors at run time. The errors are processed for additional information and intimated to application through registered callbacks. Based on the safety requirements of the system, the application can use the provided information to take necessary steps.

Self-Tests and Fault Injection API

Diagnostic Library API can be called in fault injection and self-test modes.

- Fault Injections allow the application to induce faults and verify the fault handling in their application.
- Self-Test is a mechanism for providing latent fault diagnostics. It verifies the safety mechanisms available on the device.

Figure 7 shows the Safety Diagnostic Library features on the Hercules™ MCU devices based on the ARM Cortex™ R4 and ARM Cortex R4F devices. The same concept applies to the Hercules™ MCU based on ARM Cortex R5F core.

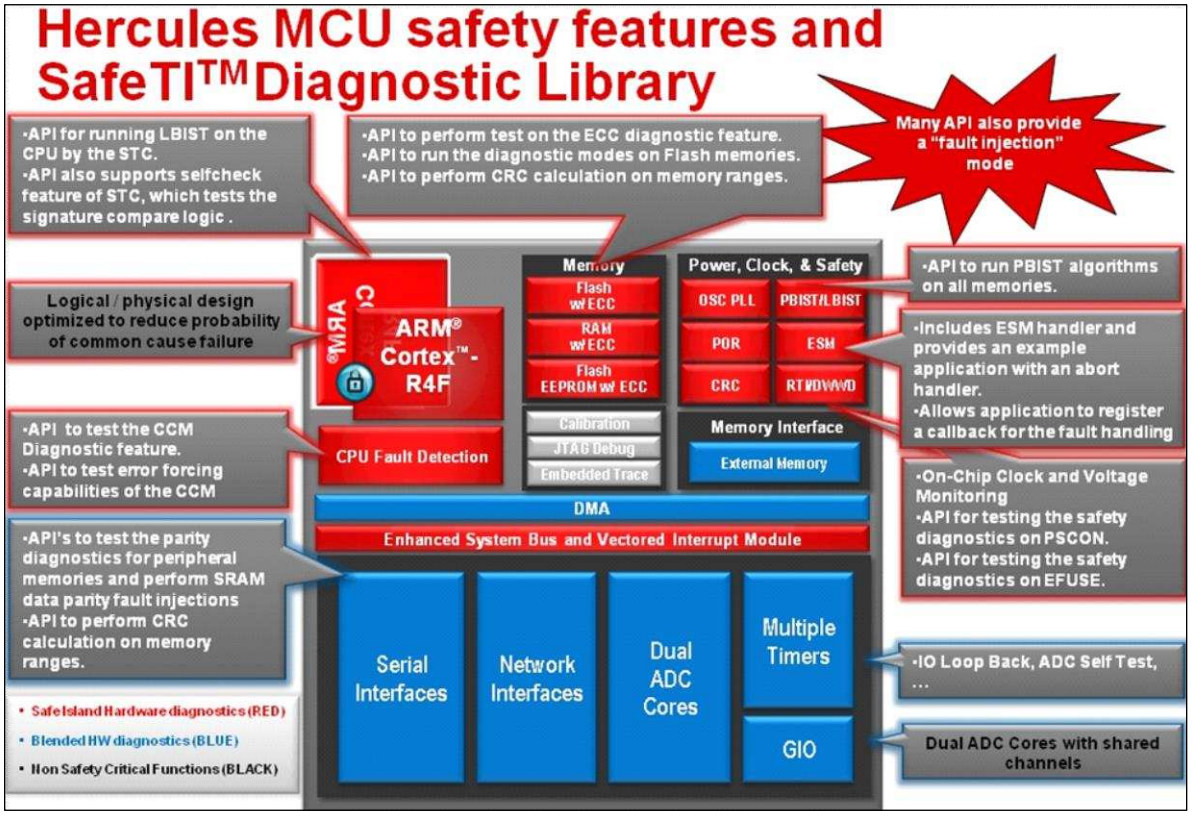

Figure 7. Safety Diagnostic Library Features on the Hercules™ MCU Devices

Criteria for the usage of the SafeTI™ Hercules™ Diagnostic Library API

The API provided by the Diagnostic Library are recommended to be used in a system for periodic tests on the fault diagnostics. It is expected that this is done in the diagnostic time interval (time interval set aside for Self-Test and when nothing else is running in the system). These API configure the fault diagnostics in special modes to check the function of the diagnostic. The return value of the Self-Test APIs indicate if the diagnostic is functioning as expected or if there is a fault in the device. Interrupting the operation of the Self-Test API can leave the system in an undefined state.

The Fault injection API is used to create faults at run time such that the application developer may be able to simulate faults and their handling during development. Similar to the Self-Test API, the Fault Injection API configures fault diagnostics in special modes to create the desired fault. It is possible to insert faults at any time. The above requirements imply that Diagnostic Library API is run as the highest-priority task in the application.

5.1 API Mapping of SafeTI™ Diagnostic Library Recommended Safety Functions

You, as a system and equipment manufacturer or designer, are responsible to ensure that your systems (and any TI hardware or software components incorporated in your systems) meet all applicable safety, regulatory, and system-level performance requirements. All application- and safety-related information in this document (including application descriptions, suggested safety measures, suggested TI products, and other materials) is provided for reference only. You understand and agree that your use of TI components in safety-critical applications is entirely at your risk, and that you (as buyer) agree to defend, indemnify, and hold TI harmless from any and all damages, claims, suits, or expense resulting from such use.

In this section, the safety mechanisms for each major functional block of the Hercules™ architecture are summarized and mapped to the APIs supported by the SafeTI™ Diagnostic Library. For more information on the safety mechanisms and the general assumption of use, see the safety manual for the Hercules™ device. The details of each safety mechanism can be found in the device-specific technical reference manual for the processor used.

Table 1. API Mapping

Device Partition	Unique Identifier	Safety Feature or Diagnostic	API Name	Remarks
Power Supply	PWR1	Voltage monitor (VMON)	Not Applicable	No software control, always enabled in hardware
	PWR2	External voltage supervisor	Not Applicable	Handled by the safety application
Power Management Module (PMM)	PMM1	Lockstep PSCON	SL_SelfTest_PSCON	
	PMM2	Privileged mode access and multi-bit keys for control registers	SL_SelfTest_PSCON	
	PMM3	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
	PMM4	Software readback of written configuration	Not Applicable	Written configuration is defined by the application.
Clock	CLK1	LPOCLKDET	Not Applicable	Handled by the safety application
	CLK2	PLL slip detector	ESM_Application_Callback	
	CLK3	Dual Clock Comparator (DCC)	ESM_Application_Callback	
	CLK4	External monitoring through ECLK	Not Applicable	Handled by the safety application
	CLK5A	Internal watchdog - DWD	Not Applicable	Handled by the safety application
	CLK5B	Internal watchdog - DWWD	Not Applicable	Handled by the safety application
	CLK5C	External watchdog	Not Applicable	Handled by the safety application
	CLK6	Periodic software readback of static clock configuration registers	Not Applicable	Static configuration is defined by the application.
	CLK7	Software readback of written configuration	Not Applicable	Written configuration is defined by the application.

Table 1. API Mapping (continued)

Device Partition	Unique Identifier	Safety Feature or Diagnostic	API Name	Remarks
Reset	RST1	External monitoring of warm reset	Not Applicable	Handled by the safety application
	RST2	Software check of last reset	SL_Init_Reset Reason	
	RST3	Software warm reset generation	SL_SW_Reset	
	RST4	Glitch filtering on reset pins	Not Applicable	No software control, always enabled in hardware
	RST5	Use of status shadow registers	SL_Init_ResetReason_XInfo	
	RST6	External watchdog	Not Applicable	Handled by the safety application
	RST7	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
	RST8	Software readback of written configuration	Not Applicable	Written configuration is defined by the application.
System Control	SYS1	Privileged mode access and multi-bit enable keys	Not Available	Not in scope for this release
	SYS2	Software readback of written configuration	Not Applicable	Written configuration is defined by the application.
	SYS3	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
Error Signaling Module (ESM)	ESM1	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
	ESM2A	Boot time software test of error path reporting	Fault Injection APIs may be used	
	ESM2B	Periodic software test of error path reporting	Fault Injection APIs may be used	
	ESM3	Use of status shadow registers	SL_Init_ResetReason_XInfo	
	ESM4	Software readback of written configuration	Not Applicable	Written configuration is defined by the application.

Table 1. API Mapping (continued)

Device Partition	Unique Identifier	Safety Feature or Diagnostic	API Name	Remarks
Cortex-R4FCentral Processing Unit (CPU)	CPU1	Lockstep compare	SL_SelfTest_CCMR4F	
	CPU2A	Boot time execution of LBIST STC	SL_SelfTest_STC	
	CPU2B	Periodic execution of LBIST STC	SL_SelfTest_STC	
	CPU3	MPU	Not Applicable	Handled by operation system/safety application
	CPU4	Online profiling using PMU	Not Applicable	Handled by the safety application
	CPU5A	Internal watchdog - DWD	Not Applicable	Handled by the safety application
	CPU5B	Internal watchdog - DWWD	Not Applicable	Handled by the safety application
	CPU5C	External watchdog	Not Applicable	Handled by the safety application
	CPU6	Illegal operation and instruction trapping	Not Available	Not in scope for this release
	CPU7	Software readback of written configuration	Not Applicable	Written configuration is defined by the application.
Primary Flash and Level 1 (L1) Interconnect	FLA1	Flash Data ECC	SL_SelfTest_Flash	
	FLA2	Hard error cache and livelock	Not Available	Not in scope for this release
	FLA3	Flash wrapper address ECC	SL_SelfTest_Flash	
	FLA4	Address parity	SL_SelfTest_Flash	
	FLA5A	Boot time hardware CRC check of Flash memory contents	SL_CRC_Calculate	
	FLA5B	Periodic hardware CRC check of Flash memory contents	SL_CRC_Calculate	
	FLA6	Bit multiplexing in Flash array	Not Applicable	CRC will indicate faults in flash
	FLA7	Flash sector protection	Not Applicable	External dependency on flash writing APIs
	FLA8	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
	FLA9	Software readback of written configuration	Not Applicable	Written configuration is defined by the application.

Table 1. API Mapping (continued)

Device Partition	Unique Identifier	Safety Feature or Diagnostic	API Name	Remarks
Flash Emulated EEPROM (FEE)	FEE1	FEE Data ECC	SL_SelfTest_FEE	
	FEE2A	Boot time hardware CRC check of FEE memory contents	SL_CRC_Calculate	
	FEE2B	Periodic hardware CRC check of FEE memory contents	SL_CRC_Calculate	
	FEE3	Bit multiplexing in FEE array	Not Applicable	CRC will indicate faults in flash
	FEE4	FEE sector protection	Not Applicable	External dependency on flash writing APIs
	FEE5	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
	FEE6	Software readback of written configuration	Not Applicable	Written configuration is defined by the application.
SRAM and Level 1 (L1) Interconnect	RAM1	Data ECC	SL_SelfTest_SRAM	
	RAM2	Hard error cache and livelock	Not Available	Not in scope for this release
	RAM3	Correctable ECC profiling	Not Available	Not in scope for this release
	RAM4	Address and control parity	SL_SelfTest_SRAM	
	RAM5	Redundant address decode	SL_SelfTest_SRAM	
	RAM6	Data and ECC storage in multiple physical banks	Not Applicable	Feature in Hardware
	RAM7A	Boot time PBIST check of SRAM	SL_SelfTest_PBIST	
	RAM7B	Periodic PBIST check of SRAM	SL_SelfTest_PBIST	
	RAM8	Bit multiplexing in SRAM array	Not Applicable	CRC will indicate faults in SRAM
	RAM9	Periodic hardware CRC check of SRAM contents	SL_CRC_Calculate	
	RAM10	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
	RAM11	Software readback of written configuration	Not Applicable	Written configuration is defined by the application.

Table 1. API Mapping (continued)

Device Partition	Unique Identifier	Safety Feature or Diagnostic	API Name	Remarks
Level 2 and Level 3 (L2 and L3) Interconnect	INC1	Error trapping (including peripheral slave error trapping)	SL_SelfTestL2L3Interconnect	
	INC2	PCR access management	Not Available	Not in scope for this release
	INC3A	Internal watchdog - DWD	Not Applicable	Handled by the safety application
	INC3B	Internal watchdog - DWWD	Not Applicable	Handled by the safety application
	INC3C	External watchdog	Not Applicable	Handled by the safety application
	INC4	Information redundancy	Not Available	Not in scope for this release
	INC5	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
	INC6A	Boot time software test of basic functionality including error tests	SL_SelfTestL2L3Interconnect	
	INC6B	Periodic software test of basic functionality including error tests	SL_SelfTestL2L3Interconnect	
	INC7	Software readback of written configuration	Not Applicable	Written configuration is defined by the application.
EFuse Static Configuration	EFU1	Boot time autoload self-test	SL_SelfTest_EFuse	
	EFU2	E-fuse ECC	Not Available	Not in scope for this release
One Time Programmable (OTP) Flash Static Configuration	OTP1	Boot time autoload self-test	Not Available	Not in scope for this release
	OTP2	OTP ECC	Not Available	Not in scope for this release
Input/Output (I/O) Multiplexing (IOMM)	IOM1	Locking control registers	Not Applicable	Hardware feature with no error response for faults
	IOM2	Master ID filtering	Not Applicable	
	IOM3	Error trapping	Not Available	Not in scope for this release
	IOM4	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
	IOM5A	Boot time software test of function using peripherals with analog I/O loopback including error tests	Not Available	Not in scope for this release
	IOM5B	Periodic software test of function using peripherals with analog I/O loopback including error tests	Not Available	Not in scope for this release
	IOM6	Software readback of written configuration	Not Applicable	Written configuration is defined by the application.

Table 1. API Mapping (continued)

Device Partition	Unique Identifier	Safety Feature or Diagnostic	API Name	Remarks
Vectored Interrupt Module (VIM)	VIM1	VIM SRAM Data Parity	Not Available	Not in scope for this release
	VIM2A	Boot time PBIST check of VIM SRAM	SL_SelfTest_PBIST	
	VIM2B	Periodic PBIST check of VIM SRAM	SL_SelfTest_PBIST	
	VIM3	Bit multiplexing in VIM SRAM array	Not Applicable	CRC will indicate faults in VIM SRAM
	VIM4	Periodic hardware CRC check of VIM SRAM contents	SL_CRC_Calculate	
	VIM5	Periodic software test of VIM functionality	Not Available	Not in scope for this release
	VIM6	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
	VIM7	Software readback of written configuration	Not Applicable	Written configuration is defined by the application.
	VIM8A	Internal watchdog - DWD	Not Applicable	Handled by the safety application
	VIM8B	Internal watchdog - DWWD	Not Applicable	Handled by the safety application
	VIM8C	External watchdog	Not Applicable	Handled by the safety application
Real Time Interrupt (RTI) Operating System Timer	RTI1	1oo2 software voting using secondary free running counter	Not Available	Not in scope for this release
	RTI2A	Internal watchdog - DWD	Not Applicable	Handled by safety application
	RTI2B	Internal watchdog - DWWD	Not Applicable	Handled by safety application
	RTI2C	External watchdog	Not Applicable	Handled by safety application
	RTI3	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.

Table 1. API Mapping (continued)

Device Partition	Unique Identifier	Safety Feature or Diagnostic	API Name	Remarks
Direct Memory Access (DMA)	DMA1	Memory protection unit for bus master accesses	Not Available	Not in scope for this release
	DMA2	Non-privileged bus master access	Not Available	Not in scope for this release
	DMA3	Information redundancy	Not Available	Not in scope for this release
	DMA4	DMA SRAM Data Parity	Not Available	Not in scope for this release
	DMA5A	Boot time PBIST check of DMA SRAM	SL_SelfTest_PBIST	
	DMA5B	Periodic PBIST check of DMA SRAM	SL_SelfTest_PBIST	
	DMA6	Bit multiplexing in DMA SRAM array	Not Available	Not in scope for this release
	DMA7	Periodic hardware CRC check of DMA SRAM contents	SL_CRC_Calculate	
	DMA8	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
	DMA9A	Boot time software test of basic functionality including error tests	Not Available	Not in scope for this release
	DMA9B	Periodic software test of basic functionality including error tests	Not Available	Not in scope for this release
High-End Timer (N2HET) Including HET Transfer Unit (HTU)	HET1	Memory protection unit for bus master accesses	Not Available	Not in scope for this release
	HET2	Information redundancy	Not Available	Not in scope for this release
	HET3	Use of DCC as program sequence watchdog	Not Available	Not in scope for this release
	HET4	Monitoring by second N2HET	Not Available	Not in scope for this release
	HET5A	Boot time software test of function using I/O loopback	Not Available	Not in scope for this release
	HET5B	Periodic software test of function using I/O loopback	Not Available	Not in scope for this release
	HET6	N2HET/HTU SRAM Data Parity	Not Available	Not in scope for this release
	HET7A	Boot time PBIST check of N2HET/HTU SRAM	SL_SelfTest_PBIST	Not in scope for this release
	HET7B	Periodic PBIST check of N2HET/HTU SRAM	SL_SelfTest_PBIST	Not in scope for this release
	HET8	Bit multiplexing in N2HET/HTU SRAM array	Not Available	Feature in Hardware
	HET9	Periodic hardware CRC check of N2HET/HTU SRAM contents	SL_CRC_Calculate	
	HET10	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.

Table 1. API Mapping (continued)

Device Partition	Unique Identifier	Safety Feature or Diagnostic	API Name	Remarks
Multi-Buffered Analog to Digital Converter (MibADC)	ADC1	Boot time input self test	SL_SelfTest_ADC	
	ADC2A	Boot time converter calibration	SL_adcCalibration	
	ADC2B	Periodic converter calibration	SL_adcCalibration	
	ADC3	Information redundancy techniques	Not Available	Not in scope for this release
	ADC4	MibADC SRAM Data Parity	SL_SelfTest_ADC	
	ADC5A	Boot time PBIST check of MibADC SRAM	SL_SelfTest_PBIST	
	ADC5B	Periodic PBIST check of MibADC SRAM	SL_SelfTest_PBIST	
	ADC6	Bit multiplexing in MibADC SRAM array	Not Available	Feature in Hardware
	ADC7	Periodic hardware CRC check of MibADC SRAM contents	SL_CRC_Calculate	
	ADC8	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
Multi-Buffered Serial Peripheral Interface (MibSPI)	MSP1A	Boot time software test of function using I/O loopback	Not Available	Not in scope for this release
	MSP1B	Periodic software test of function using I/O loopback	Not Available	Not in scope for this release
	MSP2	Parity in message	Not Available	Not in scope for this release
	MSP3	Information redundancy techniques	Not Available	Not in scope for this release
	MSP4	MibSPI SRAM Data Parity	Not Available	Not in scope for this release
	MSP5A	Boot time PBIST check of MibSPI SRAM	SL_SelfTest_PBIST	
	MSP5B	Periodic PBIST check of MibSPI SRAM	SL_SelfTest_PBIST	
	MSP6	Bit multiplexing in MibSPI SRAM array	Not Available	Feature in Hardware
	MSP7	Periodic hardware CRC check of MibSPI SRAM contents	SL_CRC_Calculate	
	MSP8	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.

Table 1. API Mapping (continued)

Device Partition	Unique Identifier	Safety Feature or Diagnostic	API Name	Remarks
Serial Peripheral Interface (SPI)	SPI1A	Boot time software test of function using I/O loopback	Not Available	Not in scope for this release
	SPI1B	Periodic software test of function using I/O loopback	Not Available	Not in scope for this release
	SPI2	Parity in message	Not Available	Not in scope for this release
	SPI3	Information redundancy techniques	Not Available	Not in scope for this release
	SPI4	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
Inter-Integrated Circuit (I2C)	IIC1A	Boot time software test of function	Not Available	Not in scope for this release
	IIC1B	Periodic software test of function	Not Available	Not in scope for this release
	IIC2	Information redundancy techniques	Not Available	Not in scope for this release
	IIC3	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
Serial Communications Interface (SCI)	SCI1A	Boot time software test of function using I/O loopback	Not Available	Not in scope for this release
	SCI1B	Periodic software test of function using I/O loopback	Not Available	Not in scope for this release
	SCI2	Information redundancy techniques	Not Available	Not in scope for this release
	SCI3	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
Local Interconnect Network (LIN)	LIN1A	Boot time software test of function using I/O loopback	Not Available	Not in scope for this release
	LIN1B	Periodic software test of function using I/O loopback	Not Available	Not in scope for this release
	LIN2	Information redundancy techniques including end to end safing	Not Available	Not in scope for this release
	LIN3	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.

Table 1. API Mapping (continued)

Device Partition	Unique Identifier	Safety Feature or Diagnostic	API Name	Remarks
Controller Area Network (DCAN)	CAN1A	Boot time software test of function using I/O loopback	Not Available	Not in scope for this release
	CAN1B	Periodic software test of function using I/O loopback	Not Available	Not in scope for this release
	CAN2	Information redundancy techniques including end to end safing	Not Available	Not in scope for this release
	CAN3	DCAN SRAM Data Parity	Not Available	Not in scope for this release
	CAN4A	Boot time PBIST check of DCAN SRAM	SL_SelfTest_PBIST	
	CAN4B	Periodic PBIST check of DCAN SRAM	SL_SelfTest_PBIST	
	CAN5	Bit multiplexing in DCAN SRAM array	Not Available	Feature in Hardware
	CAN6	Periodic hardware CRC check of DCAN SRAM contents	SL_CRC_Calculate	
	CAN7	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
FlexRay Including FlexRay Transfer Unit (FTU)	FRY1	Memory protection unit for bus master accesses	Not Available	Not in scope for this release
	FRY2	Non-privileged bus master access	Not Available	Not in scope for this release
	FRY3A	Boot time software test of function using I/O loopback in PHY	Not Available	Not in scope for this release
	FRY3B	Periodic software test of function using I/O loopback in transceiver	Not Available	Not in scope for this release
	FRY4	Information redundancy techniques including end to end safing	Not Available	Not in scope for this release
	FRY5	1oo2 Voting using Both FlexRay Channels	Not Available	Not in scope for this release
	FRY6	FlexRay and FTU SRAM Data Parity	Not Available	Not in scope for this release
	FRY7A	Boot time PBIST check of FlexRay and FTU SRAM	SL_SelfTest_PBIST	
	FRY7B	Periodic PBIST check of FlexRay and FTU SRAM	SL_SelfTest_PBIST	
	FRY8	Bit multiplexing in FlexRay and FTU SRAM array	Not Available	Feature in Hardware
	FRY9	Periodic hardware CRC check of FlexRay and FTU SRAM contents	SL_CRC_Calculate	
	FRY10	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.

Table 1. API Mapping (continued)

Device Partition	Unique Identifier	Safety Feature or Diagnostic	API Name	Remarks
General Purpose Input/Output (GIO)	GIO1A	Boot time software test of function using I/O checking	Not Available	Not in scope for this release
	GIO1B	Periodic software test of function using I/O checking	Not Available	Not in scope for this release
	GIO2	Information redundancy techniques	Not Available	Not in scope for this release
	GIO3	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
Ethernet	ETH1	Non-privileged bus master access	Not Available	Not in scope for this release
	ETH2A	Boot time software test of function using I/O loopback in PHY	Not Available	Not in scope for this release
	ETH2B	Periodic software test of function using I/O loopback in PHY	Not Available	Not in scope for this release
	ETH3	Information redundancy techniques including end to end safing	Not Available	Not in scope for this release
	ETH4A	Boot time PBIST check of Ethernet SRAM	SL_SelfTest_PBIST	
	ETH4B	Periodic PBIST check of Ethernet SRAM	SL_SelfTest_PBIST	
	ETH5	Bit multiplexing in Ethernet SRAM array	Not Available	Feature in Hardware
	ETH6	Periodic hardware CRC check of Ethernet SRAM contents	SL_CRC_Calculate	
	ETH7	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
Universal Serial Bus (USB)	USB1	Non-privileged bus master access	Not Available	Not in scope for this release
	USB2A	Boot time software test of function using I/O loopback in PHY	Not Available	Not in scope for this release
	USB2B	Periodic software test of function using I/O loopback in PHY	Not Available	Not in scope for this release
	USB3	Information redundancy techniques	Not Available	Not in scope for this release
	USB4A	Boot time PBIST check of USB SRAM	SL_SelfTest_PBIST	Not in scope for this release
	USB4B	Periodic PBIST check of USB SRAM	SL_SelfTest_PBIST	Not in scope for this release
	USB5	Bit multiplexing in USB SRAM array	Not Available	Feature in Hardware
	USB6	Periodic hardware CRC check of USB SRAM contents	SL_CRC_Calculate	
	USB7	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.

Table 1. API Mapping (continued)

Device Partition	Unique Identifier	Safety Feature or Diagnostic	API Name	Remarks
External Memory Interface (EMIF)	EMF1	Information redundancy techniques	Not Available	Not in scope for this release
	EMF2A	Boot time hardware CRC check of external memory	SL_CRC_Calculate	
	EMF2B	Periodic hardware CRC check of external memory	SL_CRC_Calculate	
	EMF3	Periodic software readback of static configuration registers	Not Applicable	Static configuration is defined by the application.
Joint Technical Action Group (JTAG) Debug/Trace/Calibration Access	JTG1	Hardware disable of JTAG port	Not Available	Hardware Configuration
	JTG2	Lockout of JTAG access using AJSM	Not Available	Configured by Application
Cortex-R4F Central Processing Unit (CPU) Debug and Trace	DBG1	Hardware disable of JTAG port	Not Available	Hardware Configuration
	DBG2	Lockout of JTAG access using AJSM	Not Available	Configured by Application
	DBG3	Use MPUs to block access to memory-mapped debug	Not Available	Configured by Application
	DBG4	Use of CoreSight debug logic key enable scheme	Not Available	Configured by Application
Data Modification Module (DMM)	DMM1	Hardware disable of JTAG port	Not Available	Hardware Configuration
	DMM2	Lockout of JTAG access using AJSM	Not Available	Configured by Application
	DMM3	Use MPUs to block access to memory-mapped debug	Not Available	Configured by Application
	DMM4	Disable DMM pin interface	Not Available	Hardware Configuration
RAM Trace Port (RTP)	RTP1	Hardware disable of JTAG port	Not Available	Hardware Configuration
	RTP2	Lockout of JTAG access using AJSM	Not Available	Configured by Application
	RTP3	Use MPUs to block access to memory-mapped debug	Not Available	Configured by Application
	RTP4	Disable RTP pin interface	Not Available	Hardware Configuration
Parameter Overlay Module (POM)	POM1	Hardware disable of JTAG port	Not Available	Hardware Configuration
	POM2	Lockout of JTAG access using AJSM	Not Available	Configured by Application
	POM3	Use MPUs to block access to memory-mapped debug	Not Available	Configured by Application
	POM4	Use of CoreSight debug logic key enable scheme	Not Available	Configured by Application

Table 2 shows the mapping of the safety requirements derived for the TPS Driver development to the corresponding APIs.

NOTE: These APIs are documented in the TPS Driver - User's Guide - vX.Y.Z.chm (X.Y.Z corresponds to the version of the SafeTI™ Diagnostic Library) available in the docs folder of the SafeTI™ Diagnostic Library installation.

Table 2. TPS Driver API Mapping to Safety Requirements

Unique Identifier	Object Heading	Object Text	API Mapping
TPS_SR22	VMON	Provide API for voltage monitoring	TPS_GetVMONStatus
TPS_SR23	VMON_1	Get VMON trim error status. (Command RD_SAFETY_STAT_4 can be used.)	TPS_GetVMONStatus
TPS_SR24	VMON_2	Provide an API which aggreage the VMON status registers and return it to Application.	TPS_GetVMONStatus
TPS_SR25	Junction Temperature Monitoring and Current Limiting	"Provide an API to return the status (whether overvoltage or undervoltage or current limit exceeded) of voltage source VDD3/5, VDD5, and VSOUT.(command RD_SAFETY_STAT_1 can be used to implement this API.)"	TPS_GetJnTempandCurrentLimitStatus
TPS_SR27	TPS Interface	Provide TPS Interfacing functions	"TpsIf_GetRegister TpsIf_GetRegisterBitField TpsIf_SetRegister TpsIf_SetRegisterBitField TpsIf_SetRegisterBitFieldVerify "
TPS_SR28	TPS interface_1	"Provide Interfacing api's to TPS so as to effectively set,clear verify and recheck the variour registers inside the TPS module"	"TpsIf_GetRegister TpsIf_GetRegisterBitField TpsIf_SetRegister TpsIf_SetRegisterBitField TpsIf_SetRegisterBitFieldVerify "
TPS_SR29	TPS interface_2	The TPS uses SPI/MibSPI interface for communicating to the MCU.The TPS init should take care of initializing function pointers for the Send and the Receive API's	TPS_TpsIf_Init
TPS_SR319	TPS_Interface_3	Provide an API which does a selftest of the command parity logic	TpsIf_TestCommandParityLogic
TPS_SR320	TPS_Interface_4	Provide an API which does a selftest of wrong command logic	TpsIf_TestWrongCommandLogic
TPS_SR321	TPS_Interface_5	Provide an API which does the test of the SPI frame transmission to the TPS device	NA
TPS_SR322	TPS_Interface_6	Provide an API to test writes to the PWM Low register	TpsIf_SpiIFTestPwmlow
TPS_SR323	TPS_Interface_7	Provide an APi to get the Token Value.	TPS_UpdateActiveWDToken
TPS_SR31	ABIST	Provide ABIST Support	TPS_GetBISTRunningStatus
TPS_SR32	ABIST_1	Provide an API to manually trigger the analog built in selftest(The api should check for the proper preconditions and trigger the ABIST)	TPS_StartBIST
TPS_SR33	ABIST_2	Provide an API to get ABISt running status.(Command RD_SAFETY_STAT_3 can be used.)	TPS_GetBISTRunningStatus
TPS_SR35	ABIST_3	Provide API to give status of Analog Built in self test.The API should return the PASS status and if it is a failure then it should return the FAILURE type.(Command RD_SAFETY_STAT_3 can be used)	"TPS_GetLBISTTestStatus TPS_GoToSafeState "
TPS_SR36	MUX Diangnostics	Provide API for MUX diagnostics	TPS_EnableAMUXSignal
TPS_SR37	AMUX diagnostics	Provide and API for AMUX diagnostics.The API set the mux_cfg value to 10 and set proper Channel Number in the DIAG_MUX_SEL register and bring out required analog internal signal on the diag_out pin.The user will input the signal_name(enumeration) required as input to the API	TPS_DisableMUXDiagnostic
TPS_SR38	DMUX Diagnostics	Provide and API for DMUX diagnostics.The API set the mux_cfg value to 01 and set proper Channel group and Channel Number in the DIAG_MUX_SEL register and bring out required digital internal signal on the diag_out pin.The user will input the signal_name(enumeration) required as input to the API	TPS_EnableDMUXSignal
TPS_SR314	ADC Threshhold comparision	Provide ADC support so as to sample AMUX diagnostic values and compare them against the threshhold values.	" TPS_CheckEnabledAMUXSignalLimits TPS_DisableMUXDiagnostic TPS_EnableAMUXSignal"

Table 2. TPS Driver API Mapping to Safety Requirements (continued)

Unique Identifier	Object Heading	Object Text	API Mapping
TPS_SR39	BIST at startup	provide an API to enable and disable automatic BIST at startuup	TPS_ConfigureBISTatStartup
TPS_SR40	LBIST	Provide LBIST support	TPS_StartBIST
TPS_SR41	LBIST_1	Provide API to give status of LBIST.The API should return the PASS status and if it is a failure then it should return the FAILURE type.(Command RD_SAFETY_STAT_3 can be used)	"TPS_GetLBISTTestStatus TPS_GoToSafeState"
TPS_SR42	LBIST_2	Provide an API to get LBIST running status.(Command RD_SAFETY_STAT_3 can be used.)	"TPS_GetBISTRunningStatus TPS_GetEECRCCheckRunningStatus"
TPS_SR43	LBIST_3	API to trigger the LBIST run.The API should trigger the LBIST if the the preconditions or entry conditions are satisfied else API should return FALSE	TPS_StartBIST
TPS_SR44	Watchdog_General	General Watchdog support	TPS_GetWatchdogFailureStatus
TPS_SR45	Watchdog General_1	Watchdog Configuration API for Configuring the Watchdog settings and initializing the watchdog.(command WR_SAFETY_FUNC_CTRL can be used.)	TPS_SetWatchdogMode
TPS_SR46	Watchdog General_2	API for enabling and disabling of the watchdog reset.(command WR_SAFETY_FUNC_CTRL can be used.)	TPS_ConfigureWatchdogReset
TPS_SR47	Watchdog General_3	Provide and API to return the WDT fail count.(The command RD_SAFETY_STAT_2 can be used)	TPS_GetWatchdogFailCount
TPS_SR48	Watchdog General_4	Provide and API to return theWD error status.(The command RD_SAFETY_STAT_4 can be used.A boolean return value based on the return status)	TPS_GetWatchdogFailureStatus
TPS_SR49	Watchdog General_5	Provide an API to clear WD_FAIL status.(Command WR_SAFETY_ERR_STAT can be used.)	TPS_ClearWatchdogFailureStatusFlag
TPS_SR50	Watchdog General_6	Get Watchdog Fail Status API.(command D_SAFETY_ERR_STAT can be used.)	TPS_GetWatchdogFailureStatus
TPS_SR51	Watchdog General_7	"Provide Watchdog status API.The API updates the structure watchdog_status with relavant information.(such as seq_err,timeout,token_err etb.)(Command RD_WDT_STATUS can be used.)"	TPS_GetWatchdogErrorType
TPS_SR52	Watchdog General_8	Provide Watchdog Synchronization API or make the syncronization part of the Configuration API	TPS_ConfigureWatchdogWindows
TPS_SR53	Watchdog(WDTI Configuration)	WDTI watchdog support	TPS_WatchdogInit
TPS_SR54	Watchdog(WDTI Configuration)_1	The WatchDog Configuration API should enable configuring of the watchdog in WDTI mode.	TPS_WatchdogInit
TPS_SR55	Watchdog(WDTI Configuration)_2	Provide an API to Enable the PWM mode for the WD/ERROR Pin	TPS_ConfigureErrorMonitoring
TPS_SR56	Watchdog(Q&A configuration)	Q&A watchdog support	"TPS_ClearWatchdogFailureStatus Flag TPS_ConfigureErrorMonitoring TPS_ConfigureWatchdogReset TPS_ConfigureWatchdogWindows TPS_GetWatchdogFailureStatus TPS_SendWdgResponse TPS_SetWatchdogMode "
TPS_SR57	Watchdog(Q&A configuration)_1	The WatchDog Configuration API should enable configuring of the watchdog in Q&A mode.	TPS_SetWatchdogMode
TPS_SR58	Watchdog(Q&A configuration)_2	Provide an API to set Close window duration.(The command WR_WDT_WIN1_CFG can be used.)	TPS_ConfigureWatchdogWindows
TPS_SR59	Watchdog(Q&A configuration)_3	Provide an API to set Open window duration.(The command WR_WDT_WIN2_CFG can be used.)	TPS_ConfigureWatchdogWindows
TPS_SR60	Watchdog(Q&A configuration)_4	A send watchdog answer API which calculates the answer for given token and returns it to application	TPS_SendWdgResponse
TPS_SR324	Watchdog(Q&A configuration)_5	"A get watchdog answer count API which retuns the current now of answers sent to TPS "	TPS_GetWatchdogAnswerCount
TPS_SR61	MCU ERROR handling_1	API for doing a self test on ERROR signal monitoring.The API should force error pin on the MCU to high and check whether the ERROR_PIN_FAIL flag is set.	TPS_TestErrorPinMonitoring
TPS_SR62	Configration register protection	Provide support for configuration register protection	TPS_ProtectConfigurationRegisters

Table 2. TPS Driver API Mapping to Safety Requirements (continued)

Unique Identifier	Object Heading	Object Text	API Mapping
TPS_SR63	Device Configuration Register Protection_1	Provide API for locking of registers.((the command SW_LOCK can be used with data 0X55) or command WR_SAFETY_ERR_CFG can be used)	TPS_ProtectConfigurationRegisters
TPS_SR64	Device Configuration Register Protection_2	Provide API for unlocking of registers.(the command SW_UNLOCK can be used with data 0X55 or command WR_SAFETY_ERR_CFG can be used.)	TPS_ProtectConfigurationRegisters
TPS_SR65	Safety Check Control	Provide support for setting Safety Check Control	TPS_ConfigureSafetyCheckControl
TPS_SR72	CRC	Provide CRC support for TPS driver	"TPS_CalculateCRC8 TPS_GetCRCErrorStatus TPS_InitializeDatastringforCRCCalculation"
TPS_SR73	CRC_1	API to enable and disable CFG register CRC check.(command WR_SAFETY_CHECK_CTRL can be used)	TPS_ConfigureSafetyCheckControl
TPS_SR74	CRC_2	API to trigger EE CRC check (command WR_SAFETY_BIST_CTRL can be used)	TPS_StartEECRCCheck
TPS_SR75	CRC_3	Provide API for Calculation of the CRC on Safety Critical Registers	TPS_CalculateCRC8
TPS_SR76	CRC_4	Provide and Api to return the CRC error status.The API can have a parameter which selcts whether the error status of EEPROM or Configuration register needs to be returned.(command RD_SAFETY_STAT_2 can be used)	TPS_GetCRCErrorStatus
TPS_SR325	NRES_MONITORING	Provide an API to enable and disable the NRES pin monitoring	TPS_ConfigureNRESMonitoring
TPS_SR83	ERROR_STATUS	Provide Support for reading and clearing device error status and count	TPS_ClearDeviceErrorCount
TPS_SR84	ERROR_STATUS_1	Provide API that provides status of ERR_PIN_FAIL flag.(The command RD_SAFETY_ERR_STAT can be used.)	TPS_GetErrorPinFailureStatusFlag
TPS_SR85	ERROR_STATUS_2	Provide an API to clear ERROR_PIN_FAIL.(Command WR_SAFETY_ERR_STAT can be used.)	TPS_ClearErrorPinFailureStatusFlag
TPS_SR86	ERROR_STATUS_3	Provide an API to clear device error count.(Command WR_SAFETY_ERR_STAT can be used.)	TPS_ClearDeviceErrorCount
TPS_SR87	ERROR_STATUS_4	Provide an API to get the device error count.(Command WR_SAFETY_ERR_STAT can be used.)	TPS_GetDeviceErrorCount
TPS_SR91	Fault_Injection	Provide support for fault injection API's which can inject faults and help in observing the device or system behavior	"TPS_FaultInjectCRC TPS_FaultInjectWD"
TPS_SR92	Fault_Injection_1	Fault Injection Watchdog.Try to inject fault in watchdog by not forwarding the watchdog answer	TPS_FaultInjectWD
TPS_SR93	Fault_Injection_2	Fault iInjection CRC.Inject fault in crc for the device configuration registers.	TPS_FaultInjectCRC
TPS_SR313	Fault_Injection_3	Fault Injection EN_DRV.Inject fault in so that the status flag EN_DRV_ERR gets set in the TPS register Safety_Stat_4	NA
TPS_SR315	Safe State	Provide an API which pushes the TPS to the Safe State.There are various implementations possible which may put the TPS to safe state.Select feasible solution and implement the go to safe state API.	TPS_GoToSafeState
TPS_SR328	Readback_Static_Configuration	Provide API to Read Back the Static Configuration.(Safety Config + Dev Config)	TPS_RegReadBackandCompare

6 System Requirements

Section 6.1 and Section 6.2 outlines the hardware and software requirements to use the SafeTI™ Diagnostic Library and TPS Driver

6.1 Software Requirements

Following are the software requirements for using the SafeTI™ Diagnostic Library:

1. CCS 5.4 or newer version using Codegen tools version 5.0.4
2. HALCoGen 3.04.00 or newer version
3. nowECC v2.17 or newer version

Following are the software requirements for using the TPS Driver:

1. CCS 5.5 or newer version using Codegen tools version 5.1.6
2. HALCoGen 4.00.00 or newer version

6.2 Hardware Requirements

Following are the hardware requirements for using the SafeTI™ Diagnostic Library:

1. Device specific TI Hercules™ MCU Development Kit

Following are the hardware requirements for using the TPS Driver:

1. TMS570LS3137 Hitex Safety Kit, RM48L952 Hitex Safety Kit, TMS570LS1227 Control Card, RM46 Control card
2. Library can be used across the Hercules™ family of processors but has not been tested on platforms other than those mentioned above

7 Failure Modes and Effects Analysis Report for SafeTI™ Diagnostic Library (TPS Driver FMEA not available)

Table 3. Failure Modes and Effects

Sr. No.	Module / Function	Potential Failure mode	Potential Effect of Failure Mode	Potential Causes of Failure Mode	Prevention	Detection
1	ESM Handling	In case of a real fault in the system, the ESM handler reports an incorrect ESM event number to the user.	End user application identifies incorrect fault. The application fault handler may invoke fault handling different from the required action.	The ESM handler does not decode the ESM event correctly. An ESM Event (real fault in the hardware) will occur	ESM handler includes group and channel information. Verified through code review/test case review and testing.	Code Review. Testing using fault injection.
2	ESM Handling	Incorrectly identifying real faults as simulated Faults due to Safety diagnostic library™ logic. (Library code assumes diagnostic mode in place of real failure/fault injection mode)	End user application does not get notification of real fault in the system.	Implementation of the ESM handler/SDL. An ESM Event (real fault in the hardware) will occur	Flag indicates fault caused due to Safety Library Selftest or fault injection. Verified through test case updating, test case review, and testing.	Code review
3	ESM Handling	Incorrectly identifying simulated faults as real Faults due to SL logic.	End user application identifies incorrect fault resulting in wrong actions being taken.	Implementation of the ESM handler/SDL.	SDL Sets a flag to indicate the fault was an injected (Simulated) Fault. Several other checks for fault creation condition are in place.	Code review and testing
4	Application Privilege Level	SL API is invoked in user mode.	Application task invoking SL API is in non-privilege mode.	SL API adds checks for privilege level, and if not met, does not execute the test, returns a failure. This is a documented requirement for SL API usage.	Unit test cases to test SL API in user mode.	
5	PBIST	Running PBIST on SRAM with code in the SRAM	Prefetch abort or invalid operation	Application in SRAM invoking PBIST algorithm on SRAM	Updated documentation to include a note against use of PBIST for SRAM.	Testing at application level
6	PBIST	Running PBIST on RAMs at runtime can return invalid data.	Corruption of data resulting in invalid operation of application.	Application invoking PBIST on SRAM	Updated documentation to include a note against use of PBIST for SRAM.	Testing at application level
7	Safety loop	Violation of time slots for safety loop.	System instability resulting in system crash	Application design issue, not allocating enough time for Safety loop cycle, or interrupting safety loop cycle by a higher priority event/task	Documentation added to require application to keep the profiling data in mind when designing safety loop.	Testing at application level
8	Safety loop	Due to misconfiguration/masking at application config level (ESM) the Diagnostic mode/ESM status is not reset.	Diagnostic mode may remain set. ESM status will not be reset.	Application level masking of the ESM event.	Safety Library Design handles diagnostic mode configuration.	Design/Code review
9	ESM Handling	Safety library fails to report ESM error for fault injection mode	Application will not be able to test fault handling feature for the ESM event.	Due to misconfiguration/masking inside Selftest/fault injection API.	Check in design/code to verify if API is for self test or for fault injection, and masking of events accordingly (only for self test mode).	Code review

8 New in this Release

The following outlines the defects and enhancements resolved in this release:

1. Maintenance update (see bug fixes section in release notes).
2. Revised Safety Software Manual for SafeTI™ Diagnostic Library.
3. Updated User Guides for SafeTI™ Diagnostic Library and TPS Driver.
4. Additional Testing to cover bugs fixed.

9 Fixed in this Release

Refer to release notes for list of defects fixed in this release.

10 Known Issues and Limitations

Refer to release notes for list of defects fixed in this release.

11 Backward Compatibility

This software release is backward compatible with previously released versions of the SafeTI™ Hercules™ Diagnostic library.

12 Compatibility with Other Systems

This software is a software API library for using the hardware-diagnostic features available in the Hercules™ Safety MCUs. It does not restrict usage of other software systems such as an RTOS or device drivers (for example, HALCoGen generated device drivers).

13 Software Manifest

You can find the Software Manifest for SafeTI™ diagnostic library in the <Installation Directory>\docs folder of your installation.

14 Change Control, Support, and Maintenance

The change request could be either a feature request, support for existing software or defect fix. All change requests initiated by the integrator either through the TI E2E system or other means shall be routed through the concerned Field Application Engineer. The change request shall be submitted in the change management system documenting the reason for change request. The change control board shall evaluate the results of impact analysis considering the impact on safety and will approve or reject the change request. If the change is approved, a change notice (ECN) is prepared and communicated to all concerned. If the change is rejected, the same is communicated to the originator and all concerned. The changes shall be implemented by the development team as per plan and after the planned verification, the software shall be released. The release notification shall be communicated to all concerned or affected parties.

15 Design Safe State (If Applicable)

This software library is used to test or run the diagnostics available on the Hercules™ Safety MCU. Running these tests can result in flagging of safety critical errors indicating a hardware fault. The system designer must handle the fault flagged by the diagnostic library to achieve a safe state as required.

16 Interface Constraints

Refer to the API user guide included above for any constraints in the use of the API.

17 Competence

The person responsible for integration of this unit into the end-application software must have a good understanding of the safety features and mechanisms available in hardware in the Hercules™ Safety MCU.

18 Justification of Claims

The Software Diagnostics Library provides a software implementation of the diagnostic features recommended for the TI Hercules™ Safety MCU family of devices. The features of the diagnostic library are implemented as described in the device Technical Reference Manual and the device Safety Manual.

19 Software Quality Metrics

Software metrics are a quantitative guide to the performance of the software. Software metrics are the basis for efficient project and quality management. The quality of the software product can be determined with the software metrics. The set of metrics shown in Table 4 is used for the evaluation of software quality. These metrics are a subset of HIS metrics.

Table 4. Software Quality Metrics

Sl. No	Metric	Description	Range
1	Number of instructions per function	Number of executable lines in the function. Indicates complexity of the code	1-50
2	Comment Density	Relationship of the number of comments to the number of statements. Provides information about clarity in the code.	>0.2
3	Number of Goto statements	Number of Goto statements found in the code. Goto statements increase the number of paths in the code. Use of gotos can lead to unreadable and unmaintainable code.	0
4	Cyclomatic Complexity	Measurement of the number of linearly independent paths through the source code. Indicates the complexity of the code	1-10
5	Number of calling functions	Measurement of the number of functions which call this function.	0-5
6	Number of called functions	Measurement of the number of different functions called by this function	0-7
7	Number of function parameters	Determine the complexity of the function interface. Complexity of the function, need for computation of stack allocation	0-5
8	Number of return points	Number of return points within a function. Provides information about complexity and maintainability of the function.	0-1
9	Language Scope	The language scope is an indicator of the cost of maintaining/changing functions. "VOCF" VOCF = (N1+N2)/(n1+n2) where: N1 = Sum of all operators, N2 = Sum of all operands, n1 = Number of different operators, n2 = Number of different operands Higher value indicates similar or duplicate code portions.	1-4
10	Number of recursions	Count the number of recursions. Recursion should generally be avoided because it makes the code less readable and harder to maintain and debug.	0
11	Number of Global variables	Measure of the number of Global variables used	0
12	Number of paths	Number of non cyclic remark paths	1-80

20 Appendix A: MISRA-C Guidelines

20.1 MISRA-C Rules Adhered – Mandatory

Table 5 shows the rules of MISRA-C 2004, which are mandatory and must be followed for all implementations.

Table 5. MISRA-C Rules Adhered

Rule No.	Type	Category
2.1	Required	Language Extensions
2.2	Required	Language Extensions
2.3	Required	Language Extensions
2.4	Advisory	Language Extensions
4.1	Required	Character sets
4.2	Required	Character sets
5.2	Required	Identifiers
5.3	Required	Identifiers
5.4	Required	Identifiers
6.1	Required	Types
6.3	Advisory	Types
6.4	Required	Types
6.5	Required	Types
7.1	Required	Constants
8.1	Required	Declarations and definitions
8.2	Required	Declarations and definitions
8.3	Required	Declarations and definitions
8.4	Required	Declarations and definitions
8.5	Required	Declarations and definitions
8.6	Required	Declarations and definitions
8.7	Required	Declarations and definitions
8.8	Required	Declarations and definitions
8.9	Required	Declarations and definitions
8.11	Required	Declarations and definitions
8.12	Required	Declarations and definitions
9.1	Required	Initialization
9.2	Required	Initialization
9.3	Required	Initialization
10.1	Required	Type conversion
10.3	Required	Type Conversion
10.4	Required	Type Conversion
10.6	Required	Type Conversion
11.1	Required	Pointer type Conversion
11.2	Required	Pointer type Conversion
12.1	Advisory	Expressions
12.2	Required	Expressions
12.3	Required	Expressions
12.4	Required	Expressions
12.5	Required	Expressions
12.7	Required	Expressions
12.8	Required	Expressions
12.9	Required	Expressions

Table 5. MISRA-C Rules Adhered (continued)

Rule No.	Type	Category
12.10	Required	Expressions
12.13	Advisory	Expressions
13.1	Required	Control Statement Expression
13.2	Advisory	Control Statement Expression
13.3	Required	Control Statement Expression
13.4	Required	Control Statement Expression
13.5	Required	Control Statement Expression
13.6	Required	Control Statement Expression
14.1	Required	Control Flow
14.2	Required	Control Flow
14.5	Required	Control Flow
14.8	Required	Control Flow
14.9	Required	Control Flow
14.10	Required	Control Flow
15.1	Required	Switch statement
15.2	Required	Switch statement
15.3	Required	Switch statement
15.4	Required	Switch statement
15.5	Required	Switch statement
16.1	Required	Functions
16.2	Required	Functions
16.3	Required	Functions
16.4	Required	Functions
16.5	Required	Functions
16.8	Required	Functions
16.9	Required	Functions
16.10	Required	Functions
17.2	Required	Pointers and Arrays
17.3	Required	Pointers and Arrays
17.5	Advisory	Pointers and Arrays
17.6	Advisory	Pointers and Arrays
18.1	Required	Structures and Unions
18.2	Required	Structures and Unions
18.4	Required	Structures and Unions
19.1	Advisory	Preprocessor directives
19.2	Advisory	Preprocessor directives
19.3	Required	Preprocessor directives
19.5	Required	Preprocessor directives
19.6	Required	Preprocessor directives
19.8	Advisory	Preprocessor directives
19.9	Required	Preprocessor directives
19.10	Required	Preprocessor directives
19.12	Required	Preprocessor directives
19.13	Advisory	Preprocessor directives
19.14	Advisory	Preprocessor directives
19.15	Required	Standard Libraries
19.16	Required	Preprocessor directives

Table 5. MISRA-C Rules Adhered (continued)

Rule No.	Type	Category
19.17	Required	Preprocessor directives
20.1	Required	Standard Libraries
20.4	Required	Standard Libraries
20.5	Required	Standard Libraries
20.6	Required	Standard Libraries
20.7	Required	Standard Libraries
20.8	Required	Standard Libraries
20.9	Required	Standard Libraries
20.10	Required	Standard Libraries
20.12	Required	Standard Libraries

20.2 MISRA-C Blanket Deviations

Table 6 shows the rules of MISRA-C 2004, which are "Blanket deviations." The source code is not checked for compliance to these rules.

Table 6. MISRA-C Blanket Deviations

Rule No.	Type	Category
1.3	Required	Environment
1.5	Advisory	Environment
3.1	Required	Documentation
3.2	Required	Documentation
3.6	Required	Documentation
5.1	Required	Identifiers
5.6	Advisory	Identifiers
5.7	Advisory	Identifiers
12.11	Advisory	Expressions
18.3	Required	Structures and Unions
19.4	Required	Preprocessor directives
19.11	Required	Preprocessor directives
20.3	Required	Standard Libraries

20.3 MISRA-C Partially Checked Rules

Table 7 shows the rules of MISRA-C 2004, which are partially checked by the tool used for Static Analysis and, hence, the code is not completely checked for compliance to these rules.

Table 7. MISRA-C Partially Checked Rules

Rule No.	Type	Category
1.1	Required	Environment
1.2	Required	Environment
1.4	Required	Environment
3.3	Advisory	Documentation
3.4	Required	Documentation
3.5	Required	Documentation
12.12	Required	Expressions
13.7	Required	Control Statement Expression
16.6	Required	Functions
20.2	Required	Standard Libraries
21.1	Required	Run time failures

20.4 MISRA-C Acceptable Deviations

Table 8 shows the rules of MISRA-C 2004, which are optional (*acceptable deviations*) and are followed as far as is reasonably practical for all implementations. Each instance of the violation from these rules is reviewed and signed off. Violations reported for these rules are reviewed and decided to fix or not on case by case basis. For these situations, if the violation is not fixed, a comment is placed on top of the source code line having the violation.

Table 8. MISRA-C Acceptable Deviations

Rule No.	Type	Category
5.5	Advisory	Identifiers
6.2	Required	Types
8.10	Required	Declarations and definitions
10.2	Required	Type conversion
10.5	Required	Type conversion
11.3	Advisory	Pointer type Conversion
11.4	Advisory	Pointer type Conversion
11.5	Required	Pointer type Conversion
12.6	Advisory	Expressions
14.3	Required	Control Flow
14.4	Required	Control Flow
14.6	Required	Control Flow
14.7	Required	Control Flow
16.7	Advisory	Pointers and Arrays
17.1	Required	Pointer and Arrays
17.4	Required	Pointer and Arrays
19.7	Advisory	Standard Libraries
20.11	Required	Standard Libraries

21 Appendix B: Development Interface Agreement

A Development Interface Agreement (DIA) is intended to capture an agreement between a customer and supplier towards the management of shared responsibilities in developing a functional safety system. In custom developments, the DIA is a key document executed between customer and supplier early in the development process. As the Hercules™ family is a commercial, off the shelf (COTS) product, TI has prepared a standard DIA within this section that describes the support that TI can provide for customer developments. Refer to your local TI sales office for disposition requests for custom DIAs.

21.1 Appointment of Safety Managers

Texas Instruments has developed the Hercules™ processors with one or more development specialist safety managers in place throughout the software development, release to market, and release to production. Safety management after release to production is maintained by separate safety managers who specialize in development and operation issues. Safety management responsibilities are continued through product end-of-life.

21.2 Tailoring of Safety Life Cycle

TI has tailored the safety life cycles of IEC 61508:2010 and ISO 26262:2011 to best match the needs of a safety element out of context (SEooC). The tailoring activity has been executed to meet the requirements in the context of software unit development.

Key elements of the tailored safety life cycle in the context of unit software development are:

- Assumptions on system level design, safety concept, and requirements
- Software Integration Plan is not developed, because the deliverable is only software unit which is the lowest atomic level software component.
- Hardware Software Interface Specification is not developed because the deliverable is only a software unit, which is the lowest atomic level software component and level of interfaces with other units or components is not visible enough.
- Analysis of dependent failures is not conducted because, at the software unit level, it is not feasible to analyze dependent failures. However, SW FMEA shall be conducted as safety analysis to ensure potential failure modes are identified and evaluated.
- Integration testing is not performed because the deliverable is only a software unit, which is the lowest atomic level software component. Because integration test is not performed, software integration test matrix is not delivered.
- Software unit test plan and safety test matrix can be combined into a single document. Similarly, unit test report and safety test report will be combined to a single test report.
- Structural coverage metrics at software architecture level are not collected and analyzed because the deliverable is only a software unit, which is the lowest atomic level software component.

21.3 Activities Performed by TI

The software products covered by this DIA are software units and software components developed as Safety Element out of Context (SEooC). As such, TI's safety activities focus on those related to management of functional safety and software components and software unit developments. System level architecture, design, and safety analysis are not in the scope of TI activities and are the responsibility of the TI customer.

Table 9. Activities Performed by TI versus Performed by SEooC Customer

Safety Life Cycle Activity	TI Execution	SEooC Customer Execution
Management of functional safety	YES	YES
Hazard and risk analysis of end equipments and items	NO	YES
Definition of end equipment and item	NO	YES
Development of end equipment safety concept	Assumptions made	YES
Allocation of end equipment requirements to sub-systems and safety software components/units	Assumptions made	YES

Table 9. Activities Performed by TI versus Performed by SEooC Customer (continued)

Safety Life Cycle Activity	TI Execution	SEooC Customer Execution
Definition of Safety requirements	YES	YES
Software design and development	YES	YES
Software safety analysis	YES	YES
Software verification and testing	YES	YES
Integration of software into end application	Support provided	YES
End equipment level safety analysis	NO	YES
End equipment level verification and validation	NO	YES
End equipment level safety assessment	NO	YES
End equipment release to production	NO	YES
Management of safety issues in production	Support provided	YES

21.4 Information to be Exchanged

In a custom development, there is an expectation under IEC 61508 and ISO 26262 that all development documents related to work products are made available to the customer. In a COTS product, this approach is not sustainable. TI has summarized the most critical development items into a series of documents that can be made available to customers either publicly or under a non-disclosure agreement (NDA). NDAs are required in order to protect proprietary and sensitive information disclosed in certain safety documents. Table 10 summarizes the product safety documentation that TI can provide to customers to assist in development of safety systems.

Table 10. Product Safety Documentation

Deliverable Name	Contents	Confidentiality	Availability
Software Safety Manual	User guide for the safety features of the product, including application level assumptions of use	Public, no NDA required	Available
Software Safety Requirements Specification	Detailed safety requirements for the software units	NDA required	Available
Test results report	Status of the test cycles executed on the software units	NDA required	Available
Code quality reports	Static code analysis and structural coverage metrics	NDA required	Available
Software architecture specification	Architecture of the software units	NDA required	Available
Safety case report	Summary of the conformance of the product to the ISO 26262 and/or IEC 61508 standards	NDA required	Available
Dynamic analysis report	Statement coverage, branch coverage and MC/DC analysis	NDA required	Available

21.5 Parties Responsible for Safety Activities

TI applies a cross functional approach to safety related development. Safety related activities are carried out by a variety of program managers, safety managers, applications engineers, design engineers, and other development engineers

21.6 Supporting Processes and Tools

TI uses a variety of tools during the software development. The tools that are relevant to the safety software development are noted in Table 11. The exact version of the tools used is available in the tool qualification report.

21.6.1 Tools Used for Software Development

Table 11. Software Development Tools

Tool Name	Purpose of the Tool
Rational IBM Clearquest	To maintain the defect database and change requests
ProjectLibre	For project planning and monitoring
Rational IBM DOORS and requirements baseline	Capturing requirements database and maintain requirements traceability
GIT	Version control of source code and artifacts
Code Collaborator	Code/docs review tracking portal
CDDS	Software release repository and tracking the downloads
Jenkins	Software build and release
Code Composer Studio	IDE for code development
Compiler	Compiling the code and generate the executables
LDRA	Static analysis verification, dynamic analysis verification, Test automation and report generation
Metrics Portal	To collect and review project specific metrics for management reviews

21.7 Hazard Analysis and Risk Assessment

Hazard and risk assessments under IEC 61508 and ISO 26262 are targeted at the system level of abstraction. When developing a software unit out of context, the system implementation is not known. Therefore TI has not executed a system hazard and risk analysis. Instead, TI has made assumptions on the results of hazard and risk analysis that are fed into the application development. The ultimate responsibility to determine if the TI software unit/component is suitable for use in the application rests on the software integrator.

21.8 Creation of Functional Safety Concept

The functional safety concept under IEC 61508 and ISO 26262 is targeted at the system level of abstraction. When developing a software unit out of context, the system implementation is not known. Therefore TI cannot generate a system functional safety concept. Instead, TI has made assumptions on the output of a system functional safety concept and this data has been fed into the software unit design. The ultimate responsibility to determine if the TI software unit or component is suitable for use in the application rests on the software integrator.

22 References

- Data Sheet – *RM48L952 16- and 32-Bit RISC Flash Microcontroller* (SPNS177)
- TRM for RM48L952 – *RM48x 16/32-Bit RISC Flash Microcontroller Technical Reference Manual* (SPNU503)
- Safety Manual for RM48L952 – *RM48x 16/32-Bit RISC Flash Microcontroller Technical Reference Manual* (SPNU577)
- Data Sheet – *TMS570LS3137 16- and 32-Bit RISC Flash Microcontroller* (SPNS162)
- TRM for TMS570LS3137 – *TMS570LS31x/21x 16/32-Bit RISC Flash Microcontroller Technical Reference Manual* (SPNU499)
- Safety Manual for TMS570LS3137 *Safety Manual for TMS570LS31x/21x Hercules™ ARM Safety Critical Microcontrollers* (SPNU511)
- HAL Code Generator tool – http://www.ti.com/tool/halcogen
- Uniflash Standalone Flash Tool for TI Microcontrollers – http://www.ti.com/tool/uniflash
- *Initialization of Hercules™ ARM Cortex-R4F Microcontrollers* (SPNA106)
- *TPS65381-Q1 Multi-Rail Power Supply for Microcontrollers in Safety-Critical Applications* (SLVSBC4)
- *Interfacing TPS65381 With Hercules™ Microcontrollers* – http://www.ti.com/lit/an/spna176/spna176.pdf

23 Revision History

Table 12. Revision History

Date	Literature Number	Comments
January 2015	SPNU592	Initial release

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265