IEC60730 Safety Library for TMS320F2833x

USER'S GUIDE

Copyright

Texas Instruments
12203 Southwest Freeway
Houston, TX 77477
http://www.ti.com/c2000

Revision Information

This is version v4.00.01.00 of this document, last updated on January,27 2014.

Table of Contents

1 Introduction

Manufacturers of household appliances must take steps to ensure safe and reliable operation of their products in order to meet the IEC60730 standard. The IEC60730 standard covers mechanical, electrical, electronic, EMC, and abnormal operation of ac appliances. Annex H of this standard covers the aspects most relevant to microcontrollers including the three software classifications defined for automatic electronic controls:

Class A functions such as room thermostats, humidity controls, lighting controls, timers and switches. These are distinguished by not being relied upon for the safety of the equipment.

Class B functions such as thermal cut-offs are intended to prevent unsafe operation of appliances such as washing machines, dishwashers, dryers, refrigerators, freezers and cookers/stoves.

Class C functions are intended to prevent special hazards such as explosions. These include automatic burner controls and thermal cut-outs for closed, unvented water heaters.

2 IEC60730 Self Test Library

2.1 IEC60730 Self Test Library

Current and future Delfino microcontrollers are designed for safety-critical industrial and consumer applications, offering integrated features. Hardware features such as – write-protected registers, limp mode and supervisory circuits – have all been integrated in Delfino MCUs. These features, tailored to the test requirements of IEC60730, make compliance in the electronic segment of the tests easier and the results more predictable.

Most home appliances including washing machines, dryers, refrigerators, freezers, and cookers/stoves fall into the Class B classification.

To fulfill Class B compliance, manufacturers must test specific components of the design. Table H.11.12.7 in Annex H of the IEC60730 standard lists the MCU components to be tested, the faults to be detected, and the appropriate reactive measures.

The Texas Instruments IEC60730 self-test library (STL) is a collection of optimized independent test functions for Delfino TMS320F2833x and Piccolo TMS320F2802x/F2803x/F2805x/F2806x MCUs. The self-test library includes C-callable optimized test functions. The library has to be built in with the application allowing the application to call the functional tests at a cyclic interval. Each functional test function returns the status of the test. In case of a failure return, the application needs to take appropriate action. Depending on the user settings, the core test functions can also jump to a fail safe routine in case of a test failure. In such cases the application has to implement a fail safe routine with appropriate action implemented in the function.

Table 2.1 below details the tests available in the STL as per table **H.1** of the **IEC60730 International Std ed. 2010 Annex H , pp 188 - 192** .

Test component	**Fault tested**	**IEC60730 Annex H Component**	**IEC60730 test definitions used**
CPU registers (CPU core)	Stuck at	1.1	H.2.16.5, H.2.16.6, H.2.19.6
CPU Porgaram Counter register	Stuck at	1.3	H.2.16.5, H.2.16.6, H.2.18.10.2
Interrupt Handling and Execution	No interrupt	2.0	H.2.16.5, H.2.18.10.4
Clock	Wrong frequency	3.0	H.2.18.10.1
Invariable Memory (Flash , ROM , OTP)	All single bit faults	4.1	H.2.19.4.1
Variable Memory	DC fault	4.2	H.2.19.6.2
Addressing	Stuck at	4.3	March13 N / CRC
Internal Data Path	Stuck at	5.1	March13 N / CRC
External communication	Wrong point in time	6.3	H.2.18.10.4
Digital I/O	Fault condition specified in H.27	7.1	H.2.18.13
Analog multiplexer	Wrong Addressing	7.2.1	H.2.18.13
Stack pointer corruption			Watch point
Illegal instruction detection			Hardware generated ISR

Table 2.1: IEC60730 Intl Std ed.3.2 2010 Annex H Tests

3 Installing the IEC60730 Safety Library

3.1 Installing the IEC60730 Safety Library

The C28x IEC60730 Library is distributed through the controlSUITE installer. The user must select the IEC60730 Safety Library Checkbox to install the library in the controlSUITE directory. By default the installation places the library components in the following directory.If not installing from controlSUITE, place the IEC60730_safety folder in the directory shown below.

```
base = C:\TI\controlSUITE\libs\IEC60730_safety\v4_00_01_00
```

The library is partitioned into a well-defined directory structure as shown in the figure 3.1 and described in table 3.1 below.

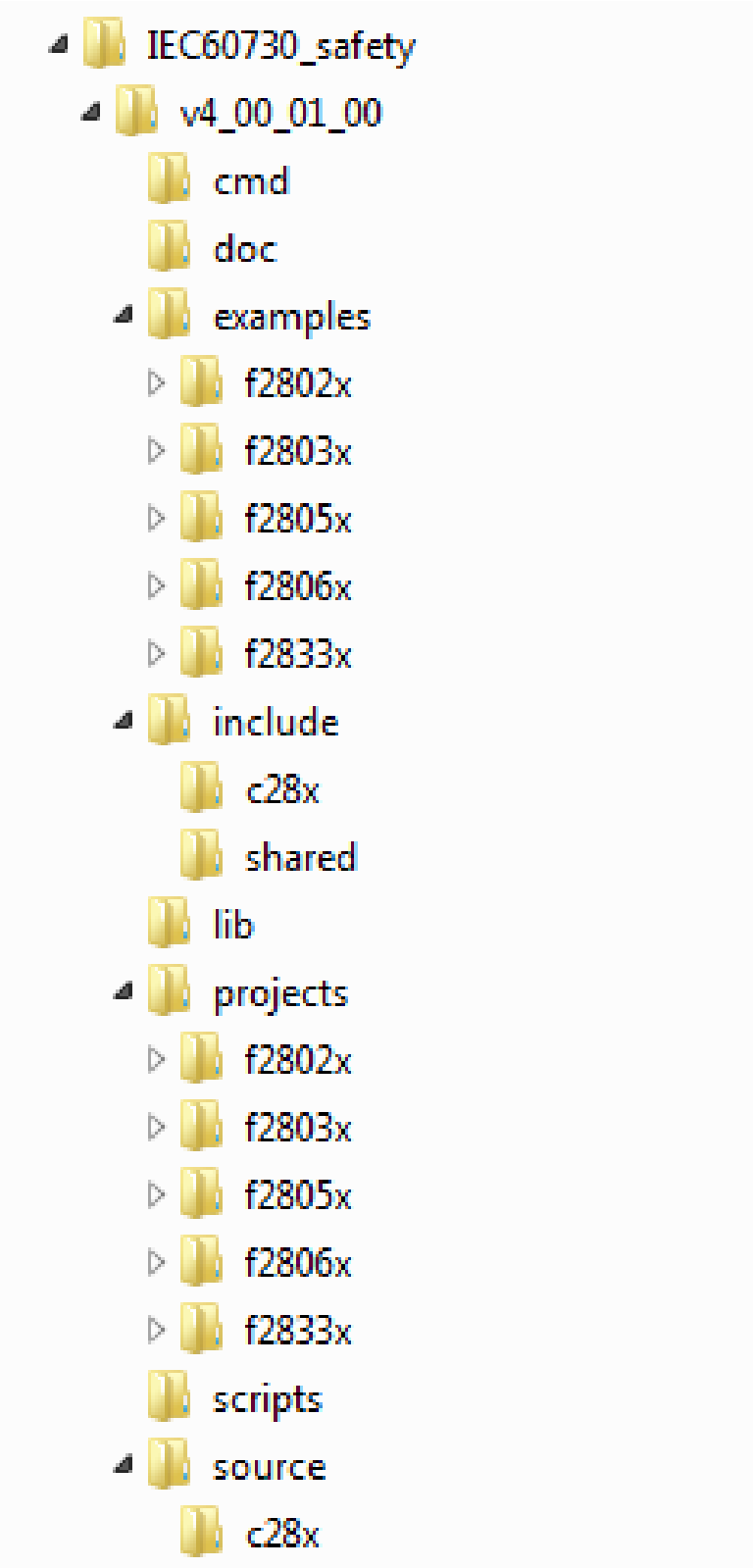

Figure 3.1: IEC60730 Safety Library Folder Structure

Directory	Contents
base	the directory described above.
base/cmd	Linker command files used in the examples.
base/doc	User guide documentation.
base/examples/f2833x	Example projects using CCSv5.
base/include/c28x	c28x specific header files.
base/include/shared	Device independent header files.
base/lib	Pre-built IEC60730 safety libraries
base/projects/f2833x	Project files used to build the library
base/scripts	Perl script to generate CRC32 on library binary file.
base/source/c28x	Library source files

Table 3.1: IEC60730 Safety Library Directory Structure Description

4 Using the IEC60730 STL

4.1 Overview of the Library

The following sub sections briefly list and describe the error handling mechanism, contents, coverage and benchmarks of the IEC60730 safety library.

4.1.1 Error Handling

All the API functions in the IEC60730 library return the status of the test. If the test passes, the functions return a unique success status code. All the success status codes are defined in STL_system_config.h. The codes start with SIG_ followed by a representation of the test. For example SIG_CPU_REG_TEST, represents the success code for CPU register test. Upon failure of the test, however, the core API functions (RAM ,CPU register,PC register,Interrupt, and Oscillator tests) can either return TEST_FAILED (0) or jump to a STL_FAIL_SAFE_failSafe() routine if JUMP_TO_FAILSAFE is set to 1 in STL_user_config.h. The rest of the API functions, however,will only return the status of the test or in the case of stack corruption detection function the status of the register setup.

Note: The STL_FAIL_SAFE_failSafe() should be implemented by the user.

4.1.2 Library Files

The IEC60730 library consists of the following files. All source files with a .c and .asm extension have an associated header file that contains function prototypes and data structures. For example for STL_adc_test.c there is an associated STL_adc_test.h file that declares the function prototypes along with data structures.

File name	Source Description
STL_cpu_test.asm	CPU core, FPU and VCU register tests.
STL_march_test.asm	Volatile memory tests using March test.
STL_crc_test.asm	CRC based memory tests.
STL_interrupt_test.c	Interrupt functionality test.
STL_isr.c	Interrupt service routines used by the library.
STL_pc_test.c	Program counter register test.
STL_oscillator_test.c	Internal oscillator test.
STL_watchdog_test.c	Watchdog test.
STL_timer_test.c	CPU timers test.
STL_clock_fail_detect.c	Initializes missing clock detection logic.
STL_pll_lock_check.c	PLL lock check test.
STL_spc_detect.c	Initializes stack corruption detection.
STL_gpio_test.c	GPIO tests.
STL_type2_adc_test.c	ADC tests.
STL_type0_ecap_test.c	eCAP APWM mode test.
STL_type0_epwm_test.c	ePWM test.
STL_type0_ecan_test.c	eCAN internal loop back test.
STL_type0_i2c_test.c	I2C internal loop back test.
STL_type0_sci_test.c	SCI internal loop back test.
STL_type1_spi_test.c	SPI internal loop bacl test.
STL_part_id_test.asm	Silicon part id test.
STL_register_test.c	Peripheral registers stuck at test.
STL_register_test_patterns.c	Peripheral registers stuck at test masks.
STL_system_config.h	Contains macros used by the library.
STL_device.h	Contains device dependent include files.
STL_type.h	Standard data types.
STL_user_config.h	Contains all the user selectable configurations.
STL_utility.asm	Helper functions used by the IEC60730 Safety library.

Table 4.1: IEC60730 Safety library files

4.1.3 Function Benchmarks

Function name	Code size (bytes)	Stack Used (bytes)	Speed in Cycles[1]	
			Flash	RAM
STL_PART_ID_TESTcheckPartNumber	28	0	87	
STL_CPU_TEST_testCpuRegisters	270	16	593	
STL_CPU_TEST_testFpuRegisters	190	24	368	
STL_PC_TEST_testPcRegister	80	24	364	
STL_MARCH_TEST_testRam	602	0	33895[2]	
STL_MARCH_TEST_testSafeRam	740	0	44789[2]	
STL_CRC_TEST_testNvMemory	334	0	1542[2]	
STL_CRC_TEST_testSafeRam	784	0	23434[2]	
STL_CRC_TEST_testRam	146	0	14583[2]	
STL_SPC_DETECT_setUpSpcDetect	100	8		
STL_INTERRUPT_TEST_testInterrupt	452	96	23323	
STL_OSCILLATOR_TEST_testOscUsingSfo	88	64	83476	
STL_TYPE3_ADC_TEST_testAdcInput	322	20	7294[3]	
STL_GPIO_TESTtestGpioInput	322	20	653[3]	
STL_GPIO_TESTtestGpioOutput	474	24	1583[3]	
STL_TYPE1_SPI_TEST_testSpiLoopback	198	20	4190[3]	
STL_TYPE0_SCI_TEST_testSciLoopback	222	20	1497479[3]	
STL_TYPE0_I2C_TEST_testI2cLoopback	188	16	10184[3]	
STL_TYPE2_ECAN_TEST_testeCanLoopback	1300	144	154429[3]	
STL_WATCHDOG_TEST_testWatchdog	164 , 72[4]	56[5]	658285[3]	
STL_TIMER_TEST_testTimer	384 , 40[4]	68[5]	155437	
STL_TYPE1_EPWM_TEST_testEpwm	482 , 52[4]	60[5]	929047	
STL_TYPE0_ECAP_TEST_testEcapApwmMode	544 , 108[4]	96[5]	154429	
STL_REGISTER_TEST_testPeripheralRegisters	152	20	4092[6]	
STL_CLOCK_FAIL_DETECT_ enableClockFailDetect	16			
STL_CLOCK_FAIL_DETECT_ checkMissingClock	8			

[1] SYSCLKOUT is set to 150MHz, and flash wait states set to 5.
[2] For a memory size of 1024 bytes
[3] Depends on the instruction based delay count used. See the simple_demo example for exact delay count used.
[4] Size of the ISR used along with the test function.
[5] Includes stack used by ISR
[6] For 24 - 16bit wide registers.

Table 4.2: Benchmarks for F2833X

4.2 Using the Library in an Application

This section describes all the necessary steps that are required to rebuild and use the library in an application. The safety library comes with a pre-built library. However, the user can change certain features in the STL_user_config.h file and rebuild the library to suit the specific application need.

4.2.1 Library Build Options

The current version of the library was built with CCSv5 using C28x Codegen Tools v6.1.0 with the following options:

```
-v28
-mt
-ml
-g
--diag_warning=225
--display_error_number
--diag_wrap=off
```

4.2.2 Rebuilding the Library

The IEC60730 safety library ships with the original CCS project file that was used to build the library. This project can be used as a template to modify and rebuild the library.

1. Create a new workspace.
2. Import the project file found in ".../controlSUITE/libs/IEC60730_safety/v4_00_01_00/projects/F2833x/" folder.
3. Make the required changes and rebuild the project.
4. The new library will be placed in the ".../controlSUITE/libs/IEC60730_safety/v4_00_01_00/lib/" folder.

4.2.3 Using the Library Across Delfino Device Family

The IEC60730 safety library is designed in such a way that it can be used across the various Delfino family devices. The device has been tested with the super set Delfino silicon devices - F28335. Certain configuration macros need to be changed in order to rebuild and use the library across device variants.

1. Select the device family by changing the value of the appropriate macro in STL_user_config.h file to 1. The macros BUILD_LIB_F2806X, BUILD_LIB_F2805X , BUILD_LIB_F2803X ,BUILD_LIB_F2802X and BUILD_LIB_F2833X define the F2806x, F2805x, F2803x ,F2802x and F2833x device family respectively. Set **only one** of these macros.
2. Select the specific device type by changing the value of the appropriate macro in STL_user_config.h file to 1. For example for F28335 device, set the DEVICE_TYPE_28355 macro to 1. Set **only one** of the device type macros.

3. The device specific sections have been tailored for F28335 within the F2833x family. If other device types within this family are to be used, then the user needs to change macros that are specific to the silicon. These macros define attributes such as memory size , valid GPIO pins etc. These macros are defined in STL_system_config.h. For example for the F28335 specific silicon, all the device specific macros are defined between **#if DEVICE_TYPE_28335** and **#endif**.
   Note: See Appendix F and G for details on creating and building a CCSv5 library project.

4.2.4 External Dependencies

The IEC60730 safety library requires the user to add functions such as error handling function and sections in linker command file for proper operation of the library. The necessary additions and changes the user needs to implement in the application are listed below. All the required additions are also implemented in the example application that ships with the safety library. **These dependencies should be strictly followed when porting to other devices within the MCU family.**

1. Functions:
   (a) In case of a failed test, the library can either return error status (0) or jump to an error handling function depending on the value of JUMP_TO_FAILSAFE macro in STL_user_config.h file. If this macro is set to one , then the user has to implement a fail safe routine.The prototype for this function is
   "void STL_fail_safe_failSafe(void)"
2. Variables: The user needs to add the following variables exactly as shown for internal oscillator test using SFO library.
   (a) Scale factor used by the SFO library.
   "int32_t MEP_ScaleFactor[PWM_CH];"
   (b) EPWM register structures used by the SFO library.
   "volatile struct EPWM_REGS *ePWM[PWM_CH]=
   {&EPwm1Regs, &EPwm1Regs, &EPwm2Regs, &EPwm3Regs, &EPwm4Regs, &EPwm5Regs, &EPwm6Regs};"
3. Library:
   (a) SFO library is used for internal oscillator1 testing.The user needs to link the library to the application. The SFO library is located in ".../controlSUITE/device_support/F2833x/v133/DSP2833x_common/lib" directory. Make sure to include the header file "SFO_V5.h"
4. Linker file:
   (a) Three different code sections for CPU program counter register test. The sections should be named "pc_test_section_1", "pc_test_section_2", and "pc_test_section_3". Please Look at the simple_demo linker command file for their exact address location."pc_test_section_1" section should be in RAM whereas "pc_test_section_2" and "pc_test_section_3" section should be in flash.
   (b) A program section in RAM for CRC test function. This section should be named "psa_crc".
   (c) A program section in RAM for STL_UTILITY_delay() function. This section should be named "ramfuncs".
   (d) A data section in RAM for CRC test variable. This section should be named "STL_psa_crc_vars".

4.2.5 Integrating the IEC60730 STL Library into a Project

The IEC60730 safety library ships with a simple example project that calls all the IEC60730 library functions. This project can be used as a template for an application code that uses the safety library. The following list describes the required actions before using the safety library in an application. All of the following required items are added and implemented in the example project.

1. All the external dependencies should be added (see 4.2.4)
2. Link the safety library with the application project.
3. Copy the PSA CRC calculating function to RAM.
4. Copy STL_UTILITY_delay() function to RAM.
5. Copy all functions that alter the flash to RAM.
6. Always disable flash pre-fetch before calling the following functions that are used to calculate CRC. " STL_crc_test_testRam " , " STL_CRC_TEST_testSafeRam " , " STL_CRC_TEST_testNvMemory " and " STL_CRC_TEST_calculateCrc ". This ensures that the prefetch buffer is in a known state when reading non-volatile memory.
7. Generate CRCs once application development is complete. In the sample example project, STL_generate_CRC() function is used to calculate golden CRCs. This function is run from Flash A. Any flash sector can be used to run the function that calculates the CRC. In the simple example Flash A was used as the sector's CRC is not checked periodically.
8. Use "STL_crc_test_testRam" and "STL_crc_test_testSafeRam" to generate golden CRC values for RAM and safe RAM respectively and use "STL_CRC_TEST_getCrcResult" to obtain the CRC results.
9. Use "STL_CRC_TEST_testNvMemory" and "STL_CRC_TEST_getCrcResult" to get golden CRC values for non-volatile memory.
10. The march test performs a destructive test. As a result context saving may be required for some variables that may need to retain their value (for instance test counters). The application example allocates a section in M0 RAM (STL_Test_utility)for such variables and skips testing M0 RAM. The user can, however, test this area but must restore the values back.
11. Make sure the memory area containing STL_UTILITY_delay() is not overwritten when testing RAM. This function is used by some of the library functions. Make sure to restore the contents of this section, as with any other program section run from RAM, if this RAM region is tested.
12. Both the volatile and non volatile memory tests operate on 32 bit wide memory data. Make sure the start and end address you provide to the memory test functions safisfies these criteria. Otherwise, unintended memory locations could be destructively tested in the case of RAM March test.
13. Always disable interrupts before calling the safety library functions.
14. When testing a communication module in loop-back mode, disconnect the respective GPIO pins from the communication module using the GPIO MUX registers.

5 Function Descriptions

The following functions are included in this release of the IEC60730 Safety Library. The example was built using **CGT 6.1.0** with the following options:

```
-v28 -mt -ml -g --float_support=fpu32
--diag_warning=225
```

Note: The Return description in this section assumes that the user has set the JUMP_TO_FAILSAFE macro to 0 in STL_user_config.h.

5.1 User Configuration Macros

Defines

- BUILD_LIB_F2802X
- BUILD_LIB_F2803X
- BUILD_LIB_F2805X
- BUILD_LIB_F2806X
- BUILD_LIB_F2833X
- JUMP_TO_FAILSAFE
- MAXIMUM_ADC_COUNT_DRIFT
- MAXIMUM_ECAP_APWM_DUTYCYCLE_DRIFT
- MAXIMUM_ECAP_APWM_PERIOD_DRIFT
- MAXIMUM_WATCHDOG_DELAY_DRIFT
- MILLI_SECOND_DELAY
- MINIMUM_ADC_COUNT_DRIFT
- MINIMUM_ECAP_APWM_DUTYCYCLE_DRIFT
- MINIMUM_ECAP_APWM_PERIOD_DRIFT
- USE_MARCH13N_TEST

5.1.1 Define Documentation

5.1.1.1 BUILD_LIB_F2802X

Definition:

```
#define BUILD_LIB_F2802X
```

Description:

Piccolo 2802x library build option: set to 1 to build lib for 2802x devices

5.1.1.2 BUILD_LIB_F2803X

Definition:

```
#define BUILD_LIB_F2803X
```

Description:

Piccolo 2803x library build option: set to 1 to build lib for 2803x devices

5.1.1.3 BUILD_LIB_F2805X

Definition:

```
#define BUILD_LIB_F2805X
```

Description:

Piccolo 2805x library build option: set to 1 to build lib for 2805x devices

5.1.1.4 BUILD_LIB_F2806X

Definition:

```
#define BUILD_LIB_F2806X
```

Description:

Piccolo 2806x library build option: set to 1 to build lib for 2806x devices

5.1.1.5 BUILD_LIB_F2833X

Definition:

```
#define BUILD_LIB_F2833X
```

Description:

Delfino 2833x library build option: set to 1 to build lib for 2833x devices

5.1.1.6 JUMP_TO_FAILSAFE

Definition:

```
#define JUMP_TO_FAILSAFE
```

Description:

Fail safe option: 1 will cause the library to jump to a fail safe routine,0 will return status

5.1.1.7 MAXIMUM_ADC_COUNT_DRIFT

Definition:

```
#define MAXIMUM_ADC_COUNT_DRIFT
```

Description:

Acceptable maximum ADC count difference

5.1.1.8 MAXIMUM_ECAP_APWM_DUTYCYCLE_DRIFT

Definition:

```
#define MAXIMUM_ECAP_APWM_DUTYCYCLE_DRIFT
```

Description:

Acceptable maximum eCAP APWM duty cycle difference

5.1.1.9 MAXIMUM_ECAP_APWM_PERIOD_DRIFT

Definition:

```
#define MAXIMUM_ECAP_APWM_PERIOD_DRIFT
```

Description:

Acceptable maximum eCAP APWM period count difference

5.1.1.10 MAXIMUM_WATCHDOG_DELAY_DRIFT

Definition:

```
#define MAXIMUM_WATCHDOG_DELAY_DRIFT
```

Description:

Acceptable minimum count for watchdog delay

5.1.1.11 MILLI_SECOND_DELAY

Definition:

```
#define MILLI_SECOND_DELAY
```

Description:

Count value for one millisecond delay for a given processor speed

5.1.1.12 MINIMUM_ADC_COUNT_DRIFT

Definition:

```
#define MINIMUM_ADC_COUNT_DRIFT
```

Description:

Acceptable minimum ADC count difference

5.1.1.13 MINIMUM_ECAP_APWM_DUTYCYCLE_DRIFT

Definition:

```
#define MINIMUM_ECAP_APWM_DUTYCYCLE_DRIFT
```

Description:

Acceptable minimum eCAP APWM duty cycle difference

5.1.1.14 MINIMUM_ECAP_APWM_PERIOD_DRIFT

Definition:

```
#define MINIMUM_ECAP_APWM_PERIOD_DRIFT
```

Description:

Acceptable minimum eCAP APWM period count difference

5.1.1.15 USE_MARCH13N_TEST

Definition:

```
#define USE_MARCH13N_TEST
```

Description:

March test algorithm option: 1 uses MarchC13N, 0 uses MarchC-

5.2 Silicon ID Test

Functions

- uint16_t STL_PART_ID_TEST_checkPartNumber (void)
  checks device part ID.

5.2.1 Function Documentation

5.2.1.1 STL_PART_ID_TEST_checkPartNumber

checks device part ID.

Prototype:

```
uint16_t
STL_PART_ID_TEST_checkPartNumber(void)
```

Description:

This C-callable assembly routine checks the device part Id. It checks for device family ID based on the build configuration set in STL_user_config header file: BUILD_LIB_F2802X, BUILD_LIB_F2803X or BUILD_LIB_F2806X.

The return value of this function can be cast to either uint16_t or uint32_t. If cast to uint16_t, it returns the test status code. If cast to uint32_t, it returns the part id in the upper 16 bits and the test status code in the lower 16 bits.

Returns:

- If the correct part id is read, the routine returns SIG_PART_NUM_TEST. Otherwise, it returns TEST_FAILED.

5.3 CPU Core Register Tests

Functions

- uint16_t STL_CPU_TEST_testCpuRegisters (void)
  Tests CPU registers.

5.3.1 Function Documentation

5.3.1.1 STL_CPU_TEST_testCpuRegisters

Tests CPU registers.

Prototype:

```
uint16_t
STL_CPU_TEST_testCpuRegisters(void)
```

Description:

This C-callable assembly routine tests CPU core registers for stuck at bits. The following registers are tested:

- ACC
- P
- XAR0 to XAR7
- XT
- SP
- IFR, IER and DBGIER
- ST0
- DP

Returns:

- If the test passes the routine returns SIG_CPU_REG_TEST. Otherwise, it returns TEST_FAILED.

5.4 Peripherals Registers Test

Data Structures

- _STL_REGISTER_TEST_regsStuckAtTest_Obj_

Functions

- uint16_t STL_REGISTER_TEST_testPeripheralRegisters (STL_REGISTER_TEST_registerTest_Handle registerTestHandle)

  Performs stuck at test for peripheral registers.

5.4.1 Data Structure Documentation

5.4.1.1 _STL_REGISTER_TEST_regsStuckAtTest_Obj_

Definition:

```
typedef struct
{
    uint16_t *pRegisterStartAddress;
    uint16_t *pRegisterEndAddress;
    const uint16_t *pTestPattern1;
    const uint16_t *pTestPattern2;
}
_STL_REGISTER_TEST_regsStuckAtTest_Obj_
```

Members:

pRegisterStartAddress Start register address of the peripheral.

pRegisterEndAddress End register address of the peripheral.

pTestPattern1 Pointer to the first test pattern array.
pTestPattern2 Pointer to the second test pattern array.

Description:
Defines the Peripheral register test object.

5.4.2 Function Documentation

5.4.2.1 STL_REGISTER_TEST_testPeripheralRegisters

Performs stuck at test for peripheral registers.

Prototype:
```
uint16_t
STL_REGISTER_TEST_testPeripheralRegisters(STL_REGISTER_TEST_registerTest_Handle
registerTestHandle)
```

Parameters:
registerTestHandle is the handle to a Peripheral test object

Description:
This functions performs a stuck at bit test on peripheral registers. The user provides the beginning and ending addresses of the peripheral registers that need to be tested along with the address for patterns. The pattern arrays are declared in STL_register_test_pattern.h file.

Returns:
- If non of the registers are stuck at a bit ,the function returns SIG_PERIPH_R_TEST. Otherwise, it returns TEST_FAILED.

5.5 Program Counter Register Test

Functions

- uint16_t STL_PC_TEST_testPcRegister (void)
  Tests Program Counter register for stuck at bits.

5.5.1 Function Documentation

5.5.1.1 STL_PC_TEST_testPcRegister

Tests Program Counter register for stuck at bits.

Prototype:
```
uint16_t
STL_PC_TEST_testPcRegister(void)
```

Description:

This function tests the Program Counter register for stuck at bits. The routine calls three different test functions that return their addresses. Their return value is compared to an array that holds the PC test functions' addresses. If all the values match the function passes, otherwise it fails. The PC test functions need to reside in separate memory locations such that ,by the time all of them are called, all the Program Counter register bits are set and cleared - indirectly testing the PC register for stuck at bits. The user must define three sections named "pc_test_section_1", "pc_test_section_2", and "pc_test_section_3" in a linker command file that will be used in the application project.

Returns:

- If the test passes the routine returns SIG_PC_TEST.Otherwise, it returns TEST_FAILED.

5.6 Invariable and Variable CRC Memory Tests

Functions

- uint64_t STL_CRC_TEST_calculateCrc (uint32_t *pStartAddress, uint16_t memorySize)
  Calculates the CRC of a given memory range.
- uint64_t STL_CRC_TEST_getCrcResult (void)
  returns the CRC value available in the PSA CRC result register.
- uint16_t STL_CRC_TEST_testNvMemory (uint32_t *pStartAddress, uint32_t *pEndAddress, uint64_t *pExpectedCrc, uint16_t nvMemType)
  Tests invariable (non volatile) memory (FLASH, OTP and BOOTROM memory).
- uint16_t STL_CRC_TEST_testRam (uint32_t *pStartAddress, uint32_t *pEndAddress, uint64_t *pExpectedCrc)
  Tests variable memory (RAM) for stuck at bits.
- uint16_t STL_CRC_TEST_testSafeRam (uint32_t *pCopyAddress, uint64_t *pExpectedCrc, uint16_t ramType)
  Tests variable memory (RAM) for stuck at bits.

5.6.1 Function Documentation

5.6.1.1 STL_CRC_TEST_calculateCrc

Calculates the CRC of a given memory range.

Prototype:

```
uint64_t
STL_CRC_TEST_calculateCrc(uint32_t *pStartAddress,
                          uint16_t memorySize)
```

Parameters:

pStartAddress is the start address of memory

memorySize is the length of memory

Description:

This function calculates and returns the CRC starting at pStartAddress. The length of the memory is given by memorySize. The size is incremented in 32 bit words. For example if the start address is 0 and the memory size is 4, the function calculates the CRC from address 0 to address 7 inclusive. This function can be used to obtain the golden CRC values of a non volatile Memory.

Returns:

Returns a 40 bit CRC value

5.6.1.2 STL_CRC_TEST_getCrcResult

returns the CRC value available in the PSA CRC result register.

Prototype:

```
uint64_t
STL_CRC_TEST_getCrcResult(void)
```

Description:

This function reads and returns the CRC results that area available in PSA CRC result registers. This function can be used to get the golden CRC values of the RAM CRC results.

Returns:

Returns a 40 bit CRC value

5.6.1.3 STL_CRC_TEST_testNvMemory

Tests invariable (non volatile) memory (FLASH, OTP and BOOTROM memory).

Prototype:

```
uint16_t
STL_CRC_TEST_testNvMemory(uint32_t *pStartAddress,
                          uint32_t *pEndAddress,
                          uint64_t *pExpectedCrc,
                          uint16_t nvMemType)
```

Parameters:

pStartAddress is the start address of memory to be tested

pEndAddress is the end address of memory to be tested

pExpectedCrc is a pointer to the expected CRC value

nvMemType is the non volatile memory type

Description:

This function is used to test non volatile memory for single bit errors. It uses a 40-bit polynomial to calculate CRC and detect errors. The CRC calculation is performed by Parallel Signature Analyzer(PSA). The user must define a section named "STL_psa_crc_vars" in RAM to hold a 32 bit wide variable.This variable (clearDataBus)is used by the CRC calculator to clear the Data bus. The user must also define a section named "psa_crc". This section will copy the core CRC calculation activation code (Calculate_PSA_CRC in STL_crc_test.asm) from flash

and run it from RAM (similar to other functions that are loaded from flash and must be run from RAM). This function has a length of 28 words and the size of RAM that this function will be copied and run from needs to be same size or greater. The function takes four parameters. pStartAddress and pEndAddress are inclusive memory address ranges of the CRC test.The range of memory that is tested should not exceed 65535 32 bit words. expectedCrc is the expected 40 bit golden CRC value. The expected CRC value is compared to the newly calculated CRC value. The test passes if the two CRC values are identical. Since the PSA calculates CRC on each CPU cycle (on every single pipeline), it will be difficult to exactly simulate the pipeline behaviour and generate a CRC value in an external CRC code. Instead, The function STL_CRC_TEST_calculateCrc() can be used to calculate the golden CRC values. nvMemType is the type of non volatile memory that can be tested. The following constants defined in STL_system_config header file can be passed as nvMemType parameter.

- NV_TYPE_FLASH
- NV_TYPE_USER_OTP
- NV_TYPE_TI_OTP1
- NV_TYPE_TI_OTP2
- NV_TYPE_BOOTROM

Returns:

- If the provided golden CRC value is identical to the calculated CRC value, the function will return SIG_NV_MEM_CRC_TEST. Otherwise, it returns TEST_FAILED.
- If the non volatile memory range to be tested is greater than 65535, the function returns 0.
- If the provided start and end addresses are outside of a non volatile memory region , the function returns 0.

5.6.1.4 STL_CRC_TEST_testRam

Tests variable memory (RAM) for stuck at bits.

Prototype:

```
uint16_t
STL_CRC_TEST_testRam(uint32_t *pStartAddress,
                     uint32_t *pEndAddress,
                     uint64_t *pExpectedCrc)
```

Parameters:

pStartAddress is the start address of memory to be tested

pEndAddress is the end address of memory to be tested

pExpectedCrc is a pointer to the expected CRC value

Description:

This function is used to test RAM for stuck at bit errors. As this test is destructive, users may need to save the contents of RAM to be tested into a separate RAM location. The test fills the RAM area under test with alternating patterns of 0 and 1 and calculates the CRC of the RAM using Parallel Signature Analyser(PSA). For a given RAM memory region if no bit is stuck in the RAM memory, the CRC value should always be identical. Since the core function that calculates the CRC is similar to the one used in STL_CRC_TEST_testNvMemory(), all the settings such as linker file settings must be similar to the one described in the STL_CRC_TEST_testNvMemory() API guide. The function takes three parameters. pStartAddress and pEndAddress are inclusive memory address ranges of the CRC test. The start

address needs to be an even value and the the end address need to be odd. The range of memory that is tested should not exceed 65535 32 bit words. pExpectedCrc is the address of the variable that holds the expected 40 bit golden CRC value. The expected CRC value is compared to the newly calculated CRC value. The value of pExpectedCrc should be an even number. The test passes if the two CRC values are identical. To generate the golden CRC value for RAM, call this function first and use STL_CRC_TEST_getCrcResult() to get the 40 bit CRC value.

Returns:

- If the calculated and provided CRC values are identical, the test passes and the routine returns SIG_RAM_CRC_TEST. Otherwise, it returns TEST_FAILED.
- If the RAM memory range to be tested is greater than 65535, the function returns TEST_FAILED.
- If the start address is odd and/or end address is even,the function returns TEST_FAILED.
- If the provided start and end addresses are in any of the safe RAM areas (see Appendix C) or in non RAM regions such as flash, the function returns TEST_FAILED.

5.6.1.5 STL_CRC_TEST_testSafeRam

Tests variable memory (RAM) for stuck at bits.

Prototype:

```
uint16_t
STL_CRC_TEST_testSafeRam(uint32_t *pCopyAddress,
                         uint64_t *pExpectedCrc,
                         uint16_t ramType)
```

Parameters:

pCopyAddress is the start address of memory to be tested

pExpectedCrc is a pointer to the expected CRC value

ramType is the the type of safe RAM that is to be tested

Description:

This function is used to test safe RAM areas for stuck at bit errors. It is similar to STL_CRC_TEST_testRam() function with added capability to save and restore safe RAM memory regions under test.(See Appenidx C for the list of safe RAM regions.) The function takes three parameters. pCopyAddress is the the start address of a RAM area where contents of safe RAM will be copied. This area must be outside of the safe RAM regions. pCopyAddress should have an even value. pExpectedCrc is the address of the expected 40 bit golden CRC value. This address value should be an even number. The expected CRC value is compared to the newly calculated CRC value. The test passes if the two CRC values are identical. The following RAM type constants defined in STL_system_config header file can be passed as the ramType parameter.

- RAM_TYPE_STACK
- RAM_TYPE_PIE_VECTOR
- RAM_TYPE_PSA_CRC
- RAM_TYPE_BOOT_RSVD
- RAM_TYPE_PC_TEST_1

Returns:

- If the provided golden CRC value is identical to the calculated CRC value the function will return SIG_RAM_CRC_TEST. Otherwise, it returns TEST_FAILED.
- If the start address is odd,the function returns TEST_FAILED.
- If the provided copy address is in an invalid region , i.e. in any of the safe RAM areas or in a non RAM region such as flash, the function returns TEST_FAILED.
- If the RAM type is different from any of the safe RAM type constants defined in STL_system_config header file, and listed above, the function returns TEST_FAILED.

5.7 Variable Memory Tests

Functions

- uint16_t STL_MARCH_TEST_testRam (uint32_t *pStartAddress, uint32_t *pEndAddress)
  Tests Variable memory (RAM memory).

- uint16_t STL_MARCH_TEST_testSafeRam (uint32_t *pCopyAddress, uint16_t ramType)
  Tests safe Variable memory (RAM memory).

5.7.1 Function Documentation

5.7.1.1 STL_MARCH_TEST_testRam

Tests Variable memory (RAM memory).

Prototype:

```
uint16_t
STL_MARCH_TEST_testRam(uint32_t *pStartAddress,
                       uint32_t *pEndAddress)
```

Parameters:

pStartAddress is the start address of RAM to be tested

pEndAddress is the end address of RAM to be tested

Description:

This function checks the RAM memory for DC fault using march test. The following two destructive march tests are implemented.

- MarchC 13N
- MarchC-

The user can select which test to use by setting or clearing USE_MARCH13N_TEST in STL_user_config header file. If USE_MARCH13N_TEST is set to 1 MarchC 13N algorithm will be used. If USE_MARCH13N_TEST is set to 0, MarchC- algorithm will be used. The function takes two parameters - the start and end address of the RAM memory to be tested. The test runs inclusive of these two addresses. Since the function performs a 32 bit read/write when testing the RAM cell arrays and because of the 16 bit architecture of the RAM cells, the function expects the start address to be even and the end address to be odd. The maximum memory range that can be tested is limited to 65535 32 bit words.

Returns:

- If the march test passes, the routine returns SIG_RAM_MARCH_TEST. Otherwise, it returns TEST_FAILED.
- If the RAM memory range to be tested is greater than 65535, the function returns TEST_FAILED.
- If the start address is odd or end address is even,the function returns TEST_FAILED.
- If the provided start and end addresses are in any of the safe RAM areas (see Appendix C) or in non RAM regions such as flash, the function returns TEST_FAILED.

See also:

USE_MARCH13N_TEST

5.7.1.2 STL_MARCH_TEST_testSafeRam

Tests safe Variable memory (RAM memory).

Prototype:

```
uint16_t
STL_MARCH_TEST_testSafeRam(uint32_t *pCopyAddress,
                           uint16_t ramType)
```

Parameters:

pCopyAddress is the start address of RAM where contents of the safe RAM to be tested are copied.

ramType is the the type of safe RAM that is to be tested

Description:

This function is used to check the safe RAM memory areas for DC faults using march test. It uses the same method utilized in STL_MARCH_TEST_testRam(). In addition to performing the march test, it saves the safe RAM areas under test to a RAM area provided by the user before the march test and restores the values once the test is completed. If USE_MARCH13N_TEST is set to 1 MarchC 13N algorithm will be used. If USE_MARCH13N_TEST is set to 0, MarchC- algorithm will be used. The function takes two parameters - the start address of a RAM area where contents of safe RAM will be copied and the safe RAM Type. The start address needs to be an even value. The safe RAM areas are listed in Appendix C. The following RAM type constants defined in STL_system_config header file can be passed as the ramType parameter.

- RAM_TYPE_STACK
- RAM_TYPE_PIE_VECTOR
- RAM_TYPE_PSA_CRC
- RAM_TYPE_BOOT_RSVD
- RAM_TYPE_PC_TEST_1

Returns:

- If the march test passes, the routine returns SIG_RAM_MARCH_TEST. Otherwise, it returns TEST_FAILED.
- If the start address is odd,the function returns TEST_FAILED.
- If the provided copy address is in an invalid region , i.e. in any of the safe RAM areas (see Appendix C) or in a non RAM region such as flash, the function returns TEST_FAILED.
- If the RAM type is different from any of the safe RAM type constants defined in STL_system_config header file, and listed above, the function returns TEST_FAILED.

See also:
USE_MARCH13N_TEST

5.8 Stack Corruption Detection

Functions

- uint16_t STL_SPC_DETECT_setUpSpcDetect (void (*pStackCorruptIsr)(void))
  Detects stack pointer corruption.

5.8.1 Function Documentation

5.8.1.1 STL_SPC_DETECT_setUpSpcDetect

Detects stack pointer corruption.

Prototype:

```
uint16_t
STL_SPC_DETECT_setUpSpcDetect(void (*pStackCorruptIsr)(void))
```

Parameters:

pStackCorruptIsr is a pointer to a function that doesn't take arguments and doesn't return a value (a typical interrupt function)

Description:

This function initializes the hardware debug watchpoint to detect stack pointer corruption (un-authorized stack region access). When un-authorized stack region is accessed, the RTOS interrupt will be invoked. The user is responsible to place a code that will take appropriate action inside the RTOS ISR. The debug watchpoint detects stack corruption indirectly by monitoring the Data Write Address Bus(DWAB). If an application tries to write data to any address range in the unauthorized region, the debug watchpoint will issue an RTOS ISR. The watchpoint is initialized to monitor 64 memory address ranges. A maximum of 40 address locations inside the stack will be used by the RTOS ISR.The stack pointer can move a maximum of 2 address locations.Hence, the stack corruption monitor should watch the last 42 address locations of the stack for un-authorized access. However the memory range that can be monitored can only be of value 2^N-1, where N = 1,2,... the next valid value which is 64 is selected. Since the address watchpoint uses masks in the configuration registers to monitor address ranges, additional 16 address locations are monitored. For example if the stack ends at address 0x250, the function will issue an RTOS interrupt when an access is made beginning at address 0x201 which is 16 address locations above the required address location of 0x210.

The function takes a pointer to a function that doesn't take arguments and doesn't return a value. The address to this function will be assigned to RTOS ISR and will be used as ISR that will get executed when stack region is corrupted

Returns:

- Returns 1 if the hardware debug watchpoint is initialized properly. Otherwise, it returns 0.

5.9 Interrupt Functionality Test

Functions

- uint16_t STL_INTERRUPT_TEST_testInterrupt (uint32_t *pCopyAddressStart, uint32_t isrDelay)

  tests functionality of interrupt.

5.9.1 Function Documentation

5.9.1.1 STL_INTERRUPT_TEST_testInterrupt

tests functionality of interrupt.

Prototype:

```
uint16_t
STL_INTERRUPT_TEST_testInterrupt(uint32_t *pCopyAddressStart,
                                 uint32_t isrDelay)
```

Parameters:

pCopyAddressStart is the start address of RAM where contents of user application ISR is copied to.

isrDelay is the ISR service delay

Description:

This function tests the functionality of the Interrupt. The function tests the functionality of the interrupt by firing all interrupts, when possible, and checking if all the fired interrupts occur according to their priority. It also checks the internal working of the PIE interrupt hardware by redundantly comparing the fetched vector address to the expected vector address. If all the ISRs are serviced according to their priority and if the correct vector is used to fetch the ISRs the interrupt test will pass. Otherwise, the test will fail. The function takes the start address of a RAM memory as its parameter. It will use the address as a base address to copy the ISR addresses of the user application. After the function copies user application ISR addresses into the RAM, it populates the PIE vector table with STL test ISR addresses. These ISRs are defined in STL_isr.c file. all the interrupts fired within this API function will be serviced by the ISRs in this file. Once the test is complete, the function will restore the user application ISRs addresses back to PIE vector. isrDelay is the value it will take STL_UTILITY_delay() function to finish counting down to zero.

Returns:

- If the interrupt test passes, the routine returns SIG_INTERRUPT_TEST. Otherwise, it returns TEST_FAILED.
- If the provided copy address is in an invalid region , i.e. in any of the safe RAM areas or in a non RAM region such as flash, the function returns TEST_FAILED.
- If VMAP bit in ST1 register and/or ENPIE in PIECTRL register is 0 , the function returns TEST_FAILED.

5.10 Internal OScillator Test

Data Structures

- _STL_OSCILLATOR_TEST_oscTestUsingTimer2_Obj_

Functions

- uint16_t STL_OSCILLATOR_TEST_testOscUsingSfo (int16_t mepMin, int16_t mepMax, uint32_t sfoDelay)
  Tests Oscillator using SFO for proper operation.
- uint16_t STL_OSCILLATOR_TEST_testOscUsingTimer2 (STL_OSCILLATOR_TEST_oscTestUsingTimer2_ oscTestHandle)
  Tests functionality of an Oscillator.

5.10.1 Data Structure Documentation

5.10.1.1 _STL_OSCILLATOR_TEST_oscTestUsingTimer2_Obj_

Definition:

```
typedef struct
{
    uint32_t usWait;
    uint32_t minCount;
    uint32_t maxCount;
    uint16_t oscSelection;
}
_STL_OSCILLATOR_TEST_oscTestUsingTimer2_Obj_
```

Members:

usWait delay value

minCount lower bound count

maxCount upper bound count

oscSelection Oscillator selection for timer 2.

Description:

Defines the OSC test object.

5.10.2 Function Documentation

5.10.2.1 STL_OSCILLATOR_TEST_testOscUsingSfo

Tests Oscillator using SFO for proper operation.

Prototype:

```
uint16_t
STL_OSCILLATOR_TEST_testOscUsingSfo(int16_t mepMin,
                                    int16_t mepMax,
                                    uint32_t sfoDelay)
```

Parameters:

mepMin is the lower bound inclusive value of acceptable values for MEP_ScaleFactor returned by SFO().

mepMax is the upper bound (inclusive) of acceptable values for MEP_ScaleFactor returned by SFO().

sfoDelay is the delay it takes for SFO() function to execute

Description:

This function tests the Oscillator (internal or external) that generates system clock (SYSCLKOUT) for accuracy and proper operation using the Scale Factor Optimization (SFO()) function. Before using this function make sure that the PLL is already configured and the HR-PWM clock is enabled. The Oscillator accuracy is verified by checking the MEP_ScaleFactor value returned by the SFO() function.

The SFO() function is used to calibrate the MEP scale factor for the device during run time. For a given System Clock frequency at a given temperature, a known MEP scale factor value is returned by the SFO() function. Proper System Clock frequency operation is verified by comparing the MEP scale factor value returned by the SFO() function with the expected value. If the System Clock frequency is verified, the Oscillator used to source the System Clock is also verified.

Since this function uses the SFO library, the user must integrate the SFO library with the application code that calls this routine. The function takes two arguments. mepMin is the lower acceptable inclusive boundary value. Valid values are in the range [1,...,mepMax]. mepMax is the upper acceptable inclusive boundary value. Valid values are in the range [mepMin,...,255]. The user also provides sfoDelay variable which is the instruction delay required by the SFO() function to return a valid value.

Note:

Look at the HRPWM chapter in the device data sheet for more information on the SFO library.

Returns:

- If the scale factor returned by the SFO() function is within the provided [mepMin,mepMax] range, the function will return SIG_OSC_TEST. Otherwise, it returns TEST_FAILED.

5.10.2.2 STL_OSCILLATOR_TEST_testOscUsingTimer2

Tests functionality of an Oscillator.

Prototype:

```
uint16_t
STL_OSCILLATOR_TEST_testOscUsingTimer2(STL_OSCILLATOR_TEST_oscTestUsingTimer2_Ha
oscTestHandle)
```

Parameters:

oscTestHandle is the handle to a test Oscillator object

Description:

This function tests the oscillator under test for accuracy and proper operation using CPU timer 2. The function takes a handle to STL_OSCILLATOR_TEST_sciTestUsingTimer2_Obj object as its parameter. The oscillator that is being tested and used as a clock source for timer 2 is selected by oscSelection variable. Valid inputs are 1,2 and 3 corresponding to external oscillator, internal oscillator 1 and internal oscillator 2 respectively. minCount and maxCount determine the valid timer count boundaries. usWait is the count for an instruction based delay as clocked by the system clock. The function tests the accuracy of the oscillator under test by using a different oscillator source to run timer 2. For instance if the system is running on internal oscillator 1, the user can test oscillator 2 by selecting oscSelection to a value of 3. During Oscillator testing, CPU Timer 2 is stopped, reconfigured, reloaded, and set to use the Oscillator selection in oscSelection as its clock source. CPU Timer 2 is then run for the specified duration and stopped again. The Oscillator under test is verified by checking the number of cycles elapsed during the specified duration, as measured by the CPU Timer 2 counter. It is important that the System Clock frequency which is used for the delay is verified prior to running this test. The user is responsible for saving the CPU Timer 2 configuration and state before testing and then restoring them afterwards. The following registers and bits are modified:

- CpuTimer2Regs
- SysCtrlRegs.CLKCTL.bit.TMR2CLKPRESCALE
- SysCtrlRegs.CLKCTL.bit.TMR2CLKSRCSEL

Note:

make sure that the system clock is run from a different oscillator source than the one selected by oscSelection.

Returns:

- If the oscillator counts are within the specified range, the function returns SIG_OSC_TEST. Otherwise, it returns TEST_FAILED.

5.11 CPU Timers Test

Data Structures

- _STL_TIMER_TEST_timerTest_Obj_

Functions

- uint16_t STL_TIMER_TEST_testTimer (STL_TIMER_TEST_timerTest_Handle timerTestHandle)

  Tests accuracy and functionality of CPU timers.

5.11.1 Data Structure Documentation

5.11.1.1 _STL_TIMER_TEST_timerTest_Obj_

Definition:

```
typedef struct
{
    uint16_t testTimer;
    uint16_t testScaler;
    uint32_t testPeriodCount;
    uint32_t delayCount;
}
_STL_TIMER_TEST_timerTest_Obj_
```

Members:

testTimer CPU timer to be tested. Valid inputs are 0,1, and 2.

testScaler CPU timer scaler to be used.

testPeriodCount CPU timer period for 1uS interrupt.

delayCount Instruction based delay count for 1mS.

Description:

Defines the Timer test object.

5.11.2 Function Documentation

5.11.2.1 STL_TIMER_TEST_testTimer

Tests accuracy and functionality of CPU timers.

Prototype:

```
uint16_t
STL_TIMER_TEST_testTimer(STL_TIMER_TEST_timerTest_Handle
timerTestHandle)
```

Parameters:

timerTestHandle is the handle to a Test timer object

Description:

This functions tests the CPU timer specified by the timerTestHandle for accuracy and interrupt generation capability. The function sets the timer under test to issue an interrupt every 1uS. After a delay of 1mS, the function checks if the timer has serviced close to 1000 interrupts. Depending on the value of SYSCLKOUT, the user needs to provide appropriate scaler , period and delay count. The value of delayCount will be passed to a delay routine STL_UTILITY_delay().

The user must preserve and restore the registers modified by this function. The following registers are modified:

- IFR
- IER
- All the registers of the timer under test

Returns:

- If the time increments to a value of 1000 within a 5 or -5 range within the provided delay value,the function returns SIG_TIMER_TEST. Otherwise, it returns TEST_FAILED.

5.12 Watchdog Test

Functions

- uint16_t STL_WATCHDOG_TEST_testWatchdog (uint32_t delayCount)
  Tests functionality of Watchdog timer.

5.12.1 Function Documentation

5.12.1.1 STL_WATCHDOG_TEST_testWatchdog

Tests functionality of Watchdog timer.

Prototype:

```
uint16_t
STL_WATCHDOG_TEST_testWatchdog(uint32_t delayCount)
```

Parameters:

delayCount watchdog interrupt delay

Description:

This function performs a functionality test on the watchdog timer. The watchdog timer is configured to increment an 8 bit counter every 512 OSCCLK cycle and generate an interrupt when the counter overflows. The function accepts the delayCount parameter as a delay count value. The delay value should correspond to the amount of time it takes the STL_UTILITY_delay() function to complete (512 x 128)OSCCLK. The user can also adjust the maximum count drift delay value by altering the value of MAXIMUM_WATCHDOG_DELAY_DRIFT macro in STL_utility header file. This ensures that an early interrupt doesn't pass as a success. At the occurrence of the interrupt if the value of the delay is greater than MAXIMUM_WATCHDOG_DELAY_DRIFT, then an interrupt has occurred too early. The function uses its own ISR to monitor a watchdog interrupt. Once the test is complete, it restores the watchdog ISR vector content to its pre-test value.

The user must preserve and restore the registers modified by this function. The following registers are modified:

- SCR
- WDCR
- PIECTRL
- PIEIER1
- IER

Returns:

- If the watchdog interrupt is issued within the delay specified by the STL_WATCHDOG_TEST_delay() function,the function returns SIG_WDT_TEST. Otherwise, it returns TEST_FAILED.

5.13 Missing Clock Detection

Functions

- uint16_t STL_CLOCK_FAIL_DETECT_setUpClockFailDetect (void (*pFailedClockIsr)(void), uint16_t delayCount)
  Sets up clock failure detection.

5.13.1 Function Documentation

5.13.1.1 STL_CLOCK_FAIL_DETECT_setUpClockFailDetect

Sets up clock failure detection.

Prototype:

```
uint16_t
STL_CLOCK_FAIL_DETECT_setUpClockFailDetect(void (*pFailedClockIsr)(void),
                                           uint16_t delayCount)
```

Parameters:

pFailedClockIsr is a pointer to a function that doesn't take arguments and doesn't return a value (a typical interrupt function)

delayCount is the NMI watchdog period count.

Description:

The function activates clock fail detection module to generate an NMI interrupt. The user provides a pointer to the interrupt service routine that will be called when a clock fails. The user also has to assign the required amount of delay before an NMI watchdog reset occurs. The NMI counter increments at a rate of SYSCLKOUT and resets the device when its value reaches delayCount.

Returns:

- Returns 1 if no clock fail interrupt is pending, otherwise it returns 0.

5.14 PLL Lock Check

Functions

- uint16_t STL_PLL_LOCK_CHECK_checkPllLock (void)
  Checks is PLL has locked in.

5.14.1 Function Documentation

5.14.1.1 STL_PLL_LOCK_CHECK_checkPllLock

Checks is PLL has locked in.

Prototype:

```
uint16_t
STL_PLL_LOCK_CHECK_checkPllLock(void)
```

Description:

This functions checks if the PLL has locked in on a new frequency the user has set.

Returns:

- If the PLL has locked in the new frequency,the function returns SIG_PLL_LOCK. Otherwise, it returns TEST_FAILED.

5.15 Analog-to-Digital Converter Test

Data Structures

- _STL_TYPE2_ADC_TEST_adcTest_Obj_

Functions

- uint16_t STL_TYPE2_ADC_TEST_testAdcInput (STL_TYPE2_ADC_TEST_adcTest_Handle adcTestHandle)

  Tests functionality of ADC converter.

5.15.1 Data Structure Documentation

5.15.1.1 _STL_TYPE2_ADC_TEST_adcTest_Obj_

Definition:

```
typedef struct
{
    uint16_t pinACount;
    uint16_t pinBCount;
    uint16_t muxChannel;
    uint16_t singleChannelSelect;
    uint32_t delayCount;
}
_STL_TYPE2_ADC_TEST_adcTest_Obj_
```

Members:

pinACount ADC count that is compared with value sampled by channel A.

pinBCount ADC count that is compared with value sampled by channel B.

muxChannel ADC pin mux to be sampled.
singleChannelSelect Single channel selection option.
delayCount Instruction based delay count.

Description:
Defines the ADC test object.

5.15.2 Function Documentation

5.15.2.1 STL_TYPE2_ADC_TEST_testAdcInput

Tests functionality of ADC converter.

Prototype:
```
uint16_t
STL_TYPE2_ADC_TEST_testAdcInput(STL_TYPE2_ADC_TEST_adcTest_Handle
adcTestHandle)
```

Parameters:
← ***adcTestHandle*** is the handle to a test ADC object

Description:
This functions performs a plausibility check on the ADC. The proper operation of : the pin mux selection, sample and hold circuit and the A/D converter is checked with this function. This function activates the A/D to perform simultaneous sampling on channel A and B. Single channel result comparison is possible by setting singleChannelSelect to a value of 1 or 2 If singleChannelSelect is set to 1, only channel A will be compared. If singleChannelSelect is set to 2 only channel B will be compared. To compare results from both channels set the value of singleChannelSelect to 0. The function takes a handle to STL_TYPE2_ADC_TEST_adcTest_Obj object as its parameter. The object has five parameters. Sampled channel is selected by muxChannel. For example, if muxChannel is set to 2, both ADCINA2 and ADCINB2 pins will be sampled. Once the conversion on both A and B sample and hold circuits is done, the result is compared with pinACount and pinBCount respectively. Valid muxChannel values are from 0 to 7. The user can define the acceptable ADC count drift by adjusting the values of MINIMUM_ADC_COUNT_DRIFT and MAXIMUM_ADC_COUNT_DRIFT macros in STL_user_config header file. delayCount takes a value that will be passed to a delay routine STL_UTILITY_delay(). This value should be such that it gives enough delay for the ADC to complete sampling and conversion. Please look at the device users guide and data sheet to determine the appropriate amount of delay required

Note:
The function doesn't initialize the ADC. Hence,the user must initialize the ADC.

The user must preserve and restore the registers modified by this function. The following registers are modified:

- PIEIER1
- ADCCTL3
- ADCCTL2
- ADCCTL1
- ADCMAXCONV

- ADCSAMPLEMODE
- ADCCHSELSEQ1
- ADCST

Returns:

- If the counts provided by the user match the converted counts,the function returns SIG_ADC_TEST. Otherwise, it returns TEST_FAILED.

5.16 eCAP APWM Mode Test

Data Structures

- _STL_TYPE0_ECAP_TEST_ecapApwmTest_Obj_

Functions

- uint16_t STL_TYPE0_ECAP_TEST_testEcapApwmMode (STL_TYPE0_ECAP_TEST_ecapApwmTest_Har ecapApwmTestHandle)

  Tests accuracy and functionality of eCAP module in APWM mode.

5.16.1 Data Structure Documentation

5.16.1.1 _STL_TYPE0_ECAP_TEST_ecapApwmTest_Obj_

Definition:

```
typedef struct
{
    uint16_t testEcap;
    uint16_t inverseDutyCycle;
    uint32_t dutyCycleCount;
    uint32_t periodCount;
    uint32_t periodCompare;
    uint32_t timeoutCount;
}
_STL_TYPE0_ECAP_TEST_ecapApwmTest_Obj_
```

Members:

testEcap eCAP module to be tested. Valid inputs are 1,2, and 3

inverseDutyCycle Ratio of period to duty cycle.

dutyCycleCount On time to be used.

periodCount Period for the square wave.

periodCompare Period value to be compared.

timeoutCount Timeout count for delay.

Description:

Defines the eCAP APWM test object.

5.16.2 Function Documentation

5.16.2.1 STL_TYPE0_ECAP_TEST_testEcapApwmMode

Tests accuracy and functionality of eCAP module in APWM mode.

Prototype:

```
uint16_t
STL_TYPE0_ECAP_TEST_testEcapApwmMode(STL_TYPE0_ECAP_TEST_ecapApwmTest_Handle
ecapApwmTestHandle)
```

Parameters:

ecapApwmTestHandle is the handle to a test eCAP object

Description:

This functions tests the functionality and accuracy of eCAP module operating in APWM mode. The user provides the following values in the STL_TYPE0_ECAP_TEST_ecapApwmTest_Obj object for a given SYSCLKOUT.

- testEcap the eCAP to be tested in APWM mode valid values are 1,2,and 3.
- inverseDutyCycle Period to duty cycle ratio.
- dutyCycleCount On time count.
- periodCount Period count.
- periodCompare Expected Period count.
- timeoutCount A timeout delay for STL_UTILITY_delay() function.

The macros MAXIMUM_ECAP_APWM_DUTYCYCLE_DRIFT , MINIMUM_ECAP_APWM_DUTYCYCLE_DRIFT and MINIMUM_ECAP_APWM_PERIOD_DRIFT , MAXIMUM_ECAP_APWM_PERIOD_DRIFT in STL_user_config header file can be adjusted to change the threshold value for the duty cycle and Period respectively.

After a delay of timeoutCount , the function checks if the given period count matches the actual eCAP Period count. If the two values don't match within the provided Period threshold , the function return error. Otherwise ,the function compares the inverse of actual eCAP duty cycle with the provided Period to duty cycle ratio. If these two values match within the provided threshold , the function returns a success value. Otherwise , it return error. The user must preserve and restore the registers modified by this function. The following registers are modified:

- IFR
- IER
- PIEIER4
- PIEACK
- CAP1, CAP2, ECCTL2, ECEINT, ECCLR, registers of the eCAP module under test

Returns:

- If the test passes, the function returns SIG_ECAP_APWM_TEST. Otherwise, it returns TEST_FAILED.

5.17 ePWM Test

Functions

- uint16_t STL_TYPE0_EPWM_TEST_testEpwm (void)
  Tests functionality of ePWM.

5.17.1 Function Documentation

5.17.1.1 STL_TYPE0_EPWM_TEST_testEpwm

Tests functionality of ePWM.

Prototype:

```
uint16_t
STL_TYPE0_EPWM_TEST_testEpwm(void)
```

Description:

This function tests the EPWM functionality: 6 ePWM's (ePWM1 - ePWM6) are initialized (same period, started sync'ed), an interrupt is taken on a zero event for each ePWM timer so that each of them will take an interrupt every event, every 2nd event or every 3rd event respectively. The interrupt counts are then compared.

The user must preserve and restore the registers modified by this function. The following registers are modified:

- IER
- TBPRD, TBCTL, ETSEL, ETSEL, ETPS, TBPHS registers of ePWM1,ePWM2,ePWM3,ePWM4,ePWM5 and ePWM6
- PCLKCR0

Returns:

- The function returns SIG_EPWM_TEST on success. Otherwise, it returns TEST_FAILED.

5.18 Gpio Test

Functions

- uint16_t STL_GPIO_TEST_testAioInput (uint16_t aioPin, uint16_t expectedValue, uint32_t gpioDelay)
  Tests functionality of AIO module.
- uint16_t STL_GPIO_TEST_testAioOutput (uint16_t aioPin, uint32_t gpioDelay)
  Tests functionality of AIO module.
- uint16_t STL_GPIO_TEST_testGpioInput (uint16_t gpioPin, uint16_t expectedValue, uint32_t gpioDelay)

Tests functionality of GPIO module.

- uint16_t STL_GPIO_TEST_testGpioOutput (uint16_t gpioPin, uint32_t gpioDelay)
  Tests functionality of GPIO module.

5.18.1 Function Documentation

5.18.1.1 STL_GPIO_TEST_testAioInput

Tests functionality of AIO module.

Prototype:

```
uint16_t
STL_GPIO_TEST_testAioInput(uint16_t aioPin,
                           uint16_t expectedValue,
                           uint32_t gpioDelay)
```

Parameters:

aioPin is the AIO pin number that is to be tested.

expectedValue is the expected voltage level at the pin.

gpioDelay is GPIO delay count

Description:

This functions performs an input plausibility check on AIO (Analog Input Output) module. The function reads the pin values specified by the aioPin parameter and compares it to the expected value as supplied by the expectedValue parameter. Valid values for aioPin are 2,4,6,10,12 and 14 corresponding to the respective AIO pins. The value of expectedValue should be either 0 or 1. gpioDelay is the value it will take STL_UTILITY_delay() function to finish counting down to zero. This delay is required for a write to configuration registers to be valid. Please look at the users guide and data sheet for the appropriate amount of delay.

The user must preserve and restore the registers modified by this function. The following registers are modified:

- AIOMUX1
- AIODIR

Returns:

- If the counts provided by the user match the converted counts,the function returns SIG_AIO_INPUT_TEST. Otherwise, it returns TEST_FAILED.

5.18.1.2 STL_GPIO_TEST_testAioOutput

Tests functionality of AIO module.

Prototype:

```
uint16_t
STL_GPIO_TEST_testAioOutput(uint16_t aioPin,
                            uint32_t gpioDelay)
```

Parameters:

aioPin is the AIO pin number that is to be tested.

gpioDelay is GPIO delay count

Description:

This functions performs an output plausibility check on AIO (Analog Input Output) module. The function sets and clears the pin specified by the aioPin. The function uses AIOTOGGLE register to alter the output value, and AIODAT register to read and verify if the output value is changed accordingly. Valid values for aioPin are 2,4,6,10,12 and 14 corresponding to the respective AIO pins. gpioDelay is the value it will take STL_UTILITY_delay() function to finish counting down to zero. This delay is required for a write to configuration registers to be valid. Please look at the users guide and data sheet for the appropriate amount of delay.

The user must preserve and restore the registers modified by this function. The following registers are modified:

- AIOMUX1
- AIODIR
- AIOTOGGLE

Note:

The function alters the voltage level of the pin under test.

Returns:

- If the counts provided by the user match the converted counts,the function returns SIG_AIO_OUTPUT_TEST. Otherwise, it returns TEST_FAILED.

5.18.1.3 STL_GPIO_TEST_testGpioInput

Tests functionality of GPIO module.

Prototype:

```
uint16_t
STL_GPIO_TEST_testGpioInput(uint16_t gpioPin,
                            uint16_t expectedValue,
                            uint32_t gpioDelay)
```

Parameters:

gpioPin is the GPIO pin number that is to be tested.

expectedValue is the expected voltage level at the pin.

gpioDelay is GPIO delay count

Description:

This function performs an input plausibility check on the GPIO module. The function reads the voltage level at the pin specified by the gpioPin parameter and compares it to the value of expectedValue. Valid values for gpioPin depend on the specific device used. gpioDelay is the value it will take STL_UTILITY_delay() function to finish counting down to zero. This delay is required for a write to configuration registers to be valid. Please look at the users guide and data sheet for the appropriate amount of delay.

A pin mask is used to determine valid pins. The macros VALID_GPIOA_PIN_MASK and VALID_GPIOB_PIN_MASK in STL_system_config header file dictate valid GPIO pins to be tested. For example bit 0 represents pin 0. A pin is valid for GPIO testing if the corresponding bit in the macros is set.

The user must preserve and restore the registers modified by this function. The following registers are modified:

- GPBMUX1, GPAMUX1, GPBMUX2, GPAMUX2
- GPBDIR, GPADIR
- GPBTOGGLE, GPATOGGLE

Returns:

- If the counts provided by the user match the converted counts,the function returns SIG_GPIO_INPUT_TEST. Otherwise, it returns TEST_FAILED.

5.18.1.4 STL_GPIO_TEST_testGpioOutput

Tests functionality of GPIO module.

Prototype:

```
uint16_t
STL_GPIO_TEST_testGpioOutput(uint16_t gpioPin,
                             uint32_t gpioDelay)
```

Parameters:

gpioPin is the GPIO pin number that is to be tested.

gpioDelay is GPIO delay count

Description:

This functions performs an output plausibility check on the GPIO module. The function sets and clears the pin specified by gpioPin. The function uses GPxTOGGLE register to alter the output value, and GPxDAT register to read and verify if the output value is changed accordingly - where x is A or B depending on the pin. Valid values for gpioPin are 0 to 44 corresponding to the respective GPIO pins. gpioDelay is the value it will take STL_UTILITY_delay() function to finish counting down to zero. This delay is required for a write to configuration registers to be valid. Please look at the users guide and data sheet for the appropriate amount of delay.

A pin mask is used to determine valid pins. The macros VALID_GPIOA_PIN_MASK and VALID_GPIOB_PIN_MASK in STL_system_config header file dictate valid GPIO pins to be tested. For example bit 0 represents pin 0. A pin is valid for GPIO testing if the corresponding bit in the macros is set.

The user must preserve and restore the registers modified by this function. The following registers are modified:

- GPBMUX1, GPAMUX1, GPBMUX2, GPAMUX2
- GPBDIR, GPADIR
- GPBTOGGLE, GPATOGGLE

Note:

The function alters the voltage level of the pin under test.

Returns:

- If the counts provided by the user match the converted counts,the function returns SIG_GPIO_OUTPUT_TEST. Otherwise, it returns TEST_FAILED.

5.19 eCAN Loopback Test

Functions

- uint16_t STL_TYPE0_ECAN_TEST_testeCanLoopback (uint32_t cpuAccessDelay, uint32_t messageRecieveDelay, uint16_t eCanPort)
  Tests functionality of eCAN communication.

5.19.1 Function Documentation

5.19.1.1 STL_TYPE0_ECAN_TEST_testeCanLoopback

Tests functionality of eCAN communication.

Prototype:

```
uint16_t
STL_TYPE0_ECAN_TEST_testeCanLoopback(uint32_t cpuAccessDelay,
                                     uint32_t messageRecieveDelay,
                                     uint16_t eCanPort)
```

Parameters:

cpuAccessDelay is CPU access delay

messageRecieveDelay is message receive delay

eCanPort is the eCAN port

Description:

This functions performs a loop back test on eCAN module. The function transmits and receives messages using 16 Mailboxes and compares the transmitted value to the received value. The value of eCanPort provides the eCAN modules to be tested. The valid values for this variable are 0 and 1 for eCAN module A and B respectively. Valid values depend on he device.The eCAN module is set to run at a rate of 1 Mbps. The function accepts two delay parameters. These delay values prevent the module from staying forever in a while loop when checking the status of CCE bit of CANES register and CANTA register respectively.

The user must preserve and restore the registers modified by this function. The following registers are modified by the function.

- GPAPUD , GPAQSEL2 , GPAMUX2
- CANTIOC , CANRIOC
- CANMC
- CANTA
- CANRMP
- CANGIF0 , CANGIF1
- CANES
- CANBTC
- CANME
- CANMD
- CANMIM

- CANTRS

Returns:

- If the transmitted values are the same are the received values,the function returns SIG_ECAN_TEST. Otherwise, it returns TEST_FAILED.

5.20 I2C Loopback Test

Data Structures

- _STL_TYPE0_I2C_TEST_i2cTest_Obj_

Functions

- uint16_t STL_TYPE0_I2C_TEST_testI2cLoopback (STL_TYPE0_I2C_TEST_i2cTest_Handle i2cTestHandle)
  Tests functionality of I2C communication.

5.20.1 Data Structure Documentation

5.20.1.1 _STL_TYPE0_I2C_TEST_i2cTest_Obj_

Definition:

```
typedef struct
{
    uint8_t i2cPreScaler;
    uint16_t i2cOffClockDivider;
    uint16_t i2cOnClockDivider;
    uint32_t delayCount;
}
_STL_TYPE0_I2C_TEST_i2cTest_Obj_
```

Members:

i2cPreScaler I2C module clock scaler.
i2cOffClockDivider I2C master clock on count.
i2cOnClockDivider I2C master clock on count.
delayCount Delay for loop back mode.

Description:

Defines the I2C test object.

5.20.2 Function Documentation

5.20.2.1 STL_TYPE0_I2C_TEST_testI2cLoopback

Tests functionality of I2C communication.

Prototype:

```
uint16_t
STL_TYPE0_I2C_TEST_testI2cLoopback(STL_TYPE0_I2C_TEST_i2cTest_Handle
i2cTestHandle)
```

Parameters:

← ***i2cTestHandle*** is the handle to I2C test object

Description:

This functions performs a loop back test on I2C module. The function transmits two sets of addresses and data values. The function takes a handle to STL_TYPE0_I2C_TEST_i2cTest_Obj object as its parameter. i2cPreScaler, i2cOffClockCount and i2cOnClockCount are values for the registers I2CPSC, I2CCLKL and I2CCLKH corresponding to the module clock scaler , off time master clock divider and on time master clock divider. The function calls STL_UTILITY_delay() function with delayCount as its parameter. The value of delayCount corresponds to the delay it takes the i2c module to transmit and receiver 4 data values each 8 bits wide. The user must preserve and restore the registers modified by this function. The following registers are modified by the function.

- I2CSAR
- I2CPSC
- I2CCLKL , I2CCLKH
- I2CIER
- I2CMDR
- I2CFFTX , I2CFFRX
- I2CCNT
- I2CDXR

Returns:

- If the transmitted values are the same are the received values,the function returns SIG_I2C_TEST. Otherwise, it returns TEST_FAILED.

5.21 SCI Loopback Test

Data Structures

- _STL_TYPE0_SCI_TEST_sciTest_Obj_

Functions

- uint16_t STL_TYPE0_SCI_TEST_testSciLoopback (STL_TYPE0_SCI_TEST_sciTest_Handle sciTestHandle)

  Tests functionality of SCI communication.

5.21.1 Data Structure Documentation

5.21.1.1 _STL_TYPE0_SCI_TEST_sciTest_Obj_

Definition:

```
typedef struct
{
    uint16_t sciPort;
    uint8_t sciLowBitRate;
    uint8_t sciHighBitRate;
    uint32_t delayCount;
    uint8_t *pTestData;
    uint16_t testDataSize;
}
_STL_TYPE0_SCI_TEST_sciTest_Obj_
```

Members:

sciPort SCI port.

sciLowBitRate SCI bit rate divider.

sciHighBitRate SCI bit rate divider.

delayCount Delay for loop back mode.

pTestData Pointer to test data array.

testDataSize test data size

Description:

Defines the SCI test object.

5.21.2 Function Documentation

5.21.2.1 STL_TYPE0_SCI_TEST_testSciLoopback

Tests functionality of SCI communication.

Prototype:

```
uint16_t
STL_TYPE0_SCI_TEST_testSciLoopback(STL_TYPE0_SCI_TEST_sciTest_Handle
sciTestHandle)
```

Parameters:

← ***sciTestHandle*** is the handle to SCI test object

Description:

This functions performs a loop back test on SCI module. The function takes a handle to STL_TYPE0_SCI_TEST_sciTest_Obj object as its parameter. The sciPort parameter determines the SCI port to be tested. Valid inputs for sciPortdepend on the number of SPI modules available for the specific device under test. For example if the device has two SCI modules then valid inputs will be 0 and 1. The sciLowBitRate and sciHighBitRate values correspond to the values of SCILBAUD and SCIHBAUD registers respectively which determine the baud rate divider. The function uses the following settings for the SCI.

- 1 stop bit

- no parity
- 8 char bits

The user provides the test data. The pointer pTestData points to a test data array and testDataSize is the number of characters contained in the test data array.The function transmits the provided data and checks if all the values are received correctly. The function reads the status of the FIFO status bits until the value of delaycount reaches 0 or until the FIFO status bits are set - which ever comes first.

The user must preserve and restore the registers modified by this function. The following registers are modified by the function.

- SCIFFTX
- SCIFFRX
- SCIFFCT
- SCICCR
- SCICTL2
- SCIHBAUD,SCILBAUD
- SCICTL1

Returns:

- If the transmitted values are the same are the received values,the function returns SIG_SCI_TEST. Otherwise, it returns TEST_FAILED.

5.22 SPI Loopback Test

Data Structures

- _STL_TYPE1_SPI_TEST_spiTest_Obj_

Functions

- uint16_t STL_TYPE1_SPI_TEST_testSpiLoopback (STL_TYPE1_SPI_TEST_spiTest_Handle spiTestHandle)

  Tests functionality of SPI communication.

5.22.1 Data Structure Documentation

5.22.1.1 _STL_TYPE1_SPI_TEST_spiTest_Obj_

Definition:

```
typedef struct
{
    uint16_t spiPort;
    uint8_t spiBitRateDivider;
    uint32_t delayCount;
    uint8_t *pTestData;
```

```
    uint16_t testDataSize;
}
_STL_TYPE1_SPI_TEST_spiTest_Obj_
```

Members:

spiPort SPI port.

spiBitRateDivider SPI bit rate divider.

delayCount Delay for loop back mode.

pTestData Pointer to test data array.

testDataSize test data size

Description:

Defines the SPI test object.

5.22.2 Function Documentation

5.22.2.1 STL_TYPE1_SPI_TEST_testSpiLoopback

Tests functionality of SPI communication.

Prototype:

```
uint16_t
STL_TYPE1_SPI_TEST_testSpiLoopback(STL_TYPE1_SPI_TEST_spiTest_Handle
spiTestHandle)
```

Parameters:

← ***spiTestHandle*** is the handle to SPI test object

Description:

This functions performs a loop back test on SPI module. The function takes a handle to STL_TYPE1_SPI_TEST_spiTest_Obj object as its parameter. spiPort sets the port to be tested. Valid inputs for spiPort depend on the number of SPI modules available for the specific device under test. The macro MAXIMUM_SPI_MODULES in STL_system_config header file defines the maximum number of available SPI modules for the specific device. For example if the device has two SCI modules then valid inputs will be 0 and 1. The spiBitRateDivider is the value assigned to SPIBRR, which divides the LSPCLK clock. The user provides the test data. The pointer pTestData points to a test data array and testDataSize is the number of characters contained in the test data array. The function transmits the provided data and checks if all the values are received correctly. The function reads the status of the FIFO status bits until the value of delaycount reaches 0 or until the FIFO status bits are set - which ever comes first. It takes approximately 13 instruction cycles to read the FIFO status and decrement the delay.

The user must preserve and restore the registers modified by this function. The following registers are modified by the function.

- SPIFFTX
- SPIFFRX
- SPIFFCT
- SPIBRR
- SPIPRI
- SPICTL

- SPICCR

Returns:

- If the transmitted values are the same as the received values,the function returns SIG_SPI_TEST. Otherwise, it returns TEST_FAILED.

5.23 Illegal Instruction Detection

The MCU is equipped with Hardware that issues an ISR when an invalid instruction is decoded. Opcode value of 0x0000 corresponds to ITRAP0 and 0xFFFF corresponds to ITRAP1 when this opcodes are decoded an ILLEGAL isr is issued. The user should take appropriate action when this ISR is issued. Please look at the sample demo that ships with the saftey library.

The following utility functions are used by the IEC60730 Library.

5.24 Utility Functions

Functions

- uint16_t STL_UTILITY_delay (uint32_t *pDelayValue)
  Instruction based delay function.
- uint16_t STL_UTILITY_getStatus1Register (void)
  Returns the contents of ST1 register.
- uint16_t STL_UTILITY_validateRamAddress (uint32_t *pStartAddress, uint32_t *pEndAddress)
  Checks if the given memory address is in a RAM region.
- uint16_t STL_UTILITY_validateSafeRamAddress (uint32_t *pStartAddress, uint32_t *pEndAddress)
  Checks if the given memory address is outside safe RAM region.

5.24.1 Function Documentation

5.24.1.1 STL_UTILITY_delay

Instruction based delay function.

Prototype:

```
uint16_t
STL_UTILITY_delay(uint32_t *pDelayValue)
```

Parameters:

pDelayValue is pointer to a variable contaiing the delay count.

Description:

This functions performs an instruction based delay. It decrements the value of the variable dereferenced by pDelayValue until it reaches zero. The number of cycles it takes this function to reach zero when no interrupts are occuring is y = 14x + 32.519, where x is the delay value and y is the cyce delay.

Returns:

- Returns 0

5.24.1.2 STL_UTILITY_getStatus1Register

Returns the contents of ST1 register.

Prototype:

```
uint16_t
STL_UTILITY_getStatus1Register(void)
```

Description:

This function returns the value of ST1 register

Returns:

- Contents of ST1 register

5.24.1.3 STL_UTILITY_validateRamAddress

Checks if the given memory address is in a RAM region.

Prototype:

```
uint16_t
STL_UTILITY_validateRamAddress(uint32_t *pStartAddress,
                               uint32_t *pEndAddress)
```

Description:

This function checks if the given memory address is in a RAM region

Returns:

- Returns 1 if the given address is within RAM region, 0 if not.

5.24.1.4 STL_UTILITY_validateSafeRamAddress

Checks if the given memory address is outside safe RAM region.

Prototype:

```
uint16_t
STL_UTILITY_validateSafeRamAddress(uint32_t *pStartAddress,
                                   uint32_t *pEndAddress)
```

Description:

This function checks if the given memory address is outside safe RAM region areas

Returns:

- Returns 1 if the given address is within RAM region, 0 if not.

6 Library Design

6.1 Register stuck at test

IEC60730
Component 1.1, SW class B
Fault
Check Registers for stuck-at bits
Measures
Periodic using a static memory test (H.2.16.6, H.2.19.6)
Design

```
for each register:
{
    write alternating 0/1 pattern to the register
    read back and verify the contents of the register
    if there is an error
    {
        if JUMP_TO_FAILSAFE is 1
            call STL_test_fail_failSafe() *
        else return an error code
    }
    write alternating 1/0 pattern to the register
    read back and verify the contents of the register
    if there is an error
    {
        if JUMP_TO_FAILSAFE is 1
            call STL_test_fail_failSafe() *
        else return an error code
    }
```

```
}
return a pass code
```

Tests

This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_CPU_TEST_testCpuRegisters()
- STL_CPU_TEST_testFpuRegisters()
- STL_REGISTER_TEST_testPeripheralRegisters() - Restores register content

6.2 Program Counter register test

IEC60730

Component 1.3, SW class B

Fault

Check Program Register for stuck at fault

Measures

Periodic using a static memory test (H.2.16.6, H.2.19.6)

Design

```
- define 3 functions that return their address
  /*
    this functions are located in different addresses
    such that all the PC register bits are set and cleared
  */

/* fill an array with program counter test function addresses */
- array[0] = &pcTestFunction1;
- array[1] = &pcTestFunction2;
- array[2] = &pcTestFunction3;

/* compare the values in array to the value returned by pc test functions */
for(i = 0; i < 3; i++)
{
   if( array[i] != pcTestFunctioni)
   {
      if JUMP_TO_FAILSAFE is 1
         call STL_test_fail_failSafe() *
      else
         return an error code
   }
}

return a success code
```

Tests

This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_PC_TEST_testPcRegister()

- STL_TYPE0_CLA_TEST_testMpcRegister()

6.3 RAM MarchC13N test

IEC60730
Component 4.2, SW class B
Fault
Check RAM memory for DC fault
Measures
Periodic static memory test (H.2.19.6)
Design

```
- user provides
     - Start and End RAM memory address for the
       RAM range to be tested
if( user provides an invalid region)
{
    return an error code
}
/* fill RAM under test with 0 */
for(i = 0; i < n; i++)
{
   array[i] = 0x0000 0000
}

/* march from low memory address to high memory address */
for(i = 0; i < n; i++)
{
   if( array[i] != 0x0000 0000)
   {
      if JUMP_TO_FAILSAFE is 1
         call STL_test_fail_failSafe() *
      else
         return an error code
   }
   else
   {
      array[i] = 0xFFFF FFFF
      if( array[i] != 0xFFFF FFFF)
      {
         if JUMP_TO_FAILSAFE is 1
            call STL_test_fail_failSafe() *
         else
            return an error code
      }
   }

}

for(i = 0; i < n; i++)
{
```

```
   if( array[i] != 0xFFFF FFFF)
   {
      if JUMP_TO_FAILSAFE is 1
         call STL_test_fail_failSafe() *
      else
         return an error code
   }
   else
   {
      array[i] = 0x0000 0000
      if( array[i] != 0x0000 0000)
      {
         if JUMP_TO_FAILSAFE is 1
            call STL_test_fail_failSafe() *
         else
            return an error code
      }
   }
}


/* march from high memory address to low memory address */
for(i = n-1; i >= 0 ; i--)
{
   if( array[i] != 0x0000 0000)
   {
      if JUMP_TO_FAILSAFE is 1
         call STL_test_fail_failSafe() *
      else
         return an error code
   }
   else
   {
      array[i] = 0xFFFF FFFF
      if( array[i] != 0xFFFF FFFF)
      {
         if JUMP_TO_FAILSAFE is 1
            call STL_test_fail_failSafe() *
         else
            return an error code
      }
   }

}

for(i = n-1; i >= 0 ; i--)
{
   if( array[i] != 0xFFFF FFFF)
   {
      if JUMP_TO_FAILSAFE is 1
         call STL_test_fail_failSafe() *
      else
```

```
            return an error code
      }
      else
      {
         array[i] = 0x0000 0000
         if( array[i] != 0x0000 0000)
         {
            if JUMP_TO_FAILSAFE is 1
               call STL_test_fail_failSafe() *
            else
               return an error code
         }
      }
   }

  return a success code
```

Tests

This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_MARCH_TEST_testSafeRam()
- STL_MARCH_TEST_testRam()
- STL_TYPE0_CLA_TEST_testCpuToClaMsgRam()
- STL_TYPE0_CLA_TEST_testClaToCpuMsgRam()

6.4 RAM MarchC- test

IEC60730

Component 4.2, SW class B

Fault

Check RAM memory for DC fault

Measures

Periodic static memory test (H.2.19.6)

Design

```
- user provides
     - Start and End RAM memory address for the
       RAM range to be tested
if( user provides an invalid region)
{
    return an error code
}
/* fill RAM under test with 0 */
for(i = 0; i < n; i++)
{
   array[i] = 0x0000 0000
}

/* march from low memory address to high memory address */
for(i = 0; i < n; i++)
{
```

```
   if( array[i] != 0x0000 0000)
   {
      if JUMP_TO_FAILSAFE is 1
         call STL_test_fail_failSafe() *
      else
         return an error code
   }
   else
   {
      array[i] = 0xFFFF FFFF
   }

}

for(i = 0; i < n; i++)
{
   if( array[i] != 0xFFFF FFFF)
   {
      if JUMP_TO_FAILSAFE is 1
         call STL_test_fail_failSafe() *
      else
         return an error code
   }
   else
   {
      array[i] = 0x0000 0000
   }
}


/* march from high memory address to low memory address */
for(i = n-1; i >= 0 ; i--)
{
   if( array[i] != 0x0000 0000)
   {
      if JUMP_TO_FAILSAFE is 1
         call STL_test_fail_failSafe() *
      else
         return an error code
   }
   else
   {
      array[i] = 0xFFFF FFFF
   }

}

for(i = n-1; i >= 0 ; i--)
{
   if( array[i] != 0xFFFF FFFF)
   {
      if JUMP_TO_FAILSAFE is 1
```

```
            call STL_test_fail_failSafe() *
         else
            return an error code
      }
      else
      {
         array[i] = 0x0000 0000
      }
   }
   /* check if RAM contents are all 0 */
   for(i = 0; i < n; i++)
   {
      if( array[i] != 0x0000 0000)
      {
         if JUMP_TO_FAILSAFE is 1
            call STL_test_fail_failSafe() *
         else
            return an error code
      }
   }

   return a success code
```

Tests

This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_MARCH_TEST_testSafeRam()
- STL_MARCH_TEST_testRam()
- STL_TYPE0_CLA_TEST_testCpuToClaMsgRam()
- STL_TYPE0_CLA_TEST_testClaToCpuMsgRam()

6.5 Invariable memory CRC test

IEC60730

Component 4.1, SW class B/C

Fault

Check non volatile memory for single bit errors

Measures

Single word periodic cyclic redundancy check (H.2.19.4.1)

Design

```
- user provides
     - Start and End Non volatile memory address for the
       Memory range to be tested
if( user provides an invalid region)
{
    return an error code
}
- set the CRC seed value to 0
- clear contents of Data Read Data Bus (DRDB) to 0x0000 0000
```

```
- activate Parallel Signature Analyser (PSA) to calculate CRC on DRDB
for(i = 0; i < n; i++)
{
   - read contents of memory under test
   /*
    PSA will calculate CRC on each word that goes to CPU
    via Data Read Data Bus
   */
}

- deactivate Parallel Signature Analyser (PSA)

if( calculated CRC value != golden CRC)
{
      if JUMP_TO_FAILSAFE is 1
         call STL_test_fail_failSafe() *
      else
         return an error code
}

return a success code
```

Tests

This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_CRC_TEST_testNvMemory()

6.6 Variable memory CRC test

IEC60730

Fault

Check volatile memory for stuck at bit errors

Measures

Design

```
- user provides
     - Start and End RAM memory address for the
       RAM range to be tested
if( user provides an invalid region)
{
    return an error code
}
/* fill RAM area with 0x5555 5555 */
for(i = 0; i < n; i++)
{
   array[i] = 0x5555 5555
}
- set the CRC seed value to 0
- clear contents of Data Read Data Bus (DRDB) to 0x0000 0000
```

```
- activate Parallel Signature Analyser (PSA) to calculate CRC on DRDB
for(i = 0; i < n; i++)
{
   - read contents of memory under test (array[i])
   /*
    PSA will calculate CRC on each word that goes to CPU
    via Data Read Data Bus
   */
}

- deactivate Parallel Signature Analyser (PSA)

/* fill RAM area with 0xAAAA AAAA */
for(i = 0; i < n; i++)
{
   array[i] = 0xAAAA AAAA
}

- clear contents of Data Read Data Bus (DRDB) to 0x0000 0000
- activate Parallel Signature Analyser (PSA) to calculate CRC on DRDB
for(i = 0; i < n; i++)
{
   - read contents of memory under test (array[i])
   /*
    PSA will calculate CRC on each word that goes to CPU
    via Data Read Data Bus
   */
}

- deactivate Parallel Signature Analyser (PSA)

if( calculated CRC value != golden CRC)
{
      if JUMP_TO_FAILSAFE is 1
         call STL_test_fail_failSafe() *
      else
         return an error code
}

return a success code
```

Tests

This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_CRC_TEST_testRam()
- STL_CRC_TEST_testSafeRam()

6.7 Interrupt functionality test

IEC60730
Component 2.0, SW class B
Fault
Check for no interrupt fault
Measures
Periodic using functional test (H.2.16.5)
Design

```
- clear interrupt status flags
- activate all possible interrupts using software
- enable interrupt generation
/*
  at this point if interrupt occurs execution jumps to
  STL ISR
 */
 /* BEGIN STL ISR */
 /* inside STL ISR */
     if( highest priority interrupt occurs )
     {
        set the highest priority interrupt status flag
     }

     /* for interrupts with priority */
     for each interrupt
     {
        if( higher priority interrupt is serviced )
         {
             if( current ISR is fetched from expected address )
             {
                  set the corresponding interrupt status flag
             }
         }
     }

     /* for interrupts with no priority */
     for each interrupt
     {
        if( current ISR is fetched from expected address )
        {
            set the respective interrupt status flag
        }
     }
/* END STL ISR */
for each interrupt
{
     if( corresponding interrupt status flag != 1 )
     {
        if JUMP_TO_FAILSAFE is 1
            call STL_test_fail_failSafe() *
        else  return an error code
```

```
        }
    }

 return a pass code
```

Tests

This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_INTERRUPT_TEST_testInterrupt()

6.8 Stack Corruption Detection

IEC60730

Fault

Detect Stack Corruption

Measures

Design

```
/*
  This function sets up RTOS interrupt to be generated if
  stack region is corrupted
*/
- user provides
    - ISR routine to jump to when stack region is corrupted
{

    - calculate stack region areas based on compiler output
    - configure watchpoint registers to monitor stack region access
    - RTOS ISR = ISR routine

    /*
       If a Clock missing condition is detected an ISR will be
       will be issued. This ISR will be serviced by the ISR routine
       provided by the user. User decides what to do in the ISR.
    */

    if(watchpoint register set correctly)
    {
        return 1
    }
    else
    {
        return 0
    }
}
```

Tests

This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_SPC_DETECT_setUpSpcDetect()

6.9 Internal Oscillators test

IEC60730
Component 3.0, SW class B
Fault
Check Oscillator for wrong frequency operation
Measures
Periodic wrong frequency monitoring (H.2.18.10.1)
Design

```
Oscillator test using SFO:
{
    - Call SFO_MepDis_V5() function
    - Read MEP_ScaleFactor value returned

    if( MEP value is inside of bounds)
    {
        return a pass code
    }
    else
    {
        if JUMP_TO_FAILSAFE is 1
            call STL_test_fail_failSafe() *
        else
            return an error code
    }
}
```

Tests
This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_OSCILLATOR_TEST_testExtOscUsingSf0()

6.10 CPU Timers Test

IEC60730
Component 3.0, SW class B
Fault
Test accuracy of CPU timers (Indirectly detect wrong frequency)
Measures
Indirect Frequency Monitoring (H.2.10.1)
Design

```
/*
  In this function user provides a scaler and period
```

```
  such that the timer will issue an interrupt every
  1uS
*/
- user provides
   - CPU timer to test
   - timer scaler
   - period count
   - instruction based 1mS delay count

   for the CPU timer under test
   {
         - set user provided scaler and preiod
         - delay for 1mS as set by delay count
         /*
             The delay value should be such that it will take
             1mS for the delay count to reach 0 for a given
             system clock. The timer isr would be called
             approximately 1000 times within 1mS and increments
             a counter.
         */
         if(counter value is within acceptable range)
         {
             return a success code
         }
         else
         {
             return a failure code
         }
    }
```

Tests
This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_TIMER_TEST_testTimer()

6.11 Watchdog Test

IEC60730
Component 3.0, SW class B
Fault
Test accuracy of Watchdog timer (Indirectly detect wrong frequency)
Measures
Indirect Frequency Monitoring (H.2.18.10.1)
Design

```
- user provides
   - instruction based delay count
- configure watchdog to issue an interrupt
```

```
 /*
    The delay value should be such that it will take
    (512 x 128 )OSCCLK for the delay count to reach 0.
    This is the amount of time required for the watchdog
    counter to overflow and issue an  interrupt.
 */
- delay for the amount of time provided by user
if( Watchdog interrupt has occured within the specified duration)
{
   return a success code
}
else
{
    return a failure code
}
```

Tests

This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_WATCHDOG_TEST_testWatchdog()

6.12 Missing Clock Detection

IEC60730

Component 3.0, SW class B

Fault

Detect Missing Clock (Indirectly detect wrong frequency)

Measures

Indirect Frequency Monitoring (H.2.10.1)

Design

```
/*
  This function enables the clock fail detect logic
*/
{
     -enable clock fail detect by clearing MCLKOFF bit

     if(clock missing flags == 1)
     {
          return 0
     }
     else
     {
          return 1
     }
}
```

Tests

This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_CLOCK_FAIL_DETECT_enableClockFailDetect()

6.13 PLL Lock Check

IEC60730

Fault
Check if PLL has locked in
Measures

Design

```
- read PLL status register
if( PLL lock register bit == 1)
{
     return success code
}
else
{
     return a failure code
}
```

Tests
This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_PLL_LOCK_CHECK_checkPllLock()

6.14 ADC Test

IEC60730
Component 7.2.1, SW class B
Fault
Check ADC for correct conversion and muxing capability
Measures
Plausibility check on A/D Converter (H.2.18.13)
Design

```
 - user provides
    - ADC conversion count on pin A
    - ADC conversion count on pin B
    - ADC channel to sample
    - single channel selection (no-pair)
    - delay

for the selected ADC mux
{

     - configure adc to use selected mux channel

     /* delay */
    - wait until ADC conversion is complete


     if(ADC conversion is not complete)
     {
        return an error code
     }
     else
     {
         if( singleChannelSelect is 0 or 1) {
            - adcCountDelta = pinACount - pin A adc count
            if( adcCountDelta < min threshold  ||
                adcCountDelta > max threshold )
            {
                return an error code
            }
         }
         if( singleChannelSelect is 0 or 2) {
            adcCountDelta = pinBCount - pin B adc count
            if( adcCountDelta < min threshold  ||
                adcCountDelta > max threshold )
            {
                return an error code
            }
         }
     }

      return a success code
 }
```

Tests
This component is covered by the following tests. Refer to the API reference 5 for detailed

usage.

- STL_TYPE2_ADC_TEST_testAdcInput()

6.15 eCAP APWM Mode Test

IEC60730
Component 3.0, SW class B
Fault
Check eCAP in APWM mode for accurate timing
Measures
Periodic indirect wrong frequency monitoring (H.2.18.10.1)
Design

```
- user provides
  - eCAP module to be tested
  - period to duty cycle ratio
  - On time count
  - period count
  - period count compare
  - instruction based delay count

for the selected eCAP module
{
    - configure module to operate in APWM mode
    - configure module to operate in
             - provided period count
             - provided On time count
             - configure module to generate interrupt
               on period and duty cycle match
    - delay for the amount of time provided by user
     /*
        an ECAP ISR will be generated during this delay for
        both period and duty cycle match. Each count will
        be saved.
     */
     - calculate duty cycle
     - calculate period count difference

   if(calculated period count difference is not in range)
   {
         return an error code
   }
   if(calculated duty cycle is not in range)
   {
        return an error code
   }

   return an error code
}
```

```
ECAP ISR
{
    if(period flag)
    {
       - get period count
    }
    else if (duty cycle flag)
    {
       - get duty cycle count
     }
}
```

Tests

This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_TYPE0_ECAP_TEST_testEcapApwmMode()

6.16 ePWM Test

IEC60730

Component 3.0, SW class B

Fault

Check ePWM for accurate timing and functionality

Measures

Periodic indirect wrong frequency monitoring (H.2.18.10.1)

Design

```
- configure ePWM1 - ePWM4 to issue an ISR on period match
- configure ePWM1 - ePWM4 period in such a way that,
            ePWM1 = ePWM4, ePWM2 = ePWM5, ePWM3 = ePWM6,
            ePWM1 = 2*ePWM2 = 3*ePWM3
- delay for a specified time
/*
    an ISR will be generated each time a period match occurs

*/
if( (ePWM1 = ePWM4, ePWM2 = ePWM5, ePWM3 = ePWM6 )
                    and
    (ePWM1 = 2*ePWM2 = 3*ePWM3))
{
    return a success code
}
else
{
    return an error code
}
```

Tests

This component is covered by the following tests. Refer to the API reference 5 for detailed

usage.

- STL_TYPE0_EPWM_TEST_testEpwm()

6.17 GPIO Test

IEC60730
Component 7.0,7.1, SW class B,C
Fault
Check if a given GPIO pin is working properly
Measures
Plausibility Check (H.2.18.10.1) Input comparison (H.2.18.8)
Design

```
GPIO Input Check:

  - user provides
      - GPIO pin number to be tested and expected value
      - GPIO register access delay
  if( GPIO pin number is invalid)
  {
      return an error code
  }

  for the selected pin
  {
      - configure pin as input
      - delay
      if(read value on pin == expected value)
      {
           return success code
      }
      else
      {
           return an error code
      }
  }

GPIO Output Check :

  - user provides
      - GPIO pin number to be tested
      - GPIO register access delay
  if( GPIO pin number is invalid)
  {
      return an error code
  }

  for the selected pin
  {
      - configure pin as output
```

```
        - delay
        - read pin data value = original pin value
        - toggle the pin
        - read pin data value = new pin value
        if(original pin value == new pin value)
        {
            return an error code
        }
        - read pin data value = original pin value
        - toggle the pin
        - read pin data value = new pin value
        if(original pin value != new pin value)
        {
            return an error code
        }

        return success code
    }
```

Tests

This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_GPIO_TEST_testGpioInput()
- STL_GPIO_TEST_testGpioOutput()

6.18 Communication Module Loopback Test

IEC60730

Component 6.3, SW class B

Fault

Check Comparator for correct functionality

Measures

Time Slot Monitoring (H.2.18.10.4) Logical Mmonitoring (H.2.18.10.2)

Design

```
    - configure module to operate in loopback mode

    for(i = 0; i < number of test words; i++)
    {
        - transmit word[i]
        - delay for a specified time equal to the bit rate
        if(receive flag is not set within the specified time)
        {
            return an error code
        }
        else
        {
            if(word[i] != received word)
            {
                return an error code
```

```
            }
        }
    }

    return a success code
```

Tests

This component is covered by the following tests. Refer to the API reference 5 for detailed usage.

- STL_TYPE1_SPI_TEST_testSpiLoopback()
- STL_TYPE0_I2C_TEST_testI2cLoopback()
- STL_TYPE0_SCI_TEST_testSciLoopback()
- STL_TYPE0_ECAN_TEST_testeCanLoopback()

7 Revision History

```
IEC60730_STL_Library
Version 4.00.01.00

-------------------------------------------------------------------
Bug 4740
-------------------------------------------------------------------
Summary          : March C - algorithm in v4.00.00.00 doesn't build

Bug in           : STL_march_test.asm

Severity         : normal

Affected module  : RAM

Impact           : Unable to build and use RAM test algorithm, if
                   March C option is selected with v4.00.00.00
                   release.

Description      : March C- RAM test option doesn't compile and fails
                   as is. This is an optional alternative to the more
                   intensive March 13N RAM test algorithm.
                   Due to the wrong op-code usage in the March C- RAM
                   test algorithm in v4.00.00.00, users wouldn't be
                   able to build and link the library if March C- RAM
                   test is selected. Users can use both March C - RAM
                   algorithm in v4.00.01.00.

Workaround       : The issue is fixed in the 4.00.01.00.

-------------------------------------------------------------------
Bug 4741
-------------------------------------------------------------------
Summary          : Add a device identifier for F280200

Bug in           : STL_user_config.h

Severity         : normal

Affected module  : N/A

Impact           : Unable to build safety library for F280200 with
                   v4.00.00.00 release.

Description      : A device identifier MACRO for F280200 device was
                   left out in v4.00.00.00.
                   The addition of the device identifier MACRO, will
                   enable customers to build a library for the
                   F280200 device. Users needed to add the identifier
                   MACRO themselves in order to build safety library
                   for the F280200 device in v4.00.00.00.
```

```
                  The device identifier MACRO is now added as part
                  of the safety library in v4.00.01.00.

Workaround       : The issue is fixed in the 4.00.01.00.

-----------------------------------------------------------------
Bug 5681
-----------------------------------------------------------------
Summary          : Modify Appendix G in v4.00.00.00

Bug in           : User's Guide documentation

Severity         : normal

Affected module  : N/A

Impact           : N/A

Description      : The use of Path and Build variables is the preferred
                   method of linking source files in CCSv5 projects.
                   The documentation in Appendix G in v4.00.00.00 outlines
                   the older way of linking source files using macro.ini file.
                   Appendix G of the user guide in v4.00.01.00 is modified to
                   show the use of path and build variables instead of
                   macro.ini.

Workaround       : The issue is fixed in the 4.00.01.00.

-----------------------------------------------------------------
Bug 5811
-----------------------------------------------------------------
Summary          : Correct Table 2.1 in v4.00.00.00

Bug in           : User's Guide documentation

Severity         : normal

Affected module  : N/A

Impact           : N/A

Description      : Table 2.1 of the User guide lists the wrong definition
                   number under the "IEC60730 test definition used" column
                   for Variable Memory Test in v4.00.00.00. The listed
                   definition in v4.00.00.00 is H.2.19.16.2.
                   The definition number is corrected in v4.00.01.00 to
                   H.2.19.6.2.

Workaround       : The issue is fixed in the 4.00.01.00.


IEC60730_STL_Library
```

```
Version 4.00.00.00

----------------------------------------------------------------
Bug N/A
----------------------------------------------------------------
Summary          : Initial release. No bug listed

Bug in           : N/A

Severity         : N/A

Affected module  : N/A

Description      : N/A

Workaround       : N/A
```

Appendices

A PSA CRC

A.1 PSA CRC

Parallel Serial Analyzer (PSA) is a module in c28x devices that can be used to generate a 40 bit CRC on a given memory region. The PSA polynomial is $Y = x^{40} + x^{21} + x^{19} + x^2 + 1$. The PSA calculates CRC values by monitoring Data Read Data Bus (DRDB). Once activated to monitor DRDB, when a CPU reads data via DRDB, the PSA will generate a CRC for each data in DRDB on each clock cycle. Figure. A.1 shows the relation between CPU , memory and PSA

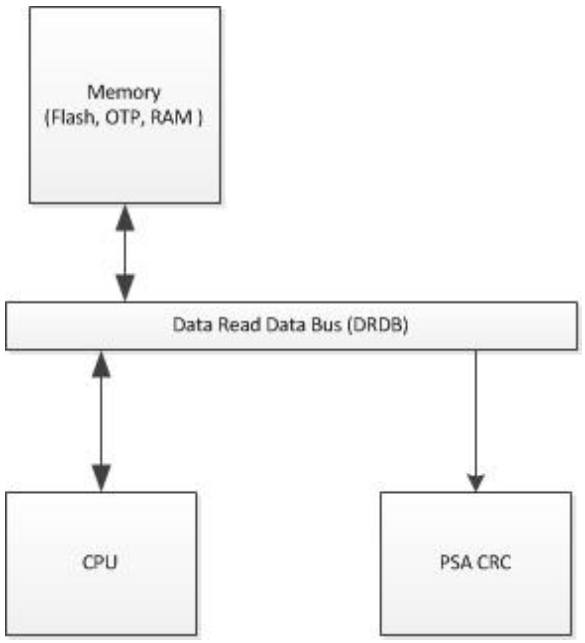

Figure A.1: PSA CPU and memory relation

The following table lists the registers that are used to access PSA in CRC generation

Register Address	R/W	Usage
0x0840	R/W	contains the Upper 8 bits of the 40 bit CRC result.
0x0842	R/W	contains the Lower 16 bits of the 40 bit CRC result.
0x0843	R/W	contains the Middle 16 bits of the 40 bit CRC result.
0x0846	R/W	PSA control register. write 0x0001 to claim DRDB. write 0x0012 to enable PSA to access DRDB. write 0x0000 to release PSA from accessing DRDB.

Table A.1: PSA register description

B Hardware watch points

B.1 Hardware watch points

The IEC60730 safety library uses hardware watchpoints to monitor stack corruption and to trigger an interrupt when a corruption occurs. The hardware watchpoint uses reference addresses and masks to qualify the memory address that will , when accessed , trigger an interrupt. The interrupt that is triggered is an RTOS interrupt. The watchpoint uses the following logic to qualify a memory address and issue an RTOS interrupt.

```
if((reference_address | mask) == (cpu_acceessed_memory | mask))
{
          issue RTOS interrrupt
}
```

Memory mapped registers are used to specify the reference address and mask. The value of the mask must be an interger value of $2^N - 1$, where N = 1,2,.... and the vaue of the refernce address that is coded in the reference address register must have a value of the reference address ORed with the mask. Once these values are provided and the watch point enabled, the watchpoint issues an RTOS interrupt when the above qualifier event occurs. In the simple example provided with IEC60730 safety library package, the stack has an inclusive range of [0x50 - 0x24F]. The memory range that is watched for stack corruption is 64. The reference address register is set to 0x23F ((0x250 - 64) | 0x3F) and the mask is set to 0x3F. With this setting the watchpoint issues an RTOS ISR when the monitored area is accessed as shown in figure B.1 below. Refer to SPRA820 for more information.

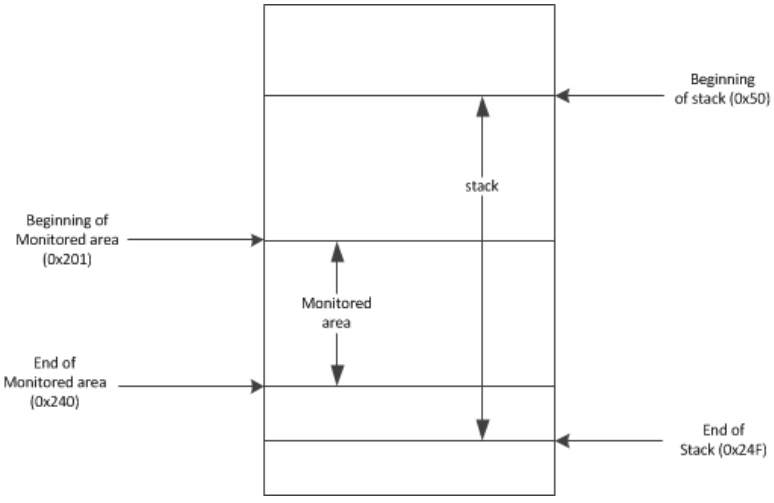

Figure B.1: Stack corruption detection

C Safe RAM areas

C.1 Safe RAM areas

The IEC60730 safety library relies on a number of functions that are run from RAM. These RAM areas including the stack are refered to as **safe RAM areas**/**regions** in this documentation. When a destructive RAM test is performed on the *safe RAM areas*, the safety library automatically saves and restores their contents. The user should avoid writing to the safe RAM areas. **Note: The user is responsible for saving and restoring code section that is run from RAM and could corrupted during MARCH test.**
The following table lists the safe RAM areas.

safe RAM	**Memory Range**
BootROM rserved	0x0 - 0x4F
stack	user specific, compiler generates labels _stack to _STACK_END
psa_crc code	linker specific linker generates labels _PSA_CRCRunStart to _PSA_CRCRunEnd
PIE vector	0xD00 - 0xDFF
PC test function	0xAAA8 to 0xAAAB

Table C.1: safe RAM regions

D Simple Test Application

D.1 Simple Test Application

This section briefly describes the steps required to run the Simple test application that ships with the IEC60730 safety library.

D.1.1 Control Card Connections and Pin Configurations

The IEC60730 safety library ships with a simple application project that calls the safety library APIs periodically and reports the status via SCI-A (UART) port. Make the following connections to log data to a PC using a hyperterminal , PUTTY or any other serial communication application.

1. Open a serial communication application (hyperterminal or PUTTY).
2. Set the settings to: 9600 baud rate, 1 stop bit, no parity and 8 bit character.
3. Connect the Standard B and the Standard A end of the USB cable to the control Card and to a PC respectively.
4. To power up the control card using USB, turn the SW1 switch on the docking station towards the "USB" label. if using external power supply turn SW1 towards the "ON" label.

Note: See the respective control card documentaiton if using an ISO control card.

The application was tested and run on F28335 controlCard. Make the following pin voltage level changes before running the application.

1. Connect ADC input A2 and B7 to a 2.86 volt supply.
2. Connect ADC input A7 and B2 to a 1.257 volt supply.
3. Voltage at pin GPIO 59 and 62 must be low for GPIO input tests.
4. Voltage at pin GPIO 63 and 6 must be high for GPIO input tests.
5. Leave pins GPIO 60, 61 and 10 unconnected. These pins are used for AIO and GPIO output tests.

Note:

D.1.2 Invariable Memory CRC check consideration

The simple app calls a function that checks a CRC checksum on all invariable memory. This includes flash and OTP. The content of the OTP could be different for different silicon. Do the following steps if there is a CRC checksum error or if any part of the application or IEC60730 STL library code is modified.

1. Import and rebuild the simple app example.
2. Run the app with emulator with **no breakpoints** for atleast 1 minute (to ensure the STL_generateCrc() function is run.

3. The newly generated CRC values will be populated in gStructCrcResult structure (in main.c). Use watch window to see the newly calculated CRC values. The golden CRC values passed on to the CRC check are defined in gGoldenCRC array (in STL_TEST_REPORT.c). Copy the newly populated CRC values from gStructCrcResult to the corresponding array cell in gGoldenCRC.

D.1.3 Fail Case Tests

The simple demo application example and the safety test library ,where possible, provide a provision for introducing error to test the safety tetst library. This is done in addition to the test during code development. The fail case test can be enabled by setting the macros LIB_TEST_ISR_ERROR , LIB_TEST_MARCH_ERROR_ASC, LIB_TEST_MARCH_ERROR_DSC, LIB_TEST_FPU_TEST_ERROR, LIB_TEST_CPU_TEST_ERROR, LIB_TEST_EPWM_TEST_ERROR defined in STL_system_config.h in Safety test library , LIB_TEST_ECAP_APWM_MODE_ERROR, LIB_TEST_OSCILLATOR_ERROR, LIB_TEST_SPI_ERROR, LIB_TEST_SCI_ERROR, LIB_TEST_I2C_ERROR, LIB_TEST_CAN_ERROR, LIB_TEST_WATCHDOG_ERROR, LIB_TEST_TIMERS_0_1_ERROR, LIB_TEST_TIMER_2_ERROR in STL_test_report.h in simple demo application and the variable gErrorTestFlag declared in main.h in the simple demo application.
Note: These macros and the variable gErrorTestFlag should be set for test purpose only.

D.1.4 Simple Application Test Flow chart

The following flow chart shows how the test application is organized in the simple application.

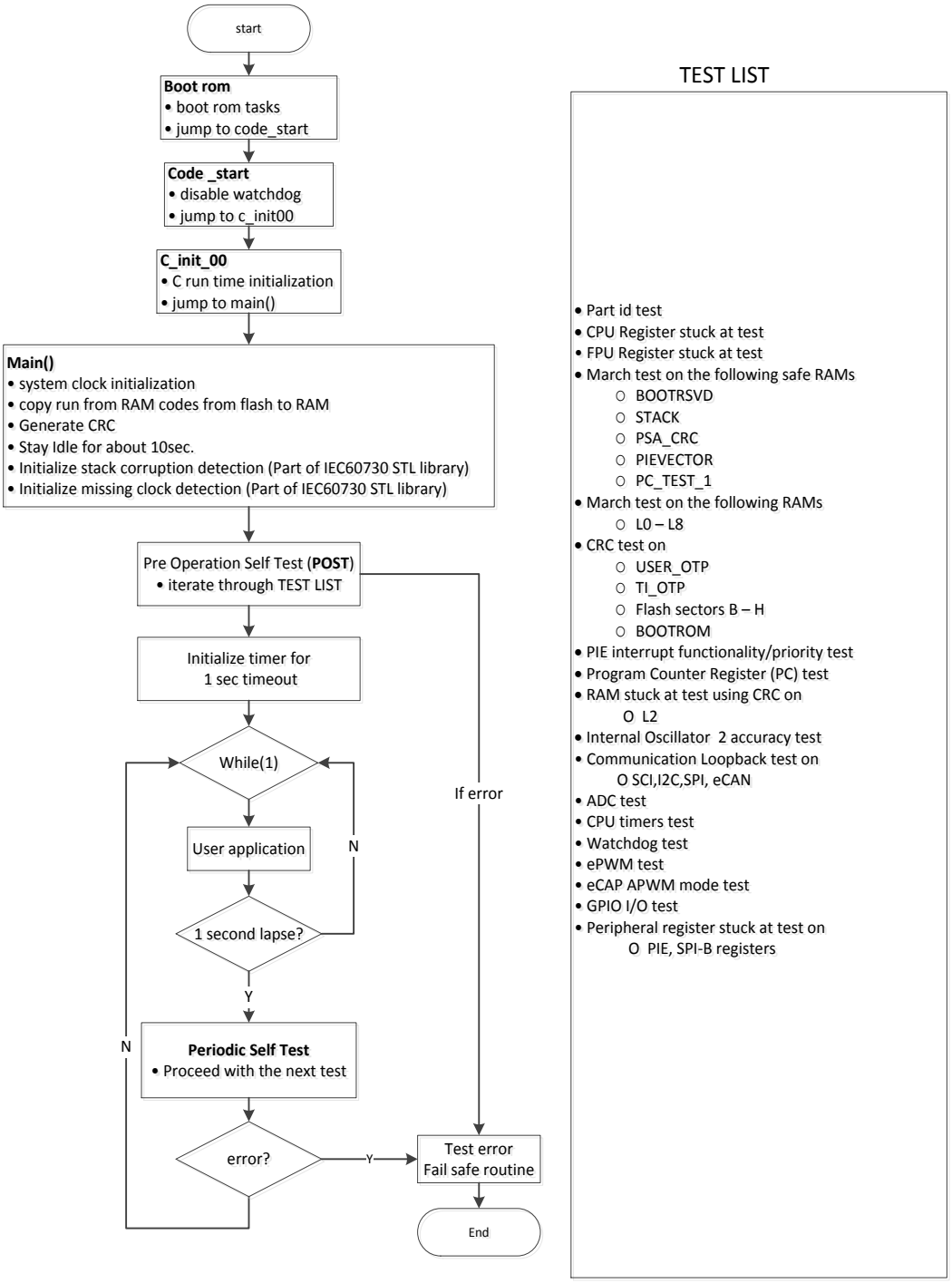

Figure D.1: Test Application Flow Chart

E Sample Test Report

E.1 Sample Test Report

The following Test reports were generated during the IEC60730 API library testing using a simple application under normal and fail case tests.

Part id identified
CPU register test passed
FPU register test passed
reserved boot RAM march test passed
stack RAM march test passed
PSA crc code RAM march test passed
PIE Vector RAM march test passed
PC register test function 1 RAM march test passed
L0 RAM march test passed
L1 RAM march test passed
L2 RAM march test passed
L3 RAM march test passed
L4 RAM march test passed
L5 RAM march test passed
L6 RAM march test passed
L7 RAM march test passed
User OTP CRC test passed
TI OTP 1 CRC test passed
Flash sector H CRC test passed
Flash sector G CRC test passed
Flash sector F CRC test passed
Flash sector E CRC test passed
Flash sector D CRC test passed
Flash sector C CRC test passed
Flash sector B CRC test passed
Boot ROM test passed
Interrupt functionality test passed
Prorgam counter functionality test passed
L2 RAM CRC test passed
Watchdog test passed
Timer 1 test passed
Timer 2 test passed
Timer 0 test passed
GPIO 60 output test passed
GPIO 61 output test passed
GPIO 10 output test passed
SPI A loopback test passed
SCI A loopback test passed
SCI B loopback test passed
SCI C loopback test passed
GPIO 59 input test passed
GPIO 63 input test passed
GPIO 62 input test passed
GPIO 9 input test passed
I2C loopback test passed
eCAP 5 APWM mode test passed
eCAP 2 APWM mode test passed
External Oscillator test passed
External input mux 2 ADC test passed
External input mux 7 ADC test passed
stack RAM CRC test passed
reserved boot RAM CRC test passed
eCAN B loopback test passed
ePWM test passed
PIE registers stuck at test passed
SPI A registers stuck at test passed

Figure E.1: Sample test report

Wrong part id
CPU register test failed
FPU register test failed
L5 RAM march test failed
VCU register test failed
reserved boot RAM march test failed
stack RAM march test failed
PSA crc code RAM march test failed
PIE Vector RAM march test failed
PC register test function 1 RAM march test failed
L0 RAM march test failed
L1 RAM march test failed
L2 RAM march test failed
L4 RAM march test failed
L6 RAM march test failed
L7 RAM march test failed
L8 RAM march test failed
User OTP CRC test passed
TI OTP 1 CRC test passed
TI OTP 2 CRC test passed
Flash sector H CRC test passed
Flash sector G CRC test passed
Flash sector F CRC test passed
Flash sector E CRC test passed
Flash sector D CRC test passed
Flash sector C CRC test failed
Flash sector B CRC test passed
Boot ROM test passed
eCAP 1 APWM mode test passed
eCAP 2 APWM mode test failed
eCAP 3 APWM mode test passed
Interrupt functionality test failed
Prorgam counter functionality test passed
L1 RAM CRC test passed
Internal Oscillator 1 test failed
Internal Oscillator 2 test failed
SPI A loopback test failed
I2C loopback test failed
SCI A loopback test failed
eCAN loopback test failed
Comparator 1 test passed
Comparator 2 test passed
AIO 12 input test passed
AIO 6 input failed
GPIO 10 input test passed
GPIO 58 input test passed
Internal ADC test passed
External input mux 2 ADC test passed
External input mux 7 ADC test passed
Watchdog test failed
Timer 1 test failed
Timer 2 test failed
Timer 0 test failed
GPIO 6 output test passed
GPIO 33 input test passed
AIO 14 input passed
ePWM test failed
PIE registers stuck at test passed
SPI B registers stuck at test passed
CLA configuration register test passed
CLA CPU to CLA RAM march test failed
CLA to CPU RAM march test passed
CLA Program Counter test passed
CLA execution registers test passed
CLA Functionality test passed
stack RAM CRC test passed
reserved boot RAM CRC test passed

Figure E.2: Sample fail test report

F Porting the Library

F.1 Porting the Library

A minor change to the user and system configuration header files is required when porting the IEC60730 safety library to other F2833x device variants. The required changes are outlined below. **Note: In addition to the changes listed below, make sure to follow the external dependencies listed in section 4.2.4.**

1. Set the BUILD_LIB_F2833X macro to 1 in STL_user_config.h. **Make sure the other build macros are set to 0.**
2. Set the appropriate device id macro to 1 in STL_user_config.h. For instance if porting to F28334, set the "DEVICE_TYPE_28334" macro to 1. **Make sure that only a single device id macro is set.**
3. Add a **#if DEVICE_TYPE_XXXXX #endif** directive in STL_system_config.h where XXXXX is the device name. For instance, if porting to F28068, add the following two lines.
   #if DEVICE_TYPE_28334

   #endif
   Device specific attributes will be defined inside the above directive. The table below lists the macros that need to be defined inside the above directive. The macro name should be as exactly as defined in table F.1.

Macros	Description
BOOT_RSVD_START_ADDRESS	Boot ROM reserved RAM, usually this address is 0x0
BOOT_RSVD_END_ADDRESS	Boot ROM reserved RAM, usually this address is 0x4F
PIE_V_START_ADDRESS	PIE vector start address, usually this address is 0xD00
PIE_V_END_ADDRESS	PIE vector start address, usually this address is 0xDFF
PC_TEST_1_START_ADDRESS	Start address for PC register test section as defined by the linker command file.
PC_TEST_1_END_ADDRESS	End address for PC register test section as defined by the linker command file.
RAMM1_END_ADDRESS	RAM M1 end address, usually this address is 0x7FF
RAML0_START_ADDRESS	RAM L0 start address, usually this address is 0x8000
RAML_END_ADDRESS	RAM L end address.
FLASH_START_ADDRESS	Flash start address excluding flash sector A.
FLASH_END_ADDRESS	Flash end address excluding flash sector A.
USER_OTP_START_ADDRESS	User OTP start address.
USER_OTP_END_ADDRESS	User OTP end address.
TI_OTP_START_ADDRESS	TI OTP start address which contains calibration data and Get mode function.
TI_OTP_END_ADDRESS	TI OTP end address which contains calibration data and Get mode function.
BOOTROM_START_ADDRESS	Bootrom start address
BOOTROM_END_ADDRESS	Bootrom end address
MAXIMUM_GPIO_VALUE	Maximum number of valid GPIO mux pins
MAXIMUM_GPIOA_VALUE	Maximum number of valid GPIO A mux pins
MAXIMUM_GPIOB_VALUE	Maximum number of valid GPIO A mux pins
VALID_GPIOA_PIN_MASK	GPIO A pin mask.Each bit represents a pin. If a bit is set to 1, the pin corresponding to that bit does exist and is valid. MSB corresponds to pin 31 LSB corresponds to pin 0
VALID_GPIOB_PIN_MASK	look the previous row for GPIO A
VALID_GPIOC_PIN_MASK	look the previous row for GPIO A
MAXIMUM_SPI_MODULES	Number of available SPI modules
MAXIMUM_ECAP_MODULES	Number of available eCAP modules
USE_ZONE_0_MEMORY	Flag to include Zone 0 Memory
ZONE_0_START_ADDRESS	Zone 0 start address
ZONE_0_END_ADDRESS	Zone 0 end address

Table F.1: System Configuration Macros

G Building Static Library

G.1 Building Static Library

Once the appropriate changes are made to the STL_user_config.h and STL_system_config.h header files, follow the procedures below to create and build a new static library project.

1. Create a folder in "../IEC60730_safety/v4_00_01_00/projects/f2833x/" directory, for instance if building for F28334,create a folder named f28334.**"../IEC60730_safety/v4_00_01_00/projects/f2833x/f28334"**
2. Open CCSv5 > click on **Project** Menu > click on **New CCS Project**
3. Make the following changes listed in the **New CCS Project window** .See figure G.1.
   (a) Project name —type in the appropriate project name
   (b) Output type —select **Static Library**
   (c) Location —browse and select the directory where the project folder is placed. For instance for F28334, the directory will be **"../IEC60730_safety/v4_00_01_00/projects/f2833x/f28334"**
   (d) Family —select **C2000**
   (e) Variant —select **2833x Delfino**, select the specific device, for instance for F28334 select **TMS320F28334**
   (f) Project templates and examples —select **Empty Project** under **Empty Projects** option.
   (g) Click **Finish**

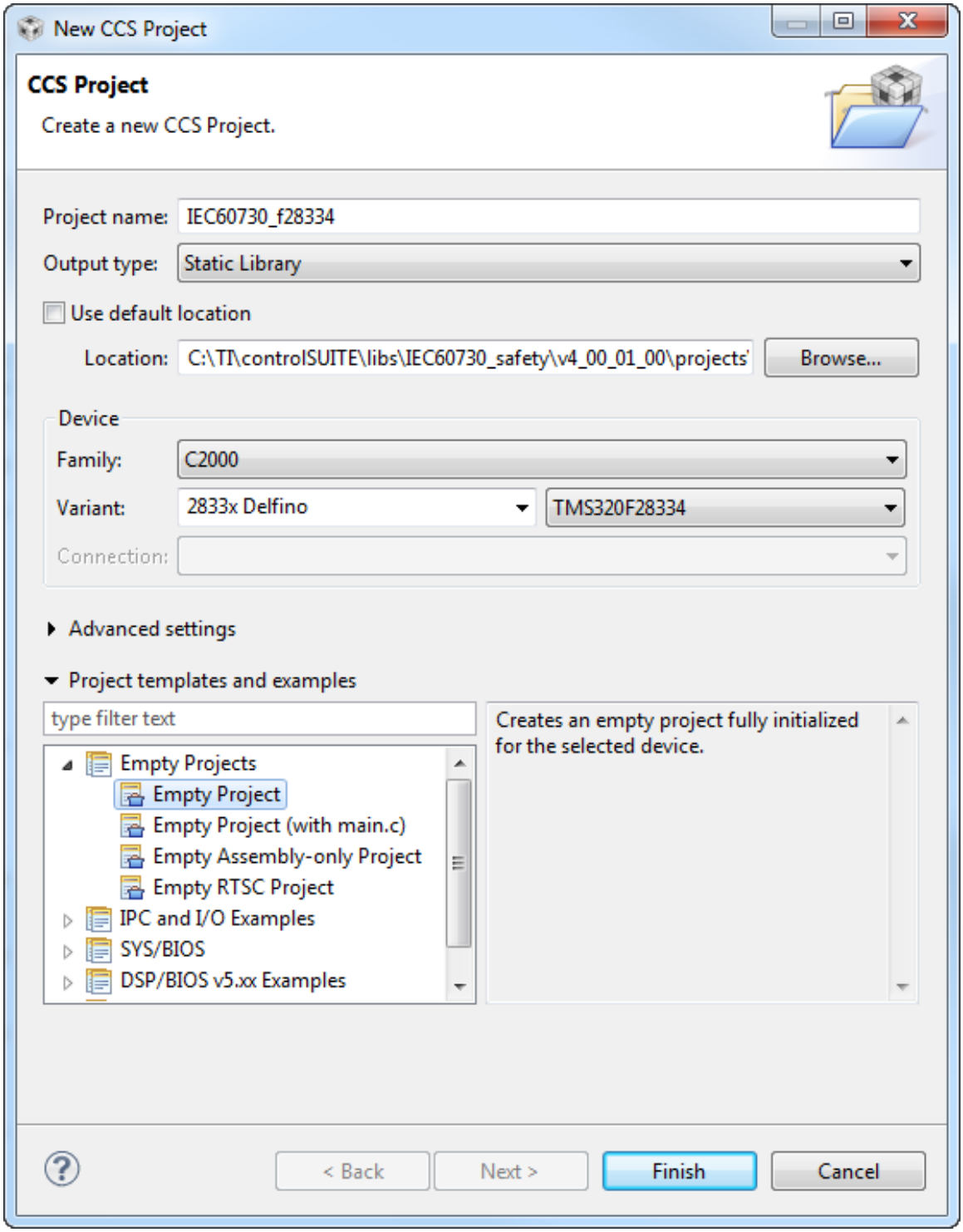

Figure G.1: Project Settings

4. Add path variable. Click on **Projects** menu > Click on **Properties**. In the "Properties" window, highlight **Linked Resources** under **Resources**. In the **Path Variables** tab, > Click on **New**. In the "New Variable" pop up window,

(a) enter **INSTALLROOT_CONTROLSUITE** in the Name text box

(b) enter **${PROJECT_LOC}/../../../../../..** in the Location text box

> Click **OK** > Click **OK**.

5. Add build variable. Click on **Projects** menu > Click on **Properties**. In the "Properties" window, highlight **Build**. In the **Variables** tab, > Click on **Add**. In the "Define A New Build Variable" pop up window,

   (a) enter **INSTALLROOT_CONTROLSUITE** in the Variable Name text box

   (b) Select "Directory" as Type.

   (c) enter **${PROJECT_ROOT}/../../../../../..** in the Value text box

   > Click **OK** > Click **OK**.

6. Add source files. > Right click on the Project name inside the **Project Explorer** window. Select **New** > **File** in the **New file** window > Click on **Advanced**. Check **Link to file in the system**. > Click **Variables**. In the **Select Path Variable** window, > Click on **INSTALROOT_CONTROLSUITE** under **Name**. Click **Extend**. This will pop up a window explorer showing the contents of controlSUITE. Browse to libs/IEC60730_safety/v4_00_01_00/source/c28x and select the *.C and *.asm source files required for the current build > Click **ok** > Click **Finish**. This process will link one source file at a time to the current project. Repeat this for all the required source files.

7. Add the search path to the version header files required by the source files. Click on **Projects** menu > click on **Properties** > expand to **Build** > **C2000 Compiler** > **Include Options**. Add the following in the **Add dir to #include search path** window. See figure G.2. For example for this library build v133 of the header files and v4_00_01_00 of the safety library was used.

   (a) **"${INSTALLROOT_CONTROLSUITE}/libs/IEC60730_safety/v4_00_01_00/include/shared"**

   (b) **"${INSTALLROOT_CONTROLSUITE}/libs/IEC60730_safety/v4_00_01_00/include/c28x"**

   (c) **"${INSTALLROOT_CONTROLSUITE}/device_support/f2833x/v133/DSP2833x_common/include"**

   (d) **"${INSTALLROOT_CONTROLSUITE}/device_support/f2833x/v133/DSP2833x_headers/include"**

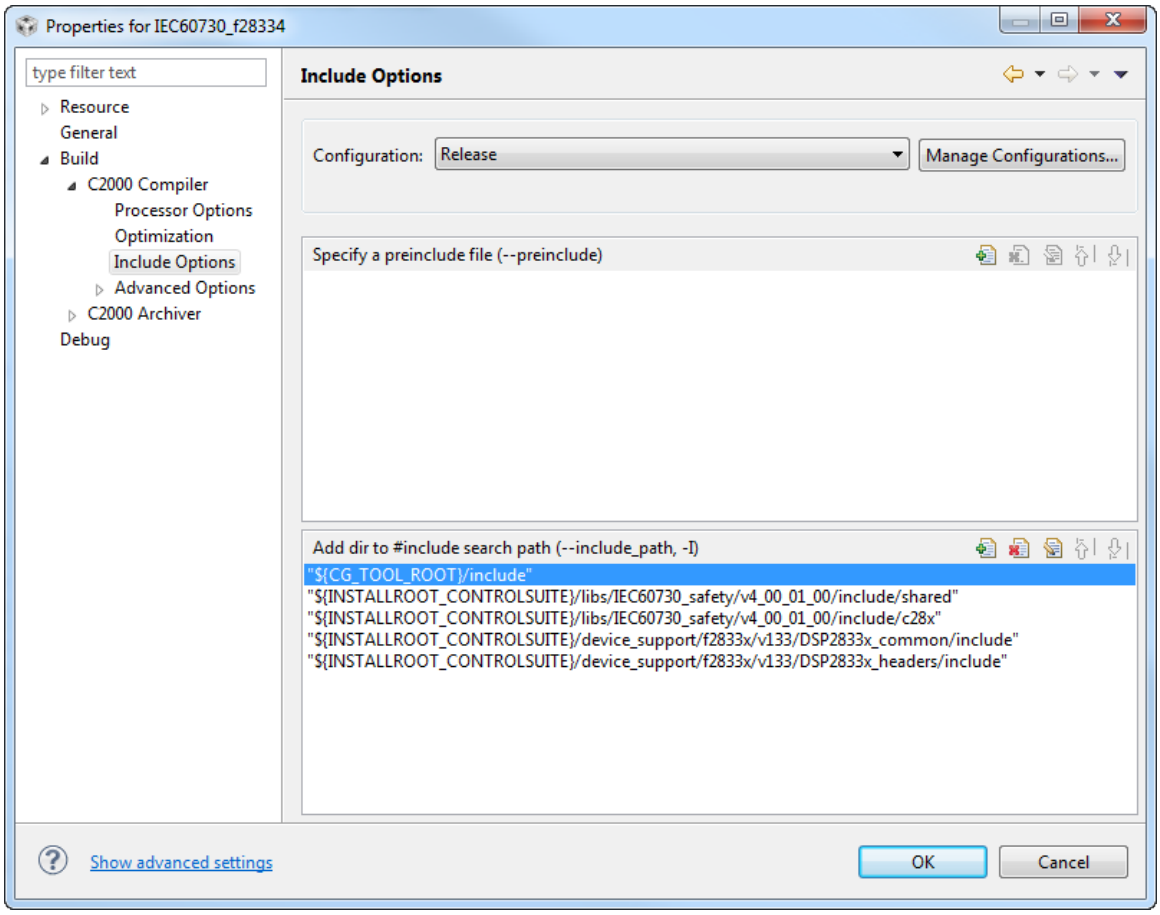

Figure G.2: Adding Header File Directory

8. Click on **Projects** menu > click on **Properties** > expand to **Build** > **C2000 Archiver** > **Basic Options**. In the **Basic Options** in **Output file** text box type in the name of the library and the directory to place the library. For instance if building for f28334, enter **"${INSTALLROOT_CONTROLSUITE}/libs/IEC60730_safety/v4_00_01_00/lib/IEC60730_F28334_STL.lib"** > click **OK**. This will set the name of the library and the directory it will be placed after it is built.See figure G.3.

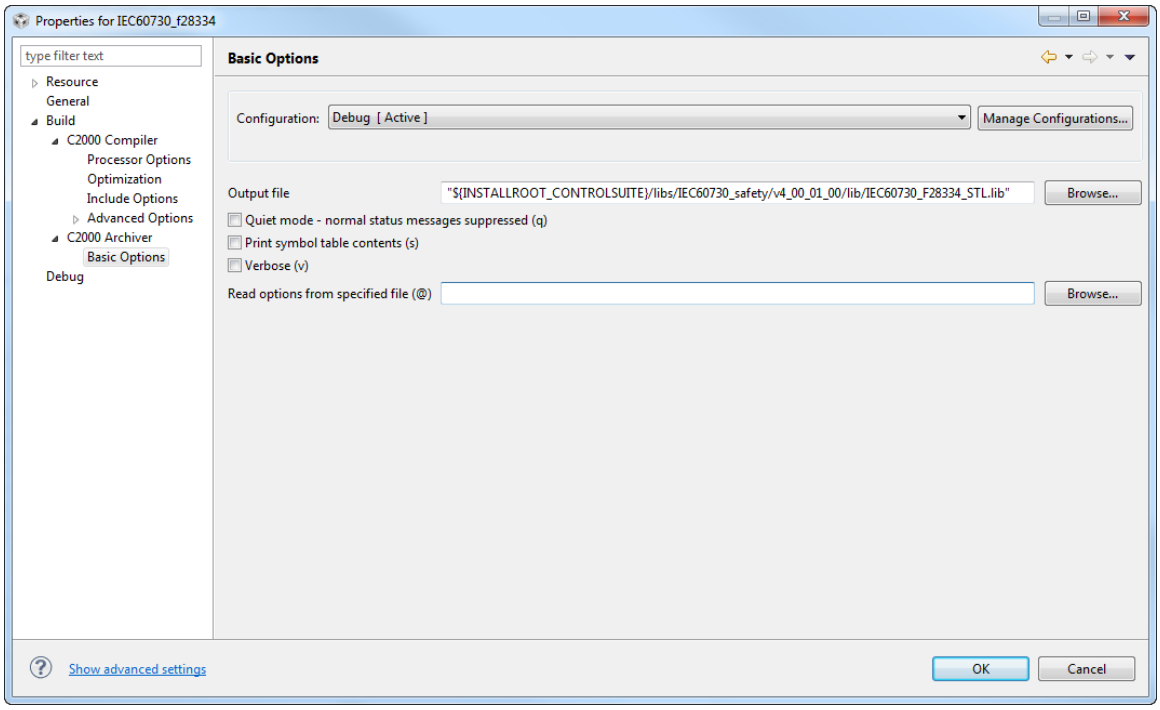

Figure G.3: Setting Library Output name and Directory

9. Finally build the project. Click on **Projects** > click on **Build Project**. The built library will be named and placed according in the directory specified in step 8 above.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (https:www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

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