

Class B Library Manual for dsPIC33A

dsPIC Functional Safety

Exported on 03/05/2026

Table of Contents

1	Diagnostic Mechanisms	4
1.1	Mechanisms description.....	4
1.2	Mechanisms Categorization	6
2	Mechanisms.....	8
2.1	ADC.....	8
2.1.1	ADC_BOUNDARY_MONITOR_TEST.....	8
2.1.2	ADC_LINEARITY_MONOTONICITY_TEST.....	9
2.1.3	ADC_STARTUP_TEST	10
2.2	CLOCK.....	10
2.2.1	CLOCK_MONITOR_FAULT_INJECTION_TEST.....	11
2.3	CPU	11
2.3.1	CPU_REGISTER_RESET_STATE_TEST.....	12
2.3.2	CPU_CONTROL_REGISTER_TEST.....	13
2.3.3	CPU_SELF_TEST.....	14
2.4	CRC.....	16
2.4.1	CRC_FUNCTIONAL_TEST	16
2.5	FLASH.....	17
2.5.1	FLASH_ECC_SINGLE_DOUBLE_ERROR_DETECTION_TEST.....	17
2.5.2	FLASH_INTEGRITY_READ_PRACTICE	18
2.5.3	FLASH_WRITE_VERIFY_TEST	19
2.5.4	FLASH_MEMORY_CRC_PRACTICE	20
2.6	GPIO	21
2.6.1	GPIO_PORTS_INPUT_PRACTICE	21
2.6.2	GPIO_PORTS_OUTPUT_TEST	22
2.6.3	GPIO_ACTIVITY_CHECK	23
2.6.4	GPIO_PORTS_INTERRUPT_GENERATION_TEST	24
2.6.5	GPIO_PPS_OUTPUT_CONNECTION_TEST	25
2.6.6	IO_MONITOR_TEST	26
2.7	INTERRUPT	27
2.7.1	INTERRUPT_SERVICING_TEST	27
2.7.2	HARD_TRAP_TEST	28

2.7.3	ISR_CLEARED_CHECK.....	29
2.7.4	INTERRUPT_FREQUENCY_CHECK.....	29
2.7.5	EXTERNAL_INTERRUPT_TEST	31
2.8	PC	32
2.8.1	PROGRAM_COUNTER_TEST	32
2.9	SRAM	33
2.9.1	SRAM_ECC_EVENT_INTERRUPT_TEST	33
2.9.2	SRAM_ECC_SINGLE_DOUBLE_ERROR_DETECTION_TEST	34
2.9.3	SRAM_REPLICATION_PRACTICE.....	35
2.10	TIMER.....	36
2.10.1	TIMER_FUNCTIONAL_TEST	36
2.10.2	TIMER_LINEARITY_TEST	37

1 Diagnostic Mechanisms

Information is organized in the form of tables (one for each Diagnostic Mechanism) that collects all the relevant information to allow the System Integrator to understand and use the Diagnostic Mechanism.

1.1 Mechanisms description

The content of the tables is described in Table 1-1:

Table 1-1. Diagnostic Mechanism Information

Field	Description
Purpose	Short explanation of what issue the Diagnostic Mechanism is addressing and how it is implemented.
Description	Additional information to clarify the details of the implementation.
Initialization	Actions needed to make sure the mechanism, as envisioned, works correctly. A list of the required initialization of the involved hardware modules is provided, starting with the module under investigation.
Reporting	Mechanism used to convey the result of the Diagnostic test to the application code calling the Diagnostic routine.
Error	
Diagnostic Type	Describes how the test result is obtained. There are multiple options: *External hardware = The diagnostic is implemented by some additional hardware circuits external to the device *Internal hardware = The diagnostic is implemented by some additional hardware circuits internal to the device *Software = The diagnostic is completely implemented in software with no use of additional internal modules and of external support hardware *Software supported by external hardware = The diagnostic is implemented in software with the support of external hardware *Software supported by internal hardware = The diagnostic is implemented in software with the support of additional internal modules *Software supported by external and internal hardware = The diagnostic is implemented in software with the support of additional internal modules and the support of external hardware
Recommended Mitigation Action	Possible action to be performed to reduce/eliminate the impact of the Failure addressed by the Diagnostic Mechanism.

Field	Description
Periodicity	Describes when the routine should be executed. In general, some of the routines are 'destructive' in the sense that they corrupt the content of the memory and/or registers; others are not. Four possible options are considered in this document: *At start-up: This is the case when running the Diagnostics corrupts the content of the memory/ registers, or the time required to run the Diagnostics is so long that it cannot likely be run during the normal operation of the device when executing the System Integrator's application code. *On-demand: This is typically not a destructive mechanism, allowing the system continuous operation. The Diagnostic Measure is run every time the device feature is used. *Periodic: This is typically not a destructive mechanism, allowing the system continuous operation. The System Integrator calls the Diagnostic routine at specific instants in time based on the application. This option allows the System Integrator to run Diagnostics that require relevant execution times at reasonable instants that do not impact the behavior and performances of the application. *Continuous: This is typically not a destructive mechanism, allowing the system continuous operation. Some Diagnostics, by their nature, can be/are run continuously without impacting the System Integrator's application behavior or performances. Note: Please note that the table provides suggestions, but the final decision of the periodicity is decided by the System Integrator.
Recommendations and Limitations	Notes and information useful to have a correct use of the Mechanism and a correct understanding of its limitations (in use and coverage) as needed.
Relevant Software Requirements	Name of any specific software required by the Diagnostic Mechanism. All these requirements are also listed in Software Requirements on System Level.
Relevant Hardware Requirements	Name assigned to any external hardware (for example, connections between device pins) needed to implement the Diagnostic Mechanism. Hardware Requirements on System Level lists all these items with a detailed description.
Dataflow Dependency	Highlights if the Diagnostic tests operate on ad hoc data or use the data that are normally managed by the application while running. There are two options: *Dependent: The tests and their results depend on the data being operated upon the device during the application code execution. *Independent: The tests and their results depend on data that are specified by the tests themselves. Usually, this is the case for Diagnostics run at start-up (see Periodicity above).

1.2 Mechanisms Categorization

All mechanisms presented in this manual can be assigned to one of the following categories. To ease the categorization, the name of the mechanism includes, as a suffix, the name of the category itself:

Table 1-2. Mechanisms Categorization Details

Category	Comment
CHECK	The mechanism verifies the correct operation of the module, with minimal intervention, to set the operating environment (hardware and software) so that the System Integrator's application code flow is not affected. The mechanism is 'dataflow-dependent' in the meaning described by the FSM (they use the data currently used by the application software) or the mechanism does not interfere with the normal operation of the System Integrator's application code: it runs transparently to it.
MONITOR	The device module includes some additional hardware dedicated to detecting a Failure (incorrect/unexpected behavior of the module). Refer to the device data sheet to operate the module. The System Integrator may (if needed) change the behavior configuration to adapt to the application requirements. The System Integrator shall enable the corresponding interrupt source, manage the interrupt flag and generate the code required by his application to cope with the detected Failure. A MONITOR safety mechanism shall have a diagnostic that
PRACTICE	The mechanism consists of some kind of redundancy (either hardware or software) that requires a direct intervention of the System Integrator's application software to be implemented and operational. It also requires a comparison of the behavior of the two redundant entities and the capability of making the corresponding decision (valid/invalid result).
TEST	The mechanism verifies the correct operation of the module and it requires predefined operating conditions (such as, for instance, a predefined input set to the module). The mechanism is 'dataflow-independent' in the meaning described by the FSM (they require a predefined input set/ sequence to be generated by the application software).

Table 1-3 shows the correlation between the category, the periodicity and the dataflow dependency.

Table 1-3. Category, Periodicity and Dataflow Dependency

Category	Periodicity (1)	Dataflow Dependency (2)
CHECK	On-demand	Dependent
MONITOR	Continuous or on-demand	Dependent
PRACTICE	At start-up, on-demand or periodic	Dependent

Category	Periodicity (1)	Dataflow Dependency (2)
TEST	At start-up or on-demand or periodic	Independent

Notes:

1. Periodicity may also depend on the specific implementation of a diagnostic. Refer to the section 'Diagnostic Mechanisms' for details.
2. Dataflow Dependency may also depend on the specific implementation of a diagnostic. Refer to the section 'Diagnostic Mechanisms' for details.

It is a System Integrator's responsibility to analyze, select and include in the application code the diagnostics proposed in this manual to address single point faults and multiple point faults to increase the robustness of the application. If needed the System application shall define and implement additional diagnostics.

2 Mechanisms

2.1 ADC

2.1.1 ADC_BOUNDARY_MONITOR_TEST

Field	Value
Purpose	Detect Failures in the circuits dedicated to ADC boundary monitoring (see ADC_BOUNDARY_MONITOR)
Description	The Low and High Comparator threshold registers are loaded with predefined values. One of the internal Current Bias Generators is externally connected to a capacitor. The generated voltage ramp is sampled (and the value converted by the ADC) at times when the converted value is outside the interval values set in the comparator registers.
Initialization	ADC: as required by the System Integrator's application; ADC comparator: enable and load registers with threshold values
Additional Required Resources	None
Error Reporting	A software flag is set indicating the conversion result is outside the predefined range
Diagnostic Type	Software supported by external and internal hardware
Recommended Mitigation Action	Application dependent
Periodicity	On-demand
Recommendations and Limitations	None
Relevant Software Requirements	SW_ADC_BOUNDARY_MONITOR_TEST_01
Relevant Hardware Requirements	HW_CBG_OUTPUT_ADC_INPUT_CAPACITOR
Dataflow Dependency	Dependent

2.1.2 ADC_LINEARITY_MONOTONICITY_TEST

Field	Value
Purpose	Detect Failures in the ADC impacting its monotonicity and linearity
Description	Use the internal DAC, externally connecting its output (DACOUT1) to one of the ADC channels; use multiple output levels to run the test.
Initialization	ADC: as required by the System Integrator's application; DAC: enable the module; one analog input pin; one analog output pin
Additional Required Resources	Internal DAC; analog output pin
Error Reporting	A software flag is set indicating the detection of a nonlinear behavior
Diagnostic Type	Software supported by external and internal hardware
Recommended Mitigation Action	Application dependent
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_ADC_LINEARITY_MONOTONICITY_TEST_01
Relevant Hardware Requirements	HW_DAC_OUTPUT_ADC_TEST
Dataflow Dependency	Independent

2.1.3 ADC_STARTUP_TEST

Field	Value
Purpose	Detect Failures in the overall ADC module
Description	Provide an external DC voltage to the input of an ADC channel. Enable the ADC, provide the trigger and make sure a non-zero value is read in the corresponding ADC buffer. Repeat the test for each supported data format (6/8/10/12-bit resolution, integer/fractional) to test the involved control registers.
Initialization	ADC: as required by the System Integrator's application
Additional Required Resources	Analog input pins; external equipment (DC voltage source); external passive components (resistors)
Error Reporting	A software flag is set indicating a zero conversion result is obtained
Diagnostic Type	Software supported by external hardware
Recommended Mitigation Action	Application dependent
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_ADC_STARTUP_TEST_01
Relevant Hardware Requirements	HW_EXTERNAL_VOLTAGE_ADC_CHANNEL
Dataflow Dependency	Independent

2.2 CLOCK

2.2.1 CLOCK_MONITOR_FAULT_INJECTION_TEST

Field	Value
Purpose	Detect failures in the following circuits to verify the capability of detecting these events. • Artificial catastrophic fault injected by blanking the monitored clock. • High frequency drift fault injected by halving the reference clock. • Low frequency drift fault injected by halving the monitored clock
Description	The faults above can be selectively injected to verify the capability of the element to detect these events.
Initialization	None
Additional Required Resources	None
Error Reporting	A hardware flag is set and an interrupts is generated if enabled.
Diagnostic Type	Software supported by internal hardware
Recommended Mitigation Action	Application dependent
Periodicity	At start-up or on-demand or periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_CLOCK_MONITOR_FAULT_INJECTION_TEST_01
Relevant Hardware Requirements	None
Dataflow Dependency	Independent

2.3 CPU

2.3.1 CPU_REGISTER_RESET_STATE_TEST

Field	Value
Purpose	Detect Failures in the CPU module impacting the status of its registers at start-up
Description	This Diagnostic shall be executed, prior to any other Diagnostic at start-up, in order to avoid incorrect operation of other Diagnostics or application initialization code due to unexpected initial register values. No interrupts shall be enabled before executing this Diagnostic function. Some registers (W0–W15, DSRPAG, TBLPAG, PCL and PCH) and some specific control/status bits (DC, N, OV, SZ, C and SFA) are not tested in this Diagnostic Mechanism. Since it is not practically possible to check the Reset state of the Program Counter, the Diagnostic checks that the SPLIM register has been correctly initialized to its expected value: <code>__SPLIM_init</code> . This would serve as evidence that the C run-time start-up routine had executed correctly upon Reset, which would generally not be possible if the Program Counter had an incorrect Reset state.
Initialization	None
Additional Required Resources	None
Error Reporting	A software flag is set indicating an incorrect initial register value has been detected. The number of failing registers is saved.
Diagnostic Type	Software
Recommended Mitigation Action	Write the correct default values to the failing registers
Periodicity	At start-up
Recommendations and Limitations	None
Relevant Software Requirements	SW_CPU_REGISTER_RESET_STATE_TEST_01
Relevant Hardware Requirements	None
Dataflow Dependency	Independent

2.3.2 CPU_CONTROL_REGISTER_TEST

Field	Value
Purpose	Detect Failures in the CPU module affecting the integrity of its control registers
Description	This diagnostic performs a run-time write/read check of all CPU control and status registers, including W0-W15. It helps detect stuck-at faults affecting individual bits or registers. It helps detect any register access/writability issues.
Initialization	None
Additional Required Resources	None
Error Reporting	A software flag is set indicating a mismatch has been detected. Individual software flags are set indicating the failing registers.
Diagnostic Type	Software
Recommended Mitigation Action	Restore the correct values of the failing registers based on their stored values
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	Interrupts shall be temporarily disabled in order to avoid corruption of the registers being tested as a result of an ISR execution. Some registers (PCL, PCH, DSRPAG, DSWPAG, YPAG, MSTRPR and CTXTSTAT) and some specific control/status bits (OA, OB, OAB, DA, DC, RA, N, OV, Z, C, EDT, DL[2:0], IPL[3], SFA and BREN) are not tested in this Diagnostic Mechanism, because writing to these bits would potentially affect normal code execution (thereby causing unpredictable application behavior). The DOSTARTL and DOSTARTH registers are not tested as they are read-only (cannot be written).
Relevant Software Requirements	SW_CPU_CONTROL_REGISTER_TEST_01
Relevant Hardware Requirements	None
Dataflow Dependency	Independent

2.3.3 CPU_SELF_TEST

Field	Value
Purpose	Detect Failures in the CPU module impacting the functional correctness of all core instructions and features
Description	<p>Each test subset generates a unique numerical test result that is compared to the corresponding expected value. CPU self-test library functions shall be invoked periodically to verify that all CPU core features are still functioning correctly. The CPU self-test function calls generate a set of pass/fail results that can be sent as a heartbeat message through any communication interface. Alternatively, only the computed results of the various functions can be sent, with an external device analyzing the results and resetting the MCU if needed. The self-test algorithm is designed to provide at least 95% test coverage of overall CPU functionality, which includes all available instruction opcodes, addressing modes, bus structures, bits of CPU registers, math hardware and any special processor features that can impact data computation. The overall test suite is subdivided into eight test subset functions (each representing a different subset of CPU functionality), which are executed in a cyclic manner. For example, a different test subset function may be called by the application every 2 ms, thereby completing the entire test suite every 16 ms; since each test subset generates a unique 16-bit data result, eight such test results are generated during the 16 ms window. Each test subset function performs a series of computations on an original “seed” value, generating a unique 16-bit result at the end of the function execution. Interrupts shall be disabled during the execution of a single test subset function, so each test subset is designed to execute in less than 150 instruction cycles. Functional items covered by each test subset are listed below. Subset 1: All of the MOV instructions are tested. Addressing modes tested: Immediate, File Register, Register Direct and Indirect (with Pre-Increment, Post-Increment, Pre-Decrement, Post-Decrement, Literal Offset and Register Offset). SWAP and EXCH instructions are tested. All bit manipulation, bit test and bit-compare-skip operations are tested. Single-word and double-word instructions are tested. Byte MOV instructions are also checked. Subset 2: PSV and table accesses from different sections (four locations) of program memory. All bits of the program memory address bus are toggled. All bits of the data memory address bus are toggled. All bits of the X data read and write buses are toggled. CPU registers tested for read/write operations are: W0-W15, SPLIM, TBLPAG, PSVPAG/DSRPAG. Read-After-Write (RAW) dependency. NOP and NOPR instructions. Subset 3: All conditional branch and GOTO instructions with alternative conditions. Program Counter (PC) behavior during above program flow change operations. All CALL and RETURN operations. Automatic context save on stack. All stack and shadow operations. DISI instruction. Subset 4: Branch instructions tested for false condition are NOV, Z, NN, NC and NZ. All logic instructions: AND, CLR, COM, IOR, NEG, SETM and XOR instructions. All data rotate and shift instructions. All compare and compare-skip instructions. The Z, OV, DC, N and C bits of the STATUS Register are checked for their behavior. Some Byte mode logic</p>

Field	Value
	instructions are tested. Subset 5: The following arithmetic instructions are tested: ADD, ADDC, SUB, SUBB, SUBBR, INC, INC2, DEC, DEC2, SE, ZE. The addressing modes tested here are: Immediate, File Register, Register Direct. Byte instructions are also checked. Branch instructions tested for True condition are: GT, GTU, LE, LEU, NC, NN, OV. Branch instructions tested for False condition are: C, GE, GEU, LT, LTU, N, NOV. Divide Unsigned Double (DIV.UD) instruction. Subset 6: All MUL instruction variants. All DIV instruction variants, except DIVF and DIV.UD. REPEAT loop. Math error trap generation (divide-by-zero error) Subset 7: All DSP accumulator operations, including different bit states of the individual bits of both accumulators. All DSP multiplier-based instructions. All DSP MAC Register Indirect Addressing modes. All DSP shift instructions. Math error trap generation due to accumulator-related events (accumulator overflow and catastrophic overflow). CORCON bit behavior. Subset 8: Modulo Addressing (both Byte and Word modes). Bit-Reversed Addressing. DO loop. Fractional Divide (DIVF) instruction.
Initialization	Disable interrupts completely during the execution of each test subset function
Additional Required Resources	None
Error Reporting	A software flag is set indicating a mismatch in the generated and expected result
Diagnostic Type	Software
Recommended Mitigation Action	Application dependent
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	<p>Interrupts are temporarily disabled during each execution of this Diagnostic function in order to avoid corruption of the registers being tested as a result of an ISR execution. Before returning from each Diagnostic function call, the original interrupt settings are restored, allowing resumption of interrupt processing if it was previously enabled. The System Integrator application shall not have its own Math Error Trap handler routine active during the execution of this Diagnostic function, since the CPU self-test algorithm utilizes its own Math Error Trap to test the correct functioning of various sources of this trap. This Diagnostic Mechanism only detects a Fault in the specific set of CPU features tested in a specific test subset function (as listed in the Purpose/Description above), but does not distinguish between the different possible Faults within that test subset. Thus, it is not possible to detect exactly which feature produced a Fault.</p>

Field	Value
Relevant Software Requirements	SW_CPU_SELF_TEST_01
Relevant Hardware Requirements	None
Dataflow Dependency	Independent

2.4 CRC

2.4.1 CRC_FUNCTIONAL_TEST

Field	Value
Purpose	Detect Failures in the CRC module affecting the overall operation.
Description	This test includes two sub-tests; the System Integrator can select either one: Test of the correct selection of the needed polynomial; fill the FIFO with a known sequence, run the peripheral and collect the output. Check the output, comparing it to the predefined known expected sequence. Check the correct CRC hardware computation, performing the CRC in software on the FIFO content and compare outputs.
Initialization	CRC: as required by the System Integrator's application
Additional Required Resources	None
Error Reporting	A software flag is set indicating: An incorrect selection of the polynomial; or An incorrect CRC result has been generated
Diagnostic Type	Software
Recommended Mitigation Action	Application dependent

Field	Value
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_CRC_FUNCTIONAL_TEST_01
Relevant Hardware Requirements	None
Dataflow Dependency	Independent

2.5 FLASH

2.5.1 FLASH_ECC_SINGLE_DOUBLE_ERROR_DETECTION_TEST

Field	Value
Purpose	Detect Failures in the ECC Flash submodules: verify the functionality of the hardware detecting a single or a double-bit error (see FLASH_ECC_SINGLE_ERROR_MONITOR and FLASH_ECC_DOUBLE_ERROR_MONITOR)
Description	The ECC engine provides the possibility to use the fault handling by forcing the generation of an ECC error. Both single and double-bit errors can be generated in both the read and write data paths. Read path Fault injection first reads the Flash data and then modifies them prior to entering the ECC logic. Write path Fault injection modifies the actual data prior to them being written into the target Flash and will cause an ECC error on a subsequent Flash read. The System Integrator shall refer to the device datasheet for a detailed description of the required actions and software implementation.
Initialization	None
Additional Required Resources	None

Field	Value
Error Reporting	A hardware flag (interrupt flag) is set indicating a single bit in error is detected; if enabled, the interrupt is serviced. A generic trap is executed indicating a double-bit error has been detected.
Diagnostic Type	Software supported by internal hardware
Recommended Mitigation Action	Application dependent
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_FLASH_ECC_SINGLE_DOUBLE_ERROR_DETECTION_TEST_01
Relevant Hardware Requirements	None
Dataflow Dependency	Independent

2.5.2 FLASH_INTEGRITY_READ_PRACTICE

Field	Value
Purpose	Aimed to verify the integrity of Flash blocks that are not normally executed by the System Integrator's code during the safety interval
Description	This diagnostic requires the System Integrator's code to access (read) parts of Flash that contain safety code that is not executed during the safety intervals. The ECC engine, acting in the background while a read is executed, allows the System Integrator to verify the status of the Flash and to detect single and double errors.
Initialization	None
Additional Required Resources	None

Field	Value
Error Reporting	In case the ECC monitor detects a single-bit error: a hardware flag (interrupt flag, ECCSBEIF) is set indicating a single-bit error has been detected and corrected; the System Integrator shall enable the interrupt generation to prevent the fault from being latent. In case the ECC monitor detects a double-bit error: a generic hard trap is generated.
Diagnostic Type	Software supported by internal hardware
Recommended Mitigation Action	Application dependent
Periodicity	On-demand or periodic
Recommendations and Limitations	This diagnostic shall be used to check Flash blocks not normally executed by the System Integrator's code.
Relevant Software Requirements	SW_FLASH_INTEGRITY_READ_PRACTICE_01
Relevant Hardware Requirements	None
Dataflow Dependency	Dependent

2.5.3 FLASH_WRITE_VERIFY_TEST

Field	Value
Purpose	Detect Failures in the Flash affecting its access performances
Description	The Flash locations of interest are read and compared with the predefined value. The two values are checked to make sure they match.
Initialization	None
Additional Required Resources	None
Error Reporting	A software flag is set indicating the detection of a mismatch between the read Flash content and the predefined value

Field	Value
Diagnostic Type	Software
Recommended Mitigation Action	Application dependent
Periodicity	On-demand
Recommendations and Limitations	None
Relevant Software Requirements	SW_FLASH_WRITE_VERIFY_TEST_01
Relevant Hardware Requirements	None
Dataflow Dependency	Independent

2.5.4 FLASH_MEMORY_CRC_PRACTICE

Field	Value
Purpose	Aimed to increase the robustness of the Flash memory by means of periodic CRC computation on the entire Flash memory
Description	Use a periodic read of the entire Flash memory (or sections based on bandwidth needs) and compute the CRC of the data from Flash_Start_Address to Flash_End_Address using a standard polynomial, such as CCITT-16. Compare the computed CRC with the previous value stored in memory.
Initialization	None
Additional Required Resources	None
Error Reporting	A software flag is set indicating a non-matching CRC has been computed
Diagnostic Type	Software
Recommended Mitigation Action	Application dependent

Field	Value
Periodicity	At start-up to detect multi-bit Failures (since the probability of these events is low, choosing to execute the diagnostic at start-up does not significantly impact the safety performances of the device); on-demand or periodically to mainly address Flash address decoder Failures
Recommendations and Limitations	None
Relevant Software Requirements	SW_FLASH_MEMORY_CRC_PRACTICE_01
Relevant Hardware Requirements	None
Dataflow Dependency	Dependent

2.6 GPIO

2.6.1 GPIO_PORTS_INPUT_PRACTICE

Field	Value
Purpose	Aimed to minimize the impact of Failures in general purpose input ports
Description	Route the same input to two distinct pins through external connections. After every read of an I/O port pin, compare the logic state of the main input pin with the logic state of the secondary input pin. This diagnostic is effective in detecting signal propagation Failure from the physical pin to the I/O pad, and stuck-at Faults or soft errors in the PORTx register.
Initialization	GPIO: as required by the System Integrator's application
Additional Required Resources	PPS: one GPIO pin (input)

Field	Value
Error Reporting	A software flag is set indicating the comparison between the input under consideration and the secondary input failed
Diagnostic Type	Software supported by external and internal hardware
Recommended Mitigation Action	Application dependent
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_GPIO_PORTS_INPUT_PRACTICE_01
Relevant Hardware Requirements	HW_IO_INPUT_SECONDARY_IO_INPUT
Dataflow Dependency	Dependent

2.6.2 GPIO_PORTS_OUTPUT_TEST

Field	Value
Purpose	Detect Failures in GPIOs impacting the output path
Description	Route an output pin back into the device on a secondary pin through external connections. After every write to an I/O port pin, perform both of the following tests: (1) XOR the values of the LATx and PORTx bits corresponding to the main output pin; (2) compare the secondary input pin with what the main output pin was supposed to drive. This diagnostic is effective in detecting signal propagation Failure from the I/O pad to the physical pin, as well as detecting Stuck-at Faults or soft errors in the LATx and PORTx registers.
Initialization	GPIO pin (output): as required by the System Integrator's application; GPIO pin (input)
Additional Required Resources	PPS: one GPIO pin (input)

Field	Value
Error Reporting	A software flag is set indicating one of the following: the XOR of the LATx and PORTx bits yielded a non-zero value; or the comparison between the tested output and the secondary input states failed.
Diagnostic Type	Software supported by external and internal hardware
Recommended Mitigation Action	Application dependent
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_GPIO_PORTS_OUTPUT_TEST_01
Relevant Hardware Requirements	HW_IO_OUTPUT_IO_INPUT
Dataflow Dependency	Independent

2.6.3 GPIO_ACTIVITY_CHECK

Field	Value
Description	This diagnostic is intended to verify: (1) the module driving the pin is working correctly; (2) the pin functionality.
Initialization	GPIO pin (output): as required by the System Integrator's application; GPIO pin (input)
Additional Required Resources	PPS: one GPIO pin (input)
Error Reporting	A software flag is set indicating transitions on the pin have not been detected
Diagnostic Type	Software supported by external and internal hardware
Recommended Mitigation Action	Application dependent

Field	Value
Periodicity	On-demand
Recommendations and Limitations	None
Relevant Software Requirements	SW_GPIO_ACTIVITY_CHECK_01
Relevant Hardware Requirements	HW_IO_OUTPUT_IO_INPUT
Dataflow Dependency	Dependent

2.6.4 GPIO_PORTS_INTERRUPT_GENERATION_TEST

Field	Value
Purpose	Detect Failures in GPIOs affecting the capability to generate interrupts on input transitions
Description	Route a GP pin (output) to the input pins of interest; toggle the output pin and verify the generation of interrupts.
Initialization	GPIO pin (output): as required by the System Integrator's application; GPIO pin (input)
Additional Required Resources	PPS: one GPIO pin (input)
Error Reporting	A software flag is set indicating an interrupt has not been generated as the input pin changes
Diagnostic Type	Software supported by external and internal hardware
Recommended Mitigation Action	Application dependent
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_GPIO_PORTS_INTERRUPT_GENERATION_TEST_01

Field	Value
Relevant Hardware Requirements	HW_IO_OUTPUT_IO_INPUT
Dataflow Dependency	Independent

2.6.5 GPIO_PPS_OUTPUT_CONNECTION_TEST

Field	Value
Purpose	Detect Failure in the hardware impacting the correct connection of a peripheral output to a predefined pin
Description	Verify the internal mux that connects the peripheral of interest to an output general purpose pin
Initialization	GPIO pin (output): as required by the System Integrator's application; GPIO pin (input); Internal peripheral: as required by the System Integrator's application; CLC module (PPS virtual pin): enable the module to detect input activity
Additional Required Resources	PPS: one GPIO pin (input); PPS (virtual pins): one CLC input
Error Reporting	A software flag is set indicating the absence of any transition on the pin under investigation, or an unexpected sequence of status of the output pin under investigation
Diagnostic Type	Software supported by external hardware
Recommended Mitigation Action	Application dependent
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_GPIO_PPS_OUTPUT_CONNECTION_TEST_01
Relevant Hardware Requirements	HW_PERIPHERAL_OUTPUT_IO_INPUT

Field	Value
Dataflow Dependency	Dependent

2.6.6 IO_MONITOR_TEST

Field	Value
Purpose	Detect Failures in the circuit affecting the capability to detect failures in the output pins and external circuits
Description	The module fault injection capability is used: bit FLTINJ in the IO MONITOR configuration register allows emulation of a comparison mismatch event.
Initialization	None
Additional Required Resources	None
Error Reporting	The ERR bit is set; the ERRCNT register is incremented to EOVM; the OVF flag is set if the ERRCNT overflows.
Diagnostic Type	Software supported by internal hardware
Recommended Mitigation Action	Application dependent
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_IO_MONITOR_TEST_01
Relevant Hardware Requirements	None
Dataflow Dependency	Independent

2.7 INTERRUPT

2.7.1 INTERRUPT_SERVICING_TEST

Field	Value
Purpose	Detect Failures in the interrupt controller impacting the capability to generate interrupts
Description	Set the IF (Interrupt Flag) bits of every individual vector source used in the end application, one at a time, and check that the corresponding Interrupt Service Routine for each interrupt is serviced. Inside each ISR, clear the interrupt flag, and set a software flag indicating that the individual interrupt was tested and worked correctly. After the test sequence, the interrupts may be disabled. This diagnostic tests for Stuck-at Faults in the interrupt flags, as well as Faults in the interrupt vectoring logic.
Initialization	Interrupt controller: as required by the System Integrator's application
Additional Required Resources	None
Error Reporting	A software flag is set indicating the particular interrupt service worked correctly
Diagnostic Type	Software
Recommended Mitigation Action	Application dependent
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_INTERRUPT_SERVICING_TEST_01
Relevant Hardware Requirements	None
Dataflow Dependency	Independent

2.7.2 HARD_TRAP_TEST

Field	Value
Purpose	Detect Failures in the interrupt controller module preventing traps from being generated
Description	This Diagnostic tests for Stuck-at Faults in the software-generated hard trap flag, as well as Faults in the trap vectoring logic. Test for hard trap routine execution: set the SWTRAP bit, which in turn sets the SGHT bit. Check whether it causes the code execution to enter the generic hard trap handler or not.
Initialization	None
Additional Required Resources	None
Error Reporting	A software flag is set indicating a trap has not been generated
Diagnostic Type	Software
Recommended Mitigation Action	Application dependent
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_HARD_TRAP_TEST_01
Relevant Hardware Requirements	None
Dataflow Dependency	Independent

2.7.3 ISR_CLEARED_CHECK

Field	Value
Purpose	Detect Failures in the interrupt controller affecting the capability of servicing interrupts
Description	This diagnostic is included at the end of each ISR. It checks that the interrupt flag has been cleared once at the exit of the Interrupt Service Routine. This code shall be inserted into all ISRs.
Initialization	None
Additional Required Resources	None
Error Reporting	A software flag is set indicating an interrupt flag has not been cleared
Diagnostic Type	Software
Recommended Mitigation Action	Application dependent
Periodicity	On-demand
Recommendations and Limitations	None
Relevant Software Requirements	SW_ISR_CLEARED_CHECK_01
Relevant Hardware Requirements	None
Dataflow Dependency	Dependent

2.7.4 INTERRUPT_FREQUENCY_CHECK

Field	Value
Purpose	Detect Failures in the interrupt controller impacting the capability of generating multiple interrupts in a predefined time interval

Field	Value
Description	The interrupt routine can be invoked at specified time intervals; it is triggered by a timer to monitor and verify the interrupt operation. To keep track of the interrupts that occur frequently, a dedicated counter in each ISR is decremented when an interrupt occurs. For example, if the SPI is configured to generate an interrupt every 2 ms, the SPI generates at least five interrupts in 10 ms. When an SPI interrupt occurs, the counter dedicated to tracking the SPI interrupt is decremented. Thus, if the counter is initialized to five, it is decremented to zero in 10 ms, which is verified by the interrupt test function triggered every 10 ms. To keep track of interrupts that occur rarely, a dedicated counter within the interrupt test function is decremented if the specific interrupt did not occur during the last time interval.
Initialization	Interrupt controller: as required by the System Integrator's application
Additional Required Resources	None
Error Reporting	The counters used to monitor the occurrences of each individual interrupt used by the application can be inspected to determine if the interrupt is occurring at the correct frequency
Diagnostic Type	Software
Recommended Mitigation Action	Application dependent
Periodicity	On-demand
Recommendations and Limitations	None
Relevant Software Requirements	SW_INTERRUPT_FREQUENCY_CHECK_01
Relevant Hardware Requirements	None
Dataflow Dependency	Dependent

2.7.5 EXTERNAL_INTERRUPT_TEST

Field	Value
Purpose	Detect Failures in the interrupt controller preventing external interrupts to be serviced
Description	This Diagnostic Mechanism tests for Faults in the external pin edge detection mechanism and the I/O pins on which the external interrupts are present. Create the input stimuli to trigger each of the external interrupts (INTx) present on the device by:(1)Externally connecting and driving I/O pins;(2)Self-driving the pin itself (in this case, an external resistor for isolation is strongly suggested).Verify whether the interrupt occurs properly or not, by individually checking and then clearing each INTxIF interrupt flag.
Initialization	Interrupt Controller: as required by the System Integrator's application; GPIO pins (output); OC/MCCP/SCCP: enable and program to emulate interrupt request to the interrupt controller
Additional Required Resources	PPS: one GPIO pin (output); PPS (virtual pins): One Output Compare (OC/MCCP/SCCP) module
Error Reporting	A software flag is set indicating the external interrupt has not been serviced.
Diagnostic Type	Software supported by external and internal hardware
Recommended Mitigation Action	The System Integrator's application can switch from a failing external interrupt pin to a passing interrupt, if possible.
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_EXTERNAL_INTERRUPT_TEST_01
Relevant Hardware Requirements	HW_IO_OUTPUT_EXTERNAL_INTERRUPT_INPUT
Dataflow Dependency	Independent

2.8 PC

2.8.1 PROGRAM_COUNTER_TEST

Field	Value
Purpose	To check if the program counter flow is as expected
Description	The API repeatedly calls three internal helper functions to indicate that those functions have been touched, a counter variable is updated and the updated value is verified at the end of the API for the correct value to indicate a PASS or FAIL
Initialization	None
Additional Required Resources	None
Error Reporting	The diagnostic returns a software code indicating: pass (all the three internal helper functions are called); or fail (atleast one internal helper function is not called).
Diagnostic Type	Software
Recommended Mitigation Action	Application dependent
Periodicity	Periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_PROGRAM_COUNTER_TEST
Relevant Hardware Requirements	None
Dataflow Dependency	Independent

2.9 SRAM

2.9.1 SRAM_ECC_EVENT_INTERRUPT_TEST

Field	Value
Purpose	Detect Failures in the module preventing the capability to generate interrupts connected to ECC events
Description	Fault injection is used to artificially generate an ECC event, and the corresponding interrupt flag is monitored.
Initialization	None
Additional Required Resources	None
Error Reporting	A software flag is set in the event of a missed interrupt
Diagnostic Type	Software supported by internal hardware
Recommended Mitigation Action	Application dependent
Periodicity	At start-up or on-demand
Recommendations and Limitations	None
Relevant Software Requirements	SW_SRAM_ECC_EVENT_INTERRUPT_TEST_01
Relevant Hardware Requirements	None
Dataflow Dependency	Independent

2.9.2 SRAM_ECC_SINGLE_DOUBLE_ERROR_DETECTION_TEST

Field	Value
Purpose	Detect Failures in the ECC SRAM submodules: verify the functionality of the hardware detecting a single or a double-bit error (see SRAM_ECC_SINGLE_ERROR_MONITOR and SRAM_ECC_DOUBLE_ERROR_MONITOR)
Description	The ECC engine provides the possibility to use the fault handling by forcing the generation of an ECC error. Both single and double-bit errors can be generated in both the read and write data paths. Read path Fault injection first reads the SRAM data and then modifies them prior to entering the ECC logic. Write path Fault injection modifies the actual data prior to them being written into the target SRAM and will cause an ECC error on a subsequent SRAM read. The System Integrator shall refer to the device datasheet for a detailed description of the required actions and software implementation.
Initialization	None
Additional Required Resources	None
Error Reporting	A hardware flag (interrupt flag) is set indicating a single bit in error is detected; if enabled, the interrupt is serviced. A generic trap is executed indicating a double-bit error has been detected.
Diagnostic Type	Software supported by internal hardware
Recommended Mitigation Action	Application dependent
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_SRAM_ECC_SINGLE_DOUBLE_ERROR_DETECTION_TEST_01
Relevant Hardware Requirements	None
Dataflow Dependency	Independent

2.9.3 SRAM_REPLICATION_PRACTICE

Field	Value
Purpose	Aimed to minimize the impact of Failures in the SRAM module by means of patterns and their complements
Description	Applicable for safety-related and key variables. Each value is stored with its inverted value in SRAM. During every variable read, read its value and XOR it with its inverted version; any result other than 0xFFFF (all ones) means variable corruption. Provides very high Diagnostic Coverage, even for transient and soft errors. High execution time overhead for each variable access, but overall impact may be moderate as variables only need to be tested when they are used. Implicitly tests Byte, Word and Double-Word Addressing, depending on the data types used.
Initialization	None
Additional Required Resources	None
Error Reporting	A software error code is set indicating a mismatch between the two data values is detected
Diagnostic Type	Software
Recommended Mitigation Action	Application dependent
Periodicity	On-demand
Recommendations and Limitations	None
Relevant Software Requirements	SW_SRAM_REPLICATION_PRACTICE_01
Relevant Hardware Requirements	None
Dataflow Dependency	Dependent

2.10 TIMER

2.10.1 TIMER_FUNCTIONAL_TEST

Field	Value
Purpose	Detect Failures in the Timer module affecting its performance
Description	To run this diagnostic all interrupts shall be disabled. A fixed set of instructions are executed on the CPU, and then compared with the timer count at the end of the execution. Report if the count is within the tolerance value, above or below
Initialization	Timer: as required by the System Integrator's application
Additional Required Resources	None
Error Reporting	The diagnostic returns a software code indicating: the timer count is within the tolerance band; the timer count is less than the tolerance band (timer running too slow); or the timer count is greater than the tolerance band (timer running too fast).
Diagnostic Type	Software supported by internal hardware
Recommended Mitigation Action	Application dependent
Periodicity	At start-up or on-demand, or periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_TIMER_FUNCTIONAL_TEST_02
Relevant Hardware Requirements	None
Dataflow Dependency	Independent

2.10.2 **TIMER_LINEARITY_TEST**

Field	Value
Purpose	Detect Failures in the element impacting the linearity of the timer functionality
Description	Use the hardware timer to check for the difference between the timer values at different instances of time. For the first time, the TMR value is captured before and after a set number of CPU cycles of instructions is executed, which constitutes an initial call. Subsequently, when the API is called again, the TMR difference value captured in the earlier instance is compared to the current difference using the same method.
Initialization	Timer: as required by the System Integrator's application
Additional Required Resources	None
Error Reporting	The diagnostic returns a software code indicating: pass (timer values are linear over a period of time); or fail (timer values are incrementing non-linearly over a period of time).
Diagnostic Type	Software supported by internal hardware
Recommended Mitigation Action	Application dependent
Periodicity	Periodic
Recommendations and Limitations	None
Relevant Software Requirements	SW_TIMER_LINEARITY_TEST_01
Relevant Hardware Requirements	None
Dataflow Dependency	Independent