

---

**How to run self-test in the SPC58NN84xx device**

---

**Introduction**

This application note describes how to run the logic and memory BISTs (L/MBIST) implemented in the SPC58NN84xx device.

The user typically executes these self-tests during the startup phase. They are transparent for the application.

Last generations of microcontroller (55 and 40 nm), including the SPC58NN84xx, offer a high flexibility in the self-test configuration.

The user can:

- choose at which frequency self-test runs
- decide if BIST runs in parallel or sequential mode
- split the self-tests in different subsets:
  - during key-on (such as self-test in off-line mode)
  - during key-off (such as self-test in on-line mode)

The possibility to split the self-tests guarantees a faster boot time and at the same time maintains a high diagnostic coverage against latent failures.

A complete example of self-tests working in both off-line and on-line modes is available.

This application note focuses on SPC58NN84xx, but most concepts are valid also for other 40 and 55 nm devices.

# Contents

- 1 Overview ..... 6**
  - 1.1 Off-line mode configuration ..... 7
  - 1.2 On-line mode configuration ..... 7
  - 1.3 Modules used in self-test phase ..... 8
  - 1.4 L/MBIST scheduling activity ..... 10
  - 1.5 Clock configurations of self-test .....11
    - 1.5.1 Clocking in off-line mode ..... 11
    - 1.5.2 Clocking in on-line mode ..... 12
- 2 STCU registers list ..... 14**
- 3 Functional reset sequence in off-line mode ..... 17**
- 4 LBIST and MBIST partitions ..... 18**
  - 4.1 MBIST partition ..... 18
  - 4.2 LBIST partition ..... 21
  - 4.3 DCF records to program in off-line mode ..... 24
    - 4.3.1 Self-test in off-line mode ..... 25
- 5 On-line test ..... 27**
  - 5.1 Self-test flow ..... 27
  - 5.2 MEMU results ..... 28
  - 5.3 On-line configuration ..... 28
- 6 How to read STCU register results and possible reactions ..... 30**
  - 6.1 MBIST reaction ..... 30
  - 6.2 LBIST reaction ..... 31
- 7 Summary ..... 33**
- Appendix A FCCU reaction ..... 34**
- Appendix B Reference code ..... 38**
  - B.1 Document reference ..... 43

B.2 Acronyms ..... 43

**Revision history ..... 44**

## List of tables

Table 1.	Example of MBIST settings	11
Table 2.	List of STCU on/off line registers	14
Table 3.	MBIST partition	18
Table 4.	LBIST partition	21
Table 5.	Acronyms	43
Table 6.	Document revision history	44

## List of figures

Figure 1.	Basic single LBIST architecture . . . . .	6
Figure 2.	Module interconnections in off-line mode . . . . .	8
Figure 3.	Module interconnections in on-line mode . . . . .	10
Figure 4.	Clock tree for M/LBIST execution . . . . .	12
Figure 5.	Shift and capture frequencies . . . . .	13
Figure 6.	Destructive and functional reset sequence in case off-line L/MBISTs are enabled . . . . .	17
Figure 7.	Sequence of DCF records in the UTEST . . . . .	25
Figure 8.	Self test flow in on-line mode . . . . .	28
Figure 9.	MBIST reaction scheme . . . . .	30
Figure 10.	LBIST reaction scheme . . . . .	31
Figure 11.	Behavior of the FCCU in case NO fault is detected by LBIST . . . . .	36
Figure 12.	Behavior of the FCCU in case at least a fault is detected by LBIST. . . . .	37

# 1 Overview

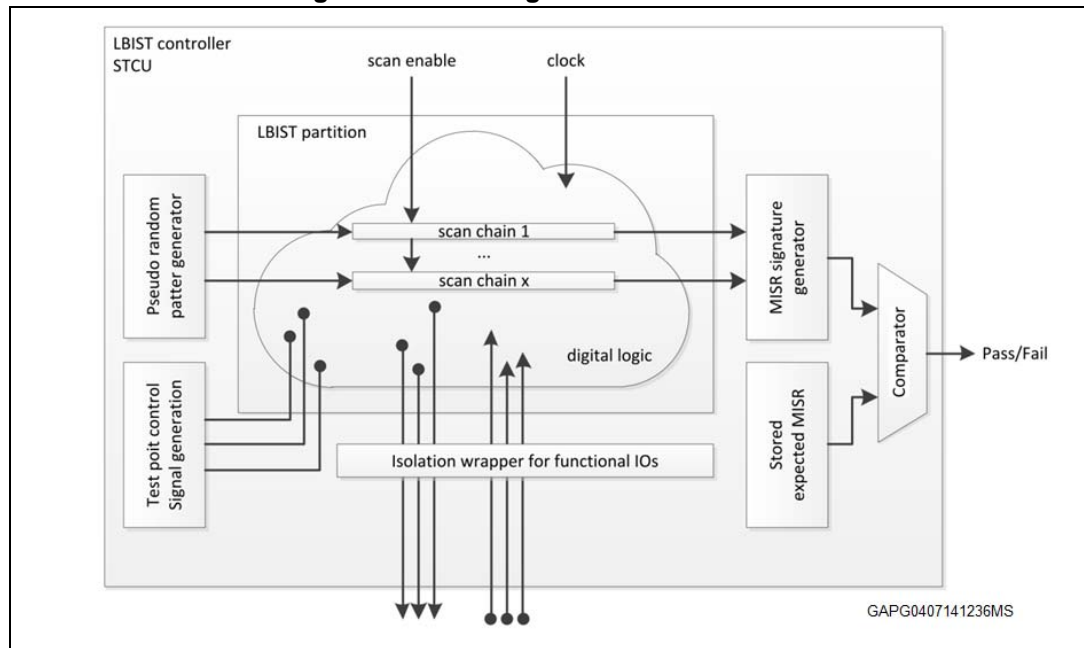
BISTs (Built-In Self-Tests) are periodic tests that detect latent faults in the functional logic of the device according to the ISO26262 standard.

The SPC58NN84xx device includes two different types of BIST:

- memory BIST (MBIST): for all volatile memories<sup>(a)</sup>
- logic BIST (LBIST): for digital logic

Figure 1 shows a basic schema of the LBIST architecture.

**Figure 1. Basic single LBIST architecture**



Logic blocks are divided in LBIST partitions. LBIST partitions are tested by the LBIST controller. It consists of a series of chains that are scanned by a pseudo random pattern generator. A hardware comparator compares the generated signatures (MISR) versus an expected one.

a. MBIST verifies the integrity of the array of the volatile memory only. For non volatile memories another type of test is in place, such as Array Integrity Check. Have a look at the reference manual to have all details on this test.

Running all LBIST/MBIST at the power-on takes a certain amount of time. If this time is too long to satisfy the user needs, a "mixed" methodology can be used: for example splitting the L/MBISTs in two groups of self-tests<sup>(b)</sup>:

- the first group is executed in off-line mode, during the key on phase (off-line self-test phase)
- the second one is executed in on-line mode, during the key off phase (on-line self-test phase)

The Self-Test Control Unit (STCU) hardware module controls the BIST execution for both modes, such as on-line and off-line.

STCU has two sets of registers that collect the L/MBIST results separately for the off-line and on-line modes. Moreover a programmable watchdog timer (WDG), inside STCU peripheral, checks that the self-test operations (both LBIST and MBIST) have been completed within the assigned time slot.

## 1.1 Off-line mode configuration

The users configure STCU according to their needs. The SSCM loads this configuration during the boot phase from UTest sector of the flash (through DCF records).

User saves the right configuration in the UTest sectors.

The main parameters necessary to execute the off-line self-test are:

- L/MBIST scheduling activity
- LBIST setup (MISR expected values included)
- Unrecoverable/Recoverable faults management
- CRC
- PLL management

They are discussed in more detail in the next paragraphs.

## 1.2 On-line mode configuration

The user can run the BIST on-line. For example a possible strategy is to run this test before the key off event.

Software configures the STCU and triggers the BIST execution. Afterward the software can save the result of the self-test in the NVM (Flash) before shutdown. Application can read this result after the next key-on.

For a comprehensive description of the STCU, please refer to the STCU's dedicated chapter of SPC58NN84xx reference manual.

---

b. This is an example. Other methodologies can be thought.

The main parameters necessary to execute the on-line self-test are:

- L/MBIST scheduling activity
- LBIST setup (MISR expected values included)
- Unrecoverable/Recoverable faults management
- CRC

A functional reset must be issued to the entire SoC by hardware at the completion of the self-test procedure<sup>(c)</sup>. STCU module is not affected by a functional reset. This gives the possibility to read the self-test results after that.

### 1.3 Modules used in self-test phase

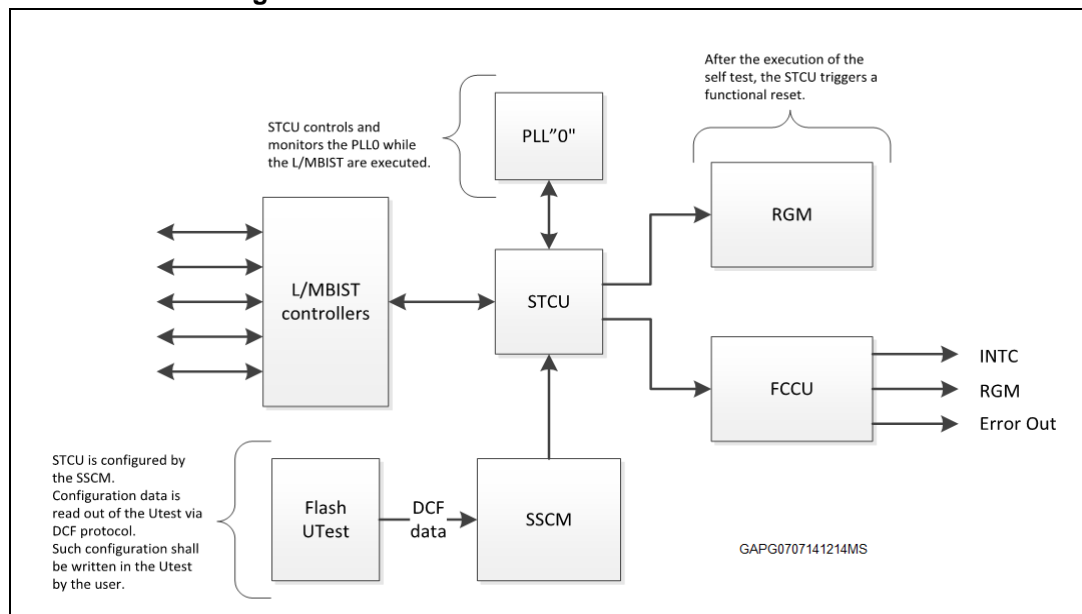
There are several hardware modules involved in the self-test execution. Their logical interconnection is different according to the self-test mode (either off-line or on-line mode).

In off-line mode, the modules involved are:

- the Reset Generation Module (RGM)
- L/MIST controllers
- the System Status and Configuration Module (SSCM)
- the Self-Test Control Unit (STCU2)
- the Fault Collection and Control Unit (FCCU)
- UTEST Flash

Figure 2 shows the interconnection of these modules.

Figure 2. Module interconnections in off-line mode



c. In order to enable the automatic generation of the functional reset after the execution of the on-line LBIST, the flag LBRMSWx of the STCU\_LBRMSW reset management register needs to be set '1'.



The self-test is executed at the start-up by the STCU whose configuration has been loaded by the SSCM from UTest sector.

STCU is not directly connected to the memories, but to the MBIST/LBIST controllers.

To run an MBIST, the STCU programs the MBIST controller. The BIST execution is monitored by memory collars.

The off-line self-test is driven by PLL0 which is directly controlled and configured by the STCU.

Any LBIST/MBIST failure event can be managed by FCCU<sup>(d)</sup> (see [Section Appendix A](#) for further clarifications). The STCU error lines (unrecoverable failures/recoverable failures) are linked to this module.

FCCU alerts the external environment in different modes: by an interrupt, by reset or reporting the error by external lines.

STCU is also connected to the RGM. After the LBIST execution a functional reset is triggered.

In on-line mode the interconnection scheme is different. In this case the involved modules are<sup>(e)</sup>:

- the Reset Generation Module (RGM)
- the Self-Test Control Unit (STCU2)
- the Fault Collection and Control Unit (FCCU)
- L/MBIST controllers
- PLL"1"

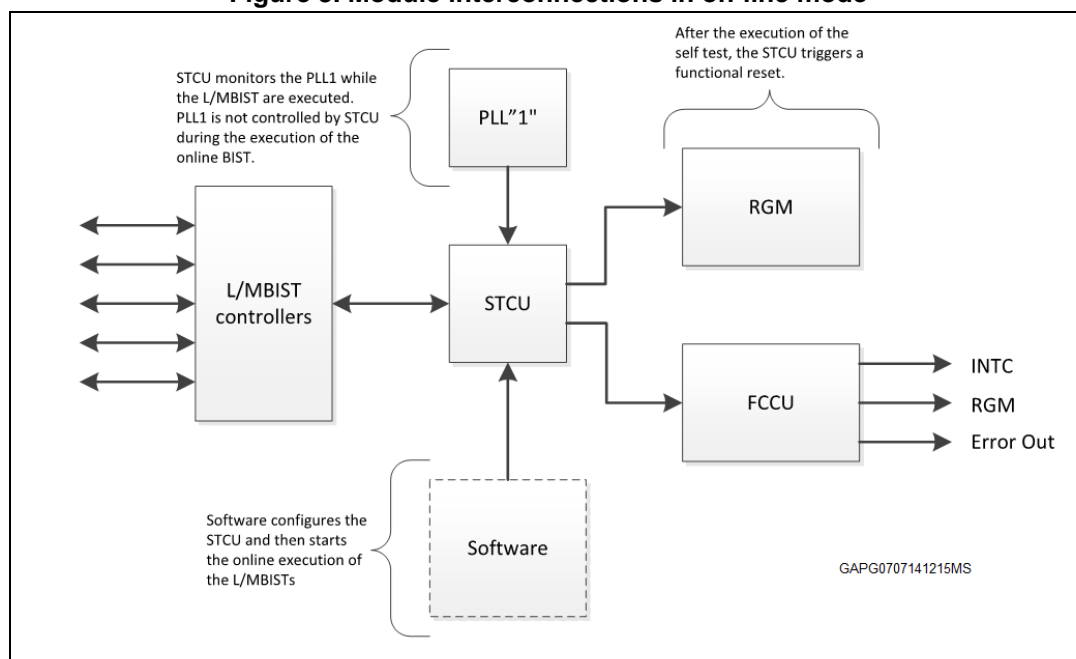
[Figure 3](#) shows the modules involved.

---

d. On SPC58NN84xx device only Recoverable/Not Critical fault condition can be managed.

e. Another module not directly connected to the STCU but involved in the self-test, is the MEMU. The MEMU collects information about single bit and double bit events in the memories.

**Figure 3. Module interconnections in on-line mode**



The on-line self-test is initiated by the software code which provides the proper commands to the STCU via IPS interface.

Connection between STCU and the memories and MBIST controller are the same as the off-line self-test.

In on-line mode, the action of STCU is limited to PLL lock signal monitoring. In this case, reference clock for self-test execution is the system clock (the same used by the application).

Either PLL0 or PLL1 can be used to drive the LBIST/MBIST (in off-line mode only PLL0 can be used). [Figure 3](#) assumes PLL1 has been selected. Additional details on clocking configuration and constraints can be found in [Section 1.5](#).

STCU is connected to RGM and FCCU modules to report errors.

## 1.4 L/MBIST scheduling activity

STCU2 is very flexible in terms of scheduling L/MBIST. It allows programming the concurrent<sup>(f)</sup> or sequential execution of the MBISTs or LBISTs.

Although the concurrent execution is shorter in terms of time, it causes higher power dissipation.

The mechanism used to provide this flexibility is a linked list where the starting pointer is the bitfield STCU\_CFG[PTR].

f. Also referenced as parallel.

The additional pointers are in the bitfield STCU\_LB\_CTRLx[PTR] for LBISTx (where x is the selected LBIST) and STCU\_MB\_CTRLy[PTR] for MBIST (where y is the selected MBIST) and have to be filled depending on the selected sequence of run.

The additional bitfield STCU\_LB\_CTRL[CSM] and STCU\_MB\_CTRL[CSM] provides the flexibility to run the chosen set of the LBISTs or MBISTs either concurrently or sequentially or to close the linked list by setting the NIL pointer.

Next example shows how these registers have to be configured in case the first 3 MBISTs are configured to run in parallel:

**Table 1. Example of MBIST settings**

STCU_CFG = 0x10000101	PTR = 10h-> MBIST_0
	CLK_CFG = 0x101 -> It isn't the self-test frequency, but related to the memory controller frequency (in this case is sys_clk/6)
STCU_MB_CTRL_0 = 0x11800000	PTR = 11h-> MBIST_1
	CSM = 1 (concurrent/parallel mode)
STCU_MB_CTRL_1 = 0x12800000	PTR = 12h-> MBIST_2
	CSM = 1 (concurrent/parallel mode)
STCU_MB_CTRL_2 = 0xFF000000	PTR = FFh-> pointer to NIL. No BIST execution
	CSM = 0 (last MBIST)

In case of LBISTs runs, it needs to map 00h in STCU\_CFG register (first LBIST). Next LBISTs will be mapped in STCU\_LB\_CTRLx.

## 1.5 Clock configurations of self-test

According to the selected mode, the LBIST/MBIST can run with different clock configurations.

### 1.5.1 Clocking in off-line mode

In off-line mode, MBIST can run up to 200 MHz and LBIST up to 100 MHz driven by the PLL0.

STCU is able to control the PLL0 during the off-line self-test operations depending on the status of the STCU\_RUN[MBPLEN] and STCU\_RUN[LBPLEN].

By these flags, off-line MBIST/LBIST are executed enabling the on-chip PLL control interface selecting the parameters defined into STCU\_PLL\_CFG register (PLLODF, PLLIDF and PLLDF).

Below the formula of PLL0 frequency output:

$$f_{PLLout} = \frac{f_{VCO}}{PLLODF \times 2}$$

where

$$f_{VCO} = f_{PLLin} \times (PLLDF \times 2) = \left( \frac{f_{RC}}{PLLIDF} \right) \times (PLLDF \times 2)$$

$f_{RC}$  is the frequency of internal oscillator that is 16 MHz.  $f_{PLLin}$  should be  $\geq 8$  MHz. Alternately, it is possible to use also the internal RCOSC (16 MHz).

### 1.5.2 Clocking in on-line mode

It is important to notice that there are two main frequencies involved in the STCU and self-tests:

- STCU clock is the frequency of the peripheral bridge clock and represents the working frequency of the STCU itself
- L/MIST clock: represents the clock used by the L/MBISTs

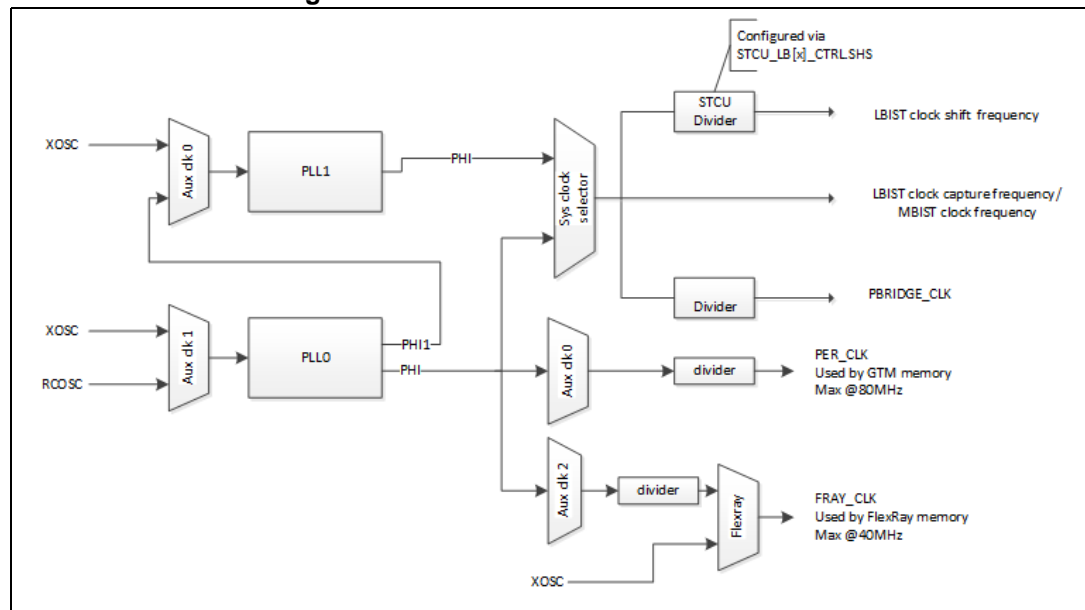
Concerning the second clock, the reference clock for M/LBIST is the system clock.

Having a glance at the SPC58NN84xx clock tree in its RM:

- the system clock (and then the running frequency of the self-test) is the frequency to the output of the System Clock Selector
- the STCU frequency is given by the peripheral A bridge clock (that is PRIDGEA\_CLK in [Figure 4](#)).

[Figure 4](#) underlines the main selectors involved in self-test configuration.

**Figure 4. Clock tree for M/LBIST execution**



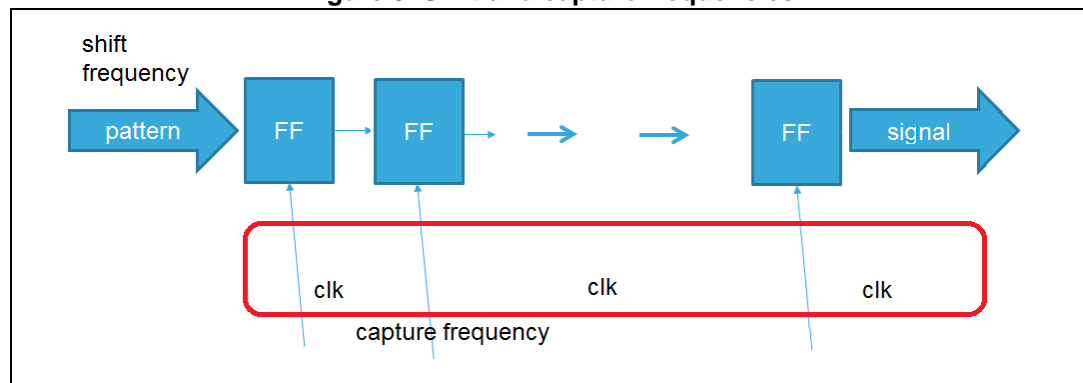
Going into detail, the MBISTs run at system clock frequency, while the LBISTs run with the following:

- capture frequency
  - it is the system clock frequency
- shift frequency
  - it is the system clock frequency divided by STCU\_LB[x]\_CTRL.SHS parameter (where 'x' is the LBIST under analysis) = `bist_clk`

A LBIST partition consists in a chain formed by a series of flip flop connected back to back. The output of one flip flop (FF) is connected to another.

Shift and capture frequencies are represented in [Figure 5](#).

**Figure 5. Shift and capture frequencies**



LBISTs are guaranteed @100 MHz for LBIST0 and 50 MHz for the other partitions.

Both PLL0 and PLL1 can be used to run the on-line L/MBIST, although the typical configuration is to use the PLL1 driven by PLL0 as shown in [Figure 4](#).

There are some constraints related to MBIST execution regarding the GTM and FlexRay modules:

- GTM memory works up to 80 MHz
- FlexRay memory works up to 40 MHz

## 2 STCU registers list

STCU offers two sets of independent registers to collect the L/MBIST results. One set is used during the off-line mode and the other during the on-line mode.

To increase the safety capability of this module, a different security key sequence is also required during off/on-line self-test in order to unlock the write access to the relevant STCU registers.

Next table summarizes which STCU registers are dedicated to on-line test and STCU registers dedicated to off-line test.

**Table 2. List of STCU on/off line registers**

N.	Off-line register	On-line register	Description
1	STCU_RUN	STCU_RUN_SW	Run Register. Start STCU.
2	STCU_SCK Off-line 0xD3FEA98Bh:key1-0x2C015674h:key2 On-line 0x753F924Eh: key1 - 0x8AC06DB1h: key2		SK Code Register
3	STCU_CFG		Configuration Register. Global configuration of the STCU2.
4	STCU_PLL_CFG		PLL Configuration Register. It defines the parameters used to program the PLL only when the off-line L/MBIST are performed and STCU_RUN[MBPLEN] = 1 or STCU_RUN[LBPLEN] = 1
5	STCU_WDG		Watchdog Register Granularity. It defines the time budget of LBIST and MBIST.
6		STCU_INT_FLG	Interrupt Flag Register. Only when the related control bit into the STCU_RUNSW register (MBIE and LBIE) are enabled.
7	STCU_CRCE		Expected Status Register. It includes the expected signature of the CRC-32 (Ethernet protocol)
8	STCU_CRCR		Read Status Register. It reports the value obtained at the end of the off/on-line self-test sequence.
9	STCU_ERR_STAT		Error Register. It includes the status flags related to the STCU2 internal error conditions occurred during the configuration or the on/off-line self-testing execution.
10	STCU_ERR_FM		Error FM Register. It defines the fault mapping of the STCU2 faults described into the register. STCU_ERR_STAT in terms of unrecoverable or recoverable fault.

Table 2. List of STCU on/off line registers (continued)

N.	Off-line register	On-line register	Description
11	STCU_LBS	STCU_LBSSW	LBIST Status Register. It includes the results corresponding to the execution of the selected off/on-line LBIST. 0: failed LBIST execution 1: successful LBIST execution
12	STCU_LBE	STCU_LBESW	LBIST End Flag Register. It includes the end flag related to the execution of the selected off-line LBIST. 0: LBIST execution not yet completed 1: LBIST execution finished
13		STCU_LBRMSW	LBIST Reset Management. It defines the on-line reset mapping for each LBIST.
14	STCU_LBUFM		It defines the fault mapping of each LBIST in terms of unrecoverable or recoverable fault
15	STCU_MBSL	STCU_MBSLSW	Status Low Register (from 0 to 31 MBIST). It includes the results corresponding to the execution of the selected off/on-line MBIST.
16	STCU_MBSM	STCU_MBSMSW	Status Medium Register (from 32 to 63 MBIST). It includes the results corresponding to the execution of the selected off/on-line MBIST.
17	STCU_MBSH	STCU_MBSHSW	Status High Register (from 64 to 95 MBIST). It includes the results corresponding to the execution of the selected off/on-line MBIST.
18	STCU_MBEL	STCU_MBELSW	End Flag Low Register (from 0 to 31). It includes the end flag related to the execution of the selected off/on-line MBIST.
19	STCU_MBEM	STCU_MBEMSW	End Flag Medium Register (from 32 to 63). It includes the end flag related to the execution of the selected off/on-line MBIST.
20	STCU_MBEH	STCU_MBEHSW	End Flag High Register (from 64 to 95). It includes the end flag related to the execution of the selected off/on-line MBIST.
21	STCU_MBUFML		Unrecoverable FM Low Register (from 0 to 31). It defines the fault mapping, in terms of unrecoverable or recoverable fault of the MBIST.
22	STCU_MBUFMM		Unrecoverable FM Medium Register (from 0 to 31). It defines the fault mapping, in terms of unrecoverable or recoverable fault of the MBIST.
23	STCU_MBUFMH		Unrecoverable FM High Register (from 0 to 31). It defines the fault mapping, in terms of unrecoverable or recoverable fault, of the MBIST.

**Table 2. List of STCU on/off line registers (continued)**

N.	Off-line register	On-line register	Description
24	STCU_LB_CTRL		LBIST Control Register. It defines the control setting of each LBIST controller.
25	STCU_LB_PCS		LBIST PC Stop Register. It defines the pattern counter stop of each LBIST controller.
26	STCU_LB_PRPGL		LBIST PRPG Low Register. It defines the LSB part (31..0) of the PRPG start value of the LBIST controller.
27	STCU_LB_PRPGH		LBIST PRPG High Register. It defines the MSB part (32..63) of the PRPG start value of the LBIST controller.
28	STCU_LB_MISREL	STCU_LB_MISRELSW	MISR Expected Low Register. It defines the LSB part (31..0) of the expected MISR of the off/on-line LBIST controller.
29	STCU_LB_MISREH	STCU_LB_MISREH_SW	MISR Expected High Register. It defines the MSB part (32..63) of the expected MISR of the off/on-line LBIST controller.
30	STCU_LB_MISRRL	STCU_LB_MISRRLSW	MISR Read Low Register. It reports the LSB part (31..0) of the MISR obtained at the end of the off/on-line LBIST controller execution.
31	STCU_LB_MISRRH	STCU_LB_MISRRHSW	MISR Read High Register. It reports the MSB part (32..63) of the MISR obtained at the end of the off/on-line LBIST controller execution.
32	STCU_MB_CTRL		MBIST Control Register. It defines the control setting of MBIST controller.



### 3 Functional reset sequence in off-line mode

After a LBIST execution, a functional reset is automatically generated. [Figure 6](#) shows the reset generation sequence.

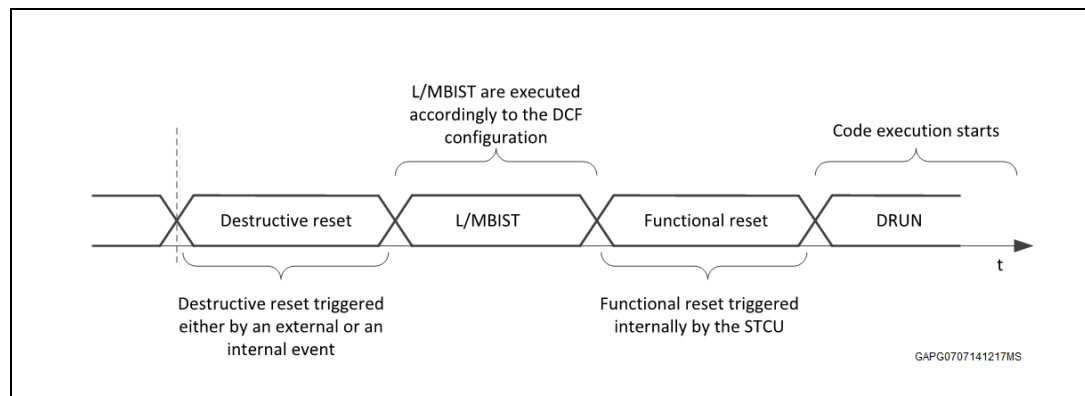
The sequence starts with a destructive reset that is triggered by an external or internal event. After the L/MBIST execution, a functional reset is automatically generated by STCU. Finally the device is ready to execute the code in DRUN mode.

During the MBIST/LBIST execution:

- a dedicated watchdog is configured to monitor self-test execution completion within expected time. The length of this phase is variable depending on the self-test requirements.
- the Self-Test Control Unit (STCU) executes all tests specified by the DCF records
- the self-test engine updates the self-test completion flag and triggers an associated functional reset on completion of self-test. In case of error, a fault is forwarded to the FCCU.

The length of this phase is variable depending on the self-test requirements.

**Figure 6. Destructive and functional reset sequence in case off-line L/MBISTs are enabled**



In case of on-line mode there is no the need of a destructive reset before L/MBIST execution. A functional reset needs to be triggered after execution of the self-test. Note that an off-line self-test will not run after this functional reset according to what was previously explained (see [Figure 6](#)).

For more details refer to Reset and Boot chapter of SPC58NN84xx reference manual (see [Section B.1: Document reference](#)).

## 4 LBIST and MBIST partitions

In terms of self-test, SPC58NN84xx is composed by 92 MBIST memory partitions (from #0 to #91) and 7 LBIST logical partitions (from #0 to #6). Detailed lists are shown in the next paragraph.

### 4.1 MBIST partition

**Table 3. MBIST partition**

Processor / Module	Memory Instance Name	MBIST partition
Core 0	ram_zxa_imem	21
	ram_zxa_dmem_0_0	25
	ram_zxa_dmem_0_1	24
	ram_zxa_dmem_1_0	23
	ram_zxa_dmem_1_1	22
	ram_zxa_ichache_0	29
	ram_zxa_ichache_1	28
	ram_zxa_ichache_2	27
	ram_zxa_ichache_3	26
	ram_zxa_dcachache_0	34
	ram_zxa_dcachache_1	33
	ram_zxa_dcachache_2	32
	ram_zxa_dcachache_3	31
	ram_zxa_itag	30
	ram_zxa_dtag	35

Table 3. MBIST partition (continued)

Processor / Module	Memory Instance Name	MBIST partition
Core 1	ram_zxb_imem	53
	ram_zxb_dmem_0_0	57
	ram_zxb_dmem_0_1	56
	ram_zxb_dmem_1_0	55
	ram_zxb_dmem_1_1	54
	ram_zxb_ichache_0	61
	ram_zxb_ichache_1	60
	ram_zxb_ichache_2	59
	ram_zxb_ichache_3	58
	ram_zxb_dcachache_0	66
	ram_zxb_dcachache_1	65
	ram_zxb_dcachache_2	64
	ram_zxb_dcachache_3	63
	ram_zxb_itag	62
	ram_zxb_dtag	67
Core_2 (IOP)	ram_zxc_imem	3
	ram_zxc_dmem_0_0	7
	ram_zxc_dmem_0_1	6
	ram_zxc_dmem_1_0	5
	ram_zxc_dmem_1_1	4
	ram_zxc_ichache_0	11
	ram_zxc_ichache_1	10
	ram_zxc_ichache_2	9
	ram_zxc_ichache_3	8
	ram_zxc_dcachache_0	18
	ram_zxc_dcachache_1	17
	ram_zxc_dcachache_2	16
	ram_zxc_dcachache_3	15
	ram_zxc_itag	12
	ram_zxc_dtag	14

Table 3. MBIST partition (continued)

Processor / Module	Memory Instance Name	MBIST partition
HSM	hsm_pram_0	36
	hsm_pram_1	37
	ram_hsm_icache_0	39
	ram_hsm_icache_1	38
	ram_hsm_itag_0	41
	ram_hsm_itag_1	40
	ram_hsm_c3	42
SYSTEM RAM 0	ram_sram_0_0	52
	ram_sram_0_1	51
SYSTEM RAM 2	ram_sram_2_0	50
	ram_sram_2_1	49
SYN Ethernet (AVB 4.20)	ram_dwc_eth0_tx_odd	45
	ram_dwc_eth0_tx_even	46
	ram_dwc_eth0_rx_odd	47
	ram_dwc_eth0_rx_even	48
Calibration	ram_overlay_0_0	70
	ram_overlay_0_1	69
	ram_overlay_1_0	91
	ram_overlay_1_1	90
DMA	ram_dma_0	68
	ram_dma_1	13
FLEXRAY	ram_flexray_dram_0	43
	ram_flexray_lram_0	44
CAN	ram_can_0	19
	ram_can_1	20

Table 3. MBIST partition (continued)

Processor / Module	Memory Instance Name	MBIST partition
GTM	ram_gtm_fifo	76
	ram_gtm_ram0_0	79
	ram_gtm_ram0_1	78
	ram_gtm_ram0_2	77
	ram_gtm_ram0_3	81
	ram_gtm_ram0_4	80
	ram_gtm_ram1_0	84
	ram_gtm_ram1_1	83
	ram_gtm_ram1_2	82
	ram_gtm_ram1_3	86
	ram_gtm_ram1_4	85
	ram_gtm_dpll_ram1a	87
	ram_gtm_dpll_ram1b	88
	ram_gtm_dpll_ram2	89
NEXUS	ram_nexus_nar_0	75
	ram_nexus_nar_1	74
	ram_nexus_nar_2	73
	ram_nexus_nar_3	72
AMU	ram_amu	71
ST FLASH	ram_mpflash	2
ROM	rom_bar	0
	rom_mpflash	1

## 4.2 LBIST partition

Table 4. LBIST partition

LBIST0	LBIST1	LBIST2	LBIST3	LBIST4	LBIST5	LBIST6
XBAR	gtm	hsm	cansubsys_1	psi5_1	platform	cansubsys_0
XBIC		flash_0	flexray_0	ethernet_0		dma_ch_mux_1_checker
AHB		sent_0	memu	adcsar_dig_2		dma_ch_mux_0_checker
AIC_0_1_2		sent_1	emios_1	linflex_0		dma_ch_mux_2_checker
DMC		emios_0	psi5_s_0	adcsar_dig_3		dma_ch_mux_3_checker

Table 4. LBIST partition (continued)

LBIST0	LBIST1	LBIST2	LBIST3	LBIST4	LBIST5	LBIST6
AIPS_0_1_2		psi5_0	fccu	adcsar_seq_1		dma_ch_mux_4_checker
PFLASH		dspi_4	ethernet_1	adcsar_seq_2		dma_ch_mux_5_checker
PRAM		dspi_5	adcsd_ana_1	adcsar_seq_b		dma_ch_mux_0
RCCU			adcsd_ana_3	adcsar_seq_0		dma_ch_mux_1
CPU checker			adcsd_ana_2	adcsar_seq_3		dma_ch_mux_2
			adcsd_ana_5	iic_0		dma_ch_mux_3
			adcsd_ana_4	dspi_0		dma_ch_mux_4
			adcsd_ana_0	dspi_9		dma_ch_mux_5
			crc_0	dspi_1		
			crc_1	dspi_2		
			body_ctu_0	dspi_3		
			fosu	dspi_8		
			cmu_2_hpbm	dspi_7		
			cmu_13_pfbridge	dspi_6		
			cmu_7_sent	adcsar_dig_1		
			cmu_11_fbridge	linflex_13		
			bam	linflex_10		
			adcsd_dig_2	linflex_4		
			adcsd_dig_4	linflex_6		
			adcsd_dig_5	linflex_1		
			adcsd_dig_1	linflex_14		
			adcsd_dig_0	linflex_15		
			adcsd_dig_3	linflex_16		
				linflex_17		
				linflex_7		
				linflex_5		
				linflex_3		
				linflex_9		
				linflex_12		
				linflex_11		

Table 4. LBIST partition (continued)

LBIST0	LBIST1	LBIST2	LBIST3	LBIST4	LBIST5	LBIST6
				linflex_2		
				linflex_8		
				adcsar_dig_0		
				adcsar10_dig_0		
				adcsar10_dig_1		
				pit_rti_0		
				fr_rom_1		
				fr_rom_0		
				ips_read_mux_2		
				adcsar_10bit_seq_0		
				adcsar_10bit_seq_1		
				ima		
				ips_read_mux_1		
				intg_ima_readmux_top		
				pit_rti_1		
				cmu_0_pll0_xoc_ir_cosc		
				cmu_4_gtm		
				cmu_12_emios		
				cmu_10_psi5_1us		
				cmu_5_sdadc		
				cmu_3_pbridge		
				cmu_9_psi5_f125		
				cmu_8_psi5_f189		
				cmu_6_saradc		
				cmu_1_core_xbar_bd		
				adcsar_ana_wrap_2		
				adcsar_ana_wrap_3		
				adcsar_ana_wrap_0		
				adcsar_ana_wrap_1		

**Table 4. LBIST partition (continued)**

LBIST0	LBIST1	LBIST2	LBIST3	LBIST4	LBIST5	LBIST6
				adcsar_ana_wrap_b		
				adcsar_ana_wrap_comp_0		
				adcsar_ana_wrap_comp_1		
				adcsar_bias		
				adcsd_bias		
				adcsar_dig_b		

Off-line self-test is performed during the boot phases (see [Table 2](#))

### 4.3 DCF records to program in off-line mode

DCF records are used to configure certain registers in the device during system boot while the reset signal is asserted.

There are two categories of DCF:

- **TEST DCF records**  
They are programmed into TEST flash during production with some information about trimming parameters (for example trip points for voltage comparators, adjusting analog to digital voltage supplies parameters, trimming oscillator parameters, frequencies and enabling RAM repair). This memory is not visible to the user who cannot modify its content.
- **UTEST DCF records**  
Some UTEST DCF records are written by the factory and programmed during production testing. Others are written by the end user. The user-defined DCF records are used for several configurations, for example configuring memory blocks as OTP, protecting flash blocks with specific password groups and define which, and how, self-tests are executed.

A DCF record consists of 2 of 32 bit contiguous words: a pointer to the location of a register<sup>(g)</sup> and the data to be written to such a register.

First 32 bits contain the data information. The other 32 bits contain information about chip select, address, start and stop bit. All details about DCF and how it needs to be used can be found at the section "Device Configuration Format (DCF) Records" of the RM.

In SPC58NN84xx device, UTEST memory is located between 0x00400300 and 0x00400FFF.

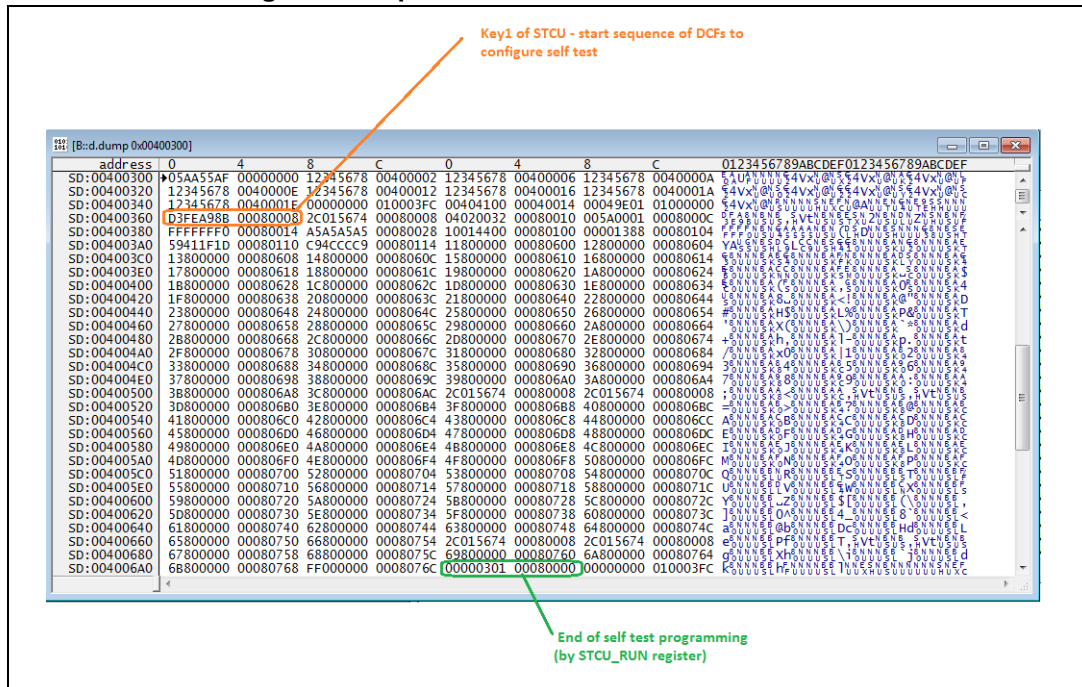
[Figure 7](#) shows the whole DCF sequence in order to start self-test in off-line mode.

This sequence describes the starting of LBIST0 + all MBIST partitions of the device.

g. This word also includes additional configuration.



Figure 7. Sequence of DCF records in the UTEST



- from 0x00400300 to 0x00400358 there are DCF programmed by the factory
- from 0x00400360 there are DCF dedicated for self-test (see the orange rectangle)
- self- test ends at 0x004006B0
- last DCF is a dummy DCF used to complete a writing in the UTEST at 128 bit

Next paragraph describes a complete example of self-test that includes MBIST and LBIST tests.

### 4.3.1 Self-test in off-line mode

After unlocking the STCU, some important registers have to be configured in order to run the test.

Hereafter the most important:

- Watchdog timer (WDG): STCU\_WDG register  
 It enables to check if the self-test operations (both LBIST and MBIST) have been completed within the assigned time slot or the STCU\_RUN or BYPASS bits have been programmed before watchdog time-out. The datasheet reports the execution time of L/MBISTs.  
 The watchdog timeout must to be higher than to the sum of the execution times of all selected BISTs.  
 In order to estimate the execution time, user needs to consider that off-line self-test can be driven either by the IRCOSC or PLL0.  
 In case RCOSC is used, the only parameter to configure is related to STCU\_CFG[CLK\_CFG]. These bits define the ratio between the sys\_clk and the TCK used to program both the LBIST and the MBIST and the STCU core clock.

- Fault mapping of the STCU faults: STCU\_ERR\_FM register  
It defines if a fault is considered either as unrecoverable or recoverable.
- Pointer to the first LBIST or MBIST should be scheduled: STCU\_CFG[PTR].  
The methodology of this test is to create a linked list where the starting pointer is the bit field STCU\_CFG[PTR]. The additional pointers are in the bit field STCU\_MB\_CTRL[CSM] and STCU\_LB\_CTRL[CSM].
- Sequential or concurrent mode: STCU\_MB\_CTRL[CSM] and STCU\_LB\_CTRL[CSM].  
These fields configure if the chosen set of the LBISTs or MBISTs run concurrently or sequentially. These fields are also used to close the linked list by setting the NIL pointer.
- LBIST pattern counter stop value: STCU\_LB\_PCS
- LBIST expected signatures: STCU\_LB\_MISREL and STCU\_LB\_MISREH.  
These registers contain the expected signatures which are compared to the ones calculated by the STCU.  
Evaluated signatures are saved in some STCU registers, such as STCU\_LB\_MISRRL and STCU\_LB\_MISRRH.
- Start STCU: STCU\_RUN[RUN]  
This is the software trigger which starts the L/MBIST execution.

As an example, please refer to the cmm file:  
"lbist0\_allMBIST\_PLL100Mhz\_parallel\_PCS0x1388" which contains the Lauterbach's script showing a configuration to run the off-line self-test. 91 MBISTs and LBIST0 (that is lake 0) run in parallel mode at 100 MHz, with 5000 (0x1388) number of patterns.

[Figure 7](#) shows these DCF saved in the UTest.

After LBIST execution an automatic functional reset is generated (see [Figure 6](#)).

## 5 On-line test

On-line self-test is performed during the application code execution and it is triggered by the software which configures and starts the L/MISTs execution without involving the SSCM module.

### 5.1 Self-test flow

LBISTs and MBISTs are destructive and the state of tested modules is not recovered automatically by the hardware to the state before the test execution. A functional reset is performed after the on-line L/MBIST before going into operation again.

To enable the automatic generation of the functional reset after the execution of the on-line LBIST, the flag LBRMSWx of the STCU\_LBRMSW reset management register needs to be set '1'.

STCU, which reports the results of L/MBISTs, is not affected by this functional reset. Then software can access such results by reading the STCU registers.

*Figure 8* reports an example of on-line self-tests of both LBISTs and MBISTs. The user can run these tests in inverse order (first MBISTs then LBISTs) or alternately to run only some MBISTs or to run only some LBISTs. The important point is not to forget to setup the functional reset after LBIST execution<sup>(h)</sup>.

Considering this example, after the MBIST execution, the RAM contains random data<sup>(i)</sup> and a read access can trigger ECC events. Before starting executing application code, the RAM needs to be initialized<sup>(j)</sup>.

Before LBIST execution, it is necessary to initialize the RAM and to store in the NVM the MEMU information about MBIST results.

The logical flow is shown in *Figure 8*.

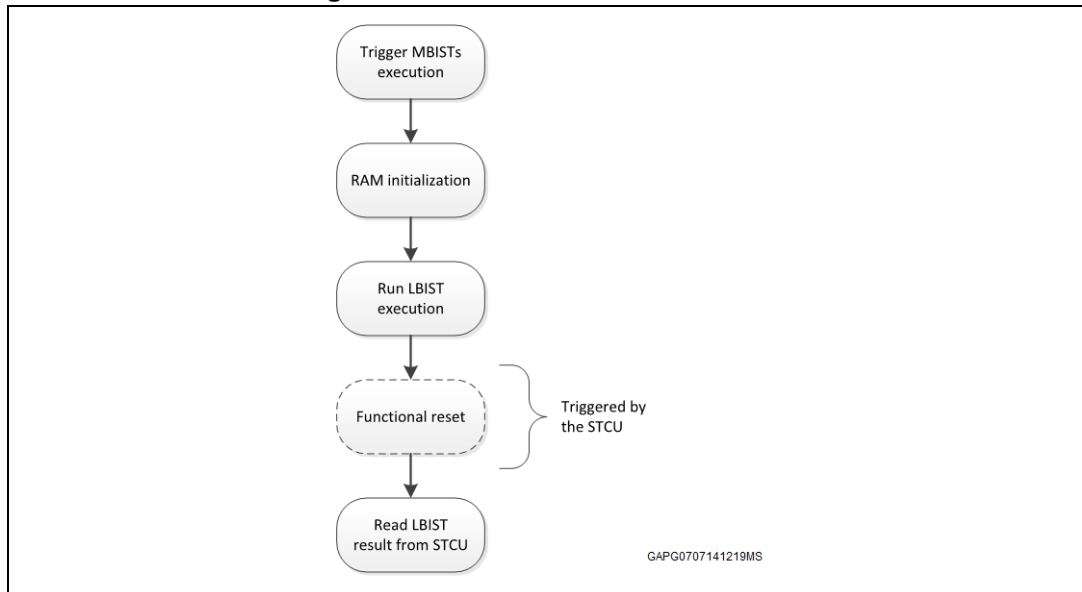
---

h. By the STCU\_LBRMSW register.

i. It means stack, heap and similar sections are destroyed by MBIST.

j. Initializing the RAM means to initialize the bit of the ECC checksum. It is done by writing the all RAM with 32bit/64bit access depending on the ECC implementation.

Figure 8. Self test flow in on-line mode



MBIST results can be interpreted reading both the STCU and the MEMU.

## 5.2 MEMU results

MBIST errors are collected by the MEMU module. MEMU implements multiple error tables used to collect memory errors coming from peripheral SRAM, flash memory and RAM.

MEMU saves more details about ECC errors than the ones reported by STCU (for example address, bit error and so on).

It can trigger an error to the FCCU.

The integrity of the MEMU is verified by LBIST3. For this reason it is suggested to perform the LBIST3 during off-line mode at boot time. When MEMU is LBISTed, it is not usable for MBIST failures during the off-line mode.

## 5.3 On-line configuration

First of all, it is necessary to enable the writing in the STCU module.

Afterwards the user unlocks the write protection to the STCU registers. The couple of keys used in on-line mode are:

- 0x753F\_924Eh key1
- 0x8AC0\_6DB1h key2

Similar setting used for the off-line self-test has to be followed in on-line mode.

After unlocking the STCU, some important registers have to be set in order to start up the test.

There follows the most important:

- Watchdog timer (WDG): STCU\_WDG  
It is used to check if the self-test operations (both LBIST and MBIST) have been completed within the assigned time slot or the STCU\_RUN or BYPASS bits have been programmed before watchdog time-out.
- STCU operative frequency.
- Fault mapping of the STCU2 faults: STCU\_ERR\_FM.  
It defines unrecoverable or recoverable faults.
- Pointer to the first LBIST or MBIST to be scheduled: STCU\_CFG[PTR].  
The philosophy of this kind of test is to create a linked list where the starting pointer is the bit field STCU\_CFG[PTR]. The additional pointers are in the bit field STCU\_MB\_CTRL[CSM]/STCU\_LB\_CTRL[CSM].
- Sequential or concurrent mode: STCU\_MB\_CTRL[CSM] and STCU\_LB\_CTRL[CSM].  
These bit fields need to run concurrently or sequentially the chosen set of the LBIST or MBIST or to close the linked list setting the NIL pointer.
- LBIST pattern counter stop value: STCU\_LB\_PCS.
- LBIST expected signatures: STCU\_LB\_MISRELSW and STCU\_LB\_MISREHSW.  
Considering [Figure 1](#) as reference, this couple of registers have to be compared with the signature (low and high) evaluated that is STCU\_LB\_MISRRLSW and STCU\_LB\_MISRRHSW. Consider that expected LBIST signatures in on-line mode are different from the ones used in off-line mode.
- Start STCU: STCU\_RUNSW[RUN].

[Section Appendix B](#) reports a complete example on how to run MBIST and LBIST in on-line mode. It includes example code for the RAM initialization code and how to store the MBIST results in the NVM before a LBIST execution.

The flash section (NVM) used to store MBIST result is the DATA FLASH BLOCK of SPC58NN84xx.

As in any application code, some actions have been performed before the real self-test execution, including:

- MCU initialization (clocks, MPU, mode entry, cores, XBAR and so on...)
- FCCU initialization
- Flash unlocking
- STCU initialization

## 6 How to read STCU register results and possible reactions

The following sections show:

- the reaction scheme related to MBIST/LBIST executions
- the STCU registers to get the self-test results

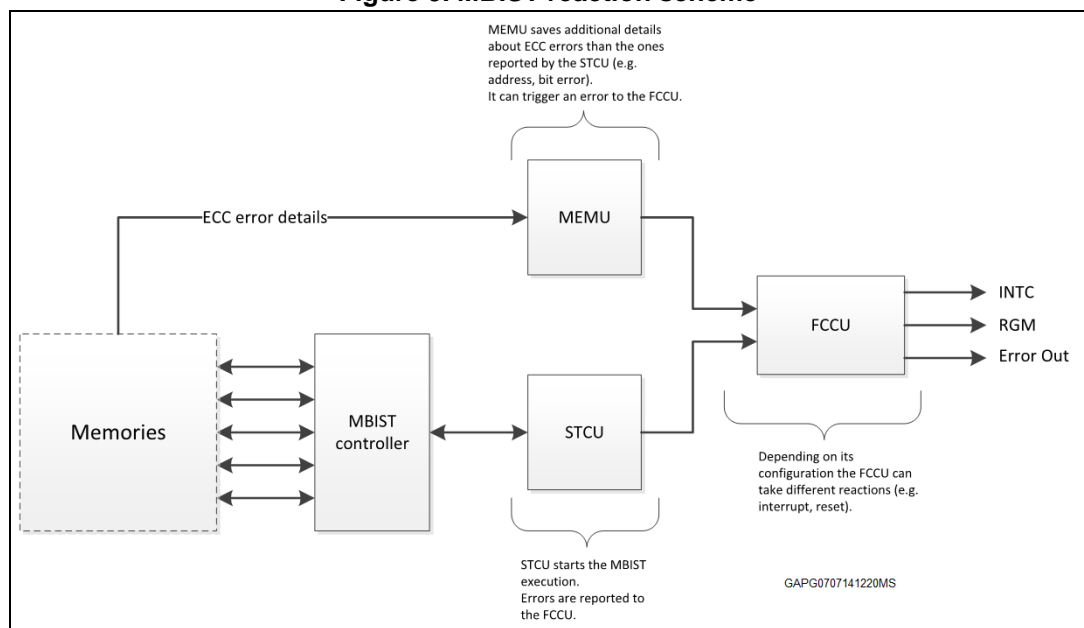
These sections are valid for both on-line and off-line modes.

### 6.1 MBIST reaction

Figure 9 shows the MBIST reaction scheme. Two independent modules collect results of the MBISTs:

- STCU which gives a passed/not passed result for each MBIST
- MEMU which gives additional information related to possible ECC events (for example address of the faulty location)

Figure 9. MBIST reaction scheme



MEMU is able to report correctable (single bit error) and uncorrectable error detected via MBIST. Unique errors are stored in correctable and non-correctable section of the reporting table of the MEMU and corresponding indication is generated to the FCCU. FCCU collects all errors (in this case related to MBIST execution) and, if configured properly, can move the device into a safe state.

FCCU can react in multiple ways to a detected fault, for example interrupt, an external signal or a short/long functional reset.

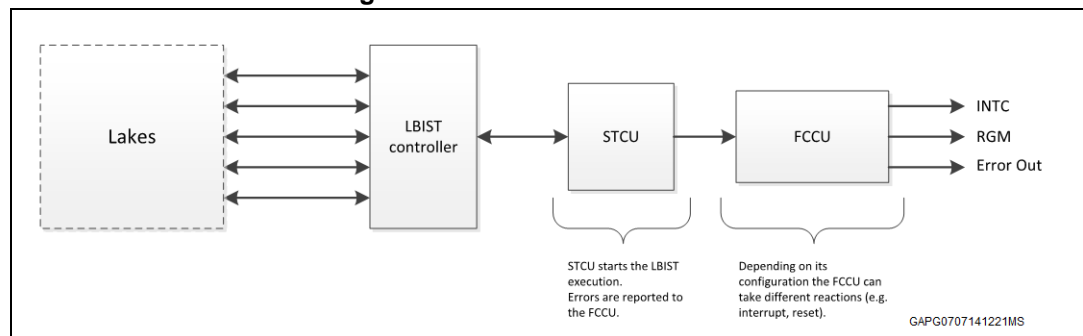
Here the list of STCU register related to MBIST results:

- STCU\_CRCR: this register reports the value obtained at the end of the off-line/on-line self-test. It can be used for diagnose and also as additional check with respect to the CRCS[SW] bit of the STCU\_ERR\_STAT register. This value has to be compared to STCU\_CRCE register value. It is common to LBIST.
- STCU\_ERR\_STAT: this register includes the status flags related to the STCU internal Error conditions occurred during the configuration or the on/off-line self-testing execution. It is common to LBIST.
- STCU\_MBSL / STCU\_MBSLSW: it is a status register. It contains the result of MBIST off-line test. Each bit is associated to a single MBIST, that is bit0 with MBIST0, bit1 with MBIST1 and so on (up to 31).
- STCU\_MBSM / STCU\_MBSMSW: as above, from 32 to 63 bit.
- STCU\_MBSH / STCU\_MBSHSW: as above, from 64 to 95 bit.
- STCU\_MBEL / STCU\_MBELSW: it is an end-flag register (low register). Each bit has an MBIST associated and gives information about if it has been performed or not.
- STCU\_MBEM / STCU\_MBEMSW: it is an end-flag register (medium register).
- STCU\_MBEH /STCU\_MBEHSW: it is an end-flag register (high register).
- STCU\_MBUFML: it defines the fault mapping, in terms of unrecoverable or recoverable fault, of the MBIST in the range NMCUT = 0..31.
- STCU\_MBUFMM: it defines the fault mapping, in terms of unrecoverable or recoverable fault, of the MBIST in the range NMCUT = 32..63.
- STCU\_MBUFMH: it defines the fault mapping, in terms of unrecoverable or recoverable fault, of the MBIST in the range NMCUT = 64..95.

## 6.2 LBIST reaction

Below LBIST reaction:

**Figure 10. LBIST reaction scheme**



STCU is directly connected to LBIST controller (associated to different lakes).

Any failure detected by the LBIST is reported to FCCU.

FCCU collects all errors (in this case related to LBIST execution) and, if configured properly, can move the device into a safe state.

FCCU can react in multiple ways to a detected fault, for example interrupt, an external signal or a short/long functional reset.

Here the list of STCU register related to LBIST results:

- STCU\_CRCCR: it is common to MBIST.
- STCU\_ERR\_STAT: it is common to MBIST.
- STCU\_LBS / STCU\_LBSSW: it is a status register. It contains the result of LBIST off-line test. Each bit is associated to a single LBIST, that is bit0 with LBIST0, bit1 with LBIST1 and so on (up to 31).
- STCU\_LBE / STCU\_LBESW: it is an end-flag register (low register). Each bit has an LBIST associated and gives information about if it has been performed or not.
- STCU\_LB\_MISRRL / STCU\_LB\_MISRRLSW: it reports the LSB part of the MISR obtained at the end of the off-line LBIST controller execution. To be compared with the value expected contained in the STCU\_LB\_MISREL register.
- STCU\_LB\_MISRRH / STCU\_LB\_MISRRHSW: it reports the HSB part of the MISR obtained at the end of the off-line LBIST controller execution. To be compared with the value expected contained in the STCU\_LB\_MISREH register.



## 7 Summary

This document describes how to perform the L/MBISTs to SPC58NN84xx device.

SPC58NN84xx is composed by 7 LBIST partitions (for the logical part) and 92 MBIST partitions (for the volatile memory part).

Self-test can run both in on-line and in off-line mode using different configurations.

Described concepts can be easily reused for other ST devices.

In order to facilitate the execution of off-line and on-line self-test, some examples are available.

## Appendix A FCCU reaction

This section gives some details on the FCCU behavior during, and after, the execution of the on-line LBIST with main focus on the error-out pins.

Before proceeding it is worth recalling the main points about FCCU and STCU activities during the execution on-line LBIST.

*Figure 3* shows the connections involved during the execution of the on-line self-test.

If the STCU detects a failure, the not critical fault #7, such as NCF[7], is triggered to the FCCU.

Independently of the result of the self-test, after the LBIST execution, a functional reset request is generated by the STCU<sup>(k)</sup>:

RMG and STCU are not impacted by this functional reset, but FCCU is.

Hereafter the main default configurations of the FCCU<sup>(l)</sup>:

- NCF[7], that is the one related to the STCU, is disabled
- error-out pins are disabled, which means they are in high impedance state

Software enables the NCF[7] and error-out pins by configuring the proper register in the SIUL and in the FCCU.

*Note*

Registers to be configured are:

- SIUL.MSCR
- FCCU.EOUT\_SIG\_EN[x]
- FCCU.NCFE = 0x0000\_0080

It is assumed that, before the safety application starts, software:

- enables the NCF[7]
- configures NCF[7]'s reaction to move the FCCU FSM to FAULT state
- activates the error-out pins

FCCU is a flexible peripheral which configuration can be adapted to the specific user's needs. Other configuration may be assumed.

Hereafter two simplified flows about FCCU status during the LBIST execution:

- *Figure 11* assumes no fault is detected
- *Figure 12* assumes at least a fault is detected

---

k. No functional reset is triggered after the MBIST execution.

l. This is the FCCU configuration after its reset.

*Figure 11* and *Figure 12* show 4 parameters:

- application phases are the phase of the running application
- FCCU FSM is the current status of the FCCU state machine
- ERROR OUT is the status of the error-out pins
- STCU is the message sent by the STCU to the FCCU

With reference to *Figure 11* it is described below the flow in case no fault is detected.

1. Device goes out of reset (or POR).  
FCCU is in its default state, that is NORMAL.  
Error-out functionality has not been enabled yet and its pins are in Hi-Z.
2. Software configures the whole MCU including the SIUL and the FCCU.  
During this phase the FCCU goes to CONFIG state and error-out pins remain in Hi-Z.
3. Safety function runs.  
FCCU is in NORMAL state and error-out pins signal Normal phase.
4. STCU is configured to run one or more LBIST.  
FCCU and error-out pins are kept unchanged.
5. LBIST is executing.  
Two cases can be differentiated depending whether LBIST #2, which includes the FCCU, is executed.
  - LBIST #2 is NOT executed  
FCCU is in NORMAL state and error-out pins signal Normal phase
  - LBIST #2 is executed  
Integrity of the FCCU is checked by the LBIST #2  
FCCU state is not available while the LBIST runs. Error-out pins are in Hi-Z.
6. A functional reset is triggered by the STCU.  
FCCU goes back to its default status as phase (1).
7. SIUL and FCCU are configured as in phase (2).
8. Safety function runs as in phase (3).

In case the LBIST detects a fault, the flow does not change till phase (5). With reference to [Figure 12](#) only phases starting from (6) are replaced by the following:

9. A functional reset is triggered by the STCU.
  - Since the LBIST detects a fault, the STCU triggers the NCF[7] to the FCCU.
  - FCCU goes back to its default status as phase (1).
  - Since by default the reaction to NCF[7] is disabled, FCCU remains in Normal state and error-out pins in Hi-Z.
10. SIUL and FCCU are configured and, depending on the fault management of the application, the STCU and FCCU can be cleaned.
11. Two different cases can be distinguished depending on the application fault management.
  - Status of STCU and FCCU is cleaned on phase (7).  
FCCU is in NORMAL state, error-out pins signal Normal phase and STCU stops triggering NCF[7] to the FCCU.
  - Status of STCU and FCCU is not cleaned on phase (7).  
As a result the FCCU goes to FAULT state and error-out pins start signaling the Error phase.  
STCU keeps signaling the NCF[7] to the FCCU.

This is a basic example on how the FCCU, STCU and RGM work together during the on-line LBIST execution. Other configurations can be implemented, for example the FCCU may go into ALARM state and trigger an interrupt in case of fault detected by the LBIST instead of going directly to the FAULT state.

**Figure 11. Behavior of the FCCU in case NO fault is detected by LBIST**

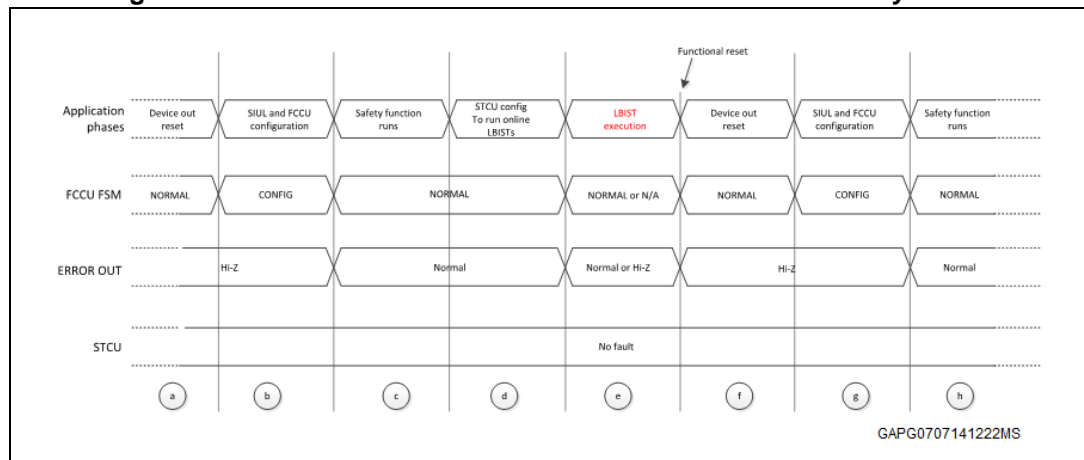
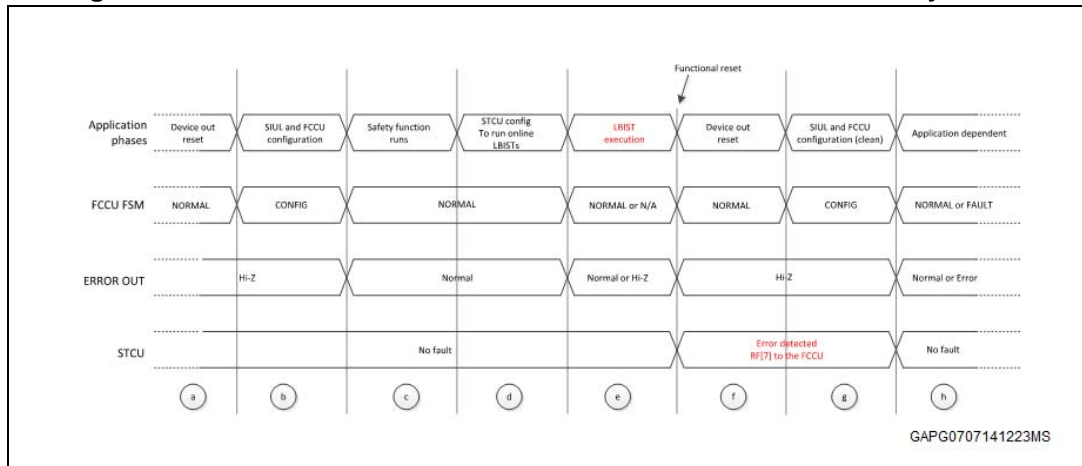


Figure 12. Behavior of the FCCU in case at least a fault is detected by LBIST



## Appendix B Reference code

This section describes the self-test algorithm in on-line mode applied on SPC58NN84xx.

LB1 to LB6 run in parallel mode at 50 MHz (provided by PLL1) plus all MBIST (from 0 to 91) in parallel mode at 50 MHz

The software used is:

- compiler: GHS version 6.1.4
- debugger: Lauterbach Trace32

```
void OnlineSelfTest(void){

    STCU_Init(0xFFFEFFFA);

    RUN_MBIST();

    RAM_initialization();

    writeMEMU_in_NVM(...);//store the MEMU context in the NVM

    STCU_Init(0xFFDDEEAA);

    RUN_LBIST();

}

/***** STCU_Init *****/
/*! this is the STCU initialization. Here is defined how much the watchdog
window size.
/*! 'window size' is the parameter used to defined it.
/*! key1 and key2 are the keys used in on line mode to unlock the STCU
/*!
\return void
*/
void STCU_Init(tU32 window_size) {

    //unlock the Online STCU2 access writing the key1/key2

    STCU2.SKC.R = 0x753F924E;    //key1
    STCU2.SKC.R = 0x8AC06DB1;    //key2
```

```
    STCU2.WDG.R = window_size;

}

/***** RUN_MBIST *****/
//! Here is defined clock configuration and the list of MBIST to be run.
//! They are 39 MBIST, from 0 to 38 in parallel mode.
/*!
\return void
*/
void RUN_MBIST(void) {

    STCU2.CFG.R = 0x105A0010; // set Initial MBIST pointer - #16
                            // PMOS bit set 1

    STCU2.MB_CTRL[0].R = 0x11800000;
    STCU2.MB_CTRL[1].R = 0x12800000;
    STCU2.MB_CTRL[2].R = 0x13800000;
    STCU2.MB_CTRL[3].R = 0x14800000;
    STCU2.MB_CTRL[4].R = 0x15800000;
    STCU2.MB_CTRL[5].R = 0x16800000;
    STCU2.MB_CTRL[6].R = 0x17800000;
    STCU2.MB_CTRL[7].R = 0x18800000;
    STCU2.MB_CTRL[8].R = 0x19800000;
    STCU2.MB_CTRL[9].R = 0x1A800000;
    STCU2.MB_CTRL[10].R = 0x1B800000;
    STCU2.MB_CTRL[11].R = 0x21800000;
    STCU2.MB_CTRL[17].R = 0x22800000;
    STCU2.MB_CTRL[18].R = 0x23800000;
    STCU2.MB_CTRL[19].R = 0x24800000;
    STCU2.MB_CTRL[20].R = 0x25800000;
    STCU2.MB_CTRL[21].R = 0x26800000;
    STCU2.MB_CTRL[22].R = 0x27800000;
    STCU2.MB_CTRL[23].R = 0x28800000;
    STCU2.MB_CTRL[24].R = 0x29800000;
    STCU2.MB_CTRL[25].R = 0x2A800000;
    STCU2.MB_CTRL[26].R = 0x2B800000;
    STCU2.MB_CTRL[27].R = 0x2C800000;
    STCU2.MB_CTRL[28].R = 0x2D800000;
```

```
STCU2.MB_CTRL[29].R = 0x2E800000;
STCU2.MB_CTRL[30].R = 0x2F800000;
STCU2.MB_CTRL[31].R = 0x30800000;

STCU2.MB_CTRL[32].R = 0x31800000;
STCU2.MB_CTRL[33].R = 0x32800000;
STCU2.MB_CTRL[34].R = 0x33800000;
STCU2.MB_CTRL[35].R = 0x34800000;
STCU2.MB_CTRL[36].R = 0x35800000;
STCU2.MB_CTRL[37].R = 0x36800000;
STCU2.MB_CTRL[38].R = 0x37800000;
STCU2.MB_CTRL[39].R = 0x38800000;
STCU2.MB_CTRL[40].R = 0x39800000;
STCU2.MB_CTRL[41].R = 0x3A800000;
STCU2.MB_CTRL[42].R = 0x3B800000;
STCU2.MB_CTRL[43].R = 0x3C800000;
STCU2.MB_CTRL[44].R = 0x3D800000;
STCU2.MB_CTRL[45].R = 0x1C800000;
STCU2.MB_CTRL[12].R = 0x1D800000;
STCU2.MB_CTRL[13].R = 0x1e800000;
STCU2.MB_CTRL[14].R = 0x1f800000;
STCU2.MB_CTRL[15].R = 0x20800000;
STCU2.MB_CTRL[16].R = 0x3E800000;
STCU2.MB_CTRL[46].R = 0x3F800000;
STCU2.MB_CTRL[47].R = 0x40800000;
STCU2.MB_CTRL[48].R = 0x41800000;
STCU2.MB_CTRL[49].R = 0x42800000;
STCU2.MB_CTRL[50].R = 0x43800000;
STCU2.MB_CTRL[51].R = 0x44800000;
STCU2.MB_CTRL[52].R = 0x45800000;
STCU2.MB_CTRL[53].R = 0x46800000;
STCU2.MB_CTRL[54].R = 0x47800000;
STCU2.MB_CTRL[55].R = 0x48800000;
STCU2.MB_CTRL[56].R = 0x49800000;
STCU2.MB_CTRL[57].R = 0x4A800000;
STCU2.MB_CTRL[58].R = 0x4B800000;
STCU2.MB_CTRL[59].R = 0x4C800000;
STCU2.MB_CTRL[60].R = 0x4D800000;
STCU2.MB_CTRL[61].R = 0x4E800000;
STCU2.MB_CTRL[62].R = 0x4F800000;
STCU2.MB_CTRL[63].R = 0x50800000;
STCU2.MB_CTRL[64].R = 0x51800000;
STCU2.MB_CTRL[65].R = 0x52800000;
STCU2.MB_CTRL[66].R = 0x53800000;
STCU2.MB_CTRL[67].R = 0x54800000;
```



```

STCU2.MB_CTRL[68].R = 0x55800000;
STCU2.MB_CTRL[69].R = 0x56800000;
STCU2.MB_CTRL[70].R = 0x57800000;
STCU2.MB_CTRL[71].R = 0x58800000;
STCU2.MB_CTRL[72].R = 0x59800000;
STCU2.MB_CTRL[74].R = 0x5B800000;
STCU2.MB_CTRL[75].R = 0x5C800000;
STCU2.MB_CTRL[76].R = 0x5D800000;
STCU2.MB_CTRL[77].R = 0x5E800000;
STCU2.MB_CTRL[78].R = 0x5F800000;
STCU2.MB_CTRL[79].R = 0x60800000;
STCU2.MB_CTRL[80].R = 0x61800000;
STCU2.MB_CTRL[81].R = 0x62800000;
STCU2.MB_CTRL[82].R = 0x63800000;
STCU2.MB_CTRL[83].R = 0x64800000;
STCU2.MB_CTRL[84].R = 0x65800000;
STCU2.MB_CTRL[85].R = 0x66800000;
STCU2.MB_CTRL[86].R = 0x67800000;
STCU2.MB_CTRL[87].R = 0x68800000;
STCU2.MB_CTRL[88].R = 0x69800000;
STCU2.MB_CTRL[89].R = 0x6A800000;
STCU2.MB_CTRL[90].R = 0x6B800000;
STCU2.MB_CTRL[91].R = 0xFF000000;
STCU2.RUNSW.R = 0x00000001;

/***** RAM_initialization *****/
/!! After the MBIST test the RAM contains random data. Before LBIST
execution, it's necessary to initialize the RAM
/*!
\return void
*/
void RAM_initialization(void) {
    #pragma asm
    //*****
    // Skip normal entry point as nothing is initialized *
    //*****
    .globl _init_ram
    .vle
_init_ram:
    e_lis r5, 0x0000
    e_or2i r5, 0x0200

    mtctr r5;/* Move to counter for use with "bdnz" */

```

```
    // # Base Address of the Local SRAM

    e_lis r5, 0x4000;
    e_or2i r5, 0x0000;

    // # Fill Local SRAM with writes of 32GPRs
    sram_init_loop:;
    e_stmw r0,0(r5); /* Write all 32 registers to SRAM */
    e_addi r5,r5,128; /* Increment the RAM pointer to next 128bytes */
    e_bdnz sram_init_loop; /* Loop for all of SRAM */

    #pragma endasm
}

/***** RUN_LBIST *****/
// ! Here is defined clock configuration and the list of LBIST to be run.
// !
\return void
*/
void RUN_LBIST(void) {

    STCU2.CFG.R = 0x015A0010;

    STCU2.LB[1].CTRL.R = 0x02804400;
    STCU2.LB[1].PCS.R = 0x137E; /*4990*/
    STCU2.LB[1].LB_PRPGL.R = 0xFFFFFFFF;
    STCU2.LB[1].MISRELSW.R = 0xdaf5582b;
    STCU2.LB[1].MISREHSW.R = 0x3b2e32fa;

    STCU2.LB[2].CTRL.R = 0x03804400;
    STCU2.LB[2].PCS.R = 0x137E; /*4990*/
    STCU2.LB[2].LB_PRPGL.R = 0xFFFFFFFF;
    STCU2.LB[2].MISRELSW.R = 0x41e3fa5d;
    STCU2.LB[2].MISREHSW.R = 0xad475ea0;

    STCU2.LB[3].CTRL.R = 0x04804400;
    STCU2.LB[3].PCS.R = 0x137E; /*4990*/
    STCU2.LB[3].LB_PRPGL.R = 0xFFFFFFFF;
    STCU2.LB[3].MISRELSW.R = 0x1DF4D30E;
    STCU2.LB[3].MISREHSW.R = 0xFFFB7696;

    STCU2.LB[4].CTRL.R = 0x05804400;
    STCU2.LB[4].PCS.R = 0x137E; /*4990*/
```

```

STCU2.LB[4].LB_PRPGL.R = 0xFFFFFFFF;
STCU2.LB[4].MISRELSW.R = 0x2f07f39c;
STCU2.LB[4].MISREHSW.R = 0xaf9195ce;

STCU2.LB[5].CTRL.R = 0x06804400;
STCU2.LB[5].PCS.R = 0x137E; /*4990*/
STCU2.LB[5].LB_PRPGL.R = 0xFFFFFFFF;
STCU2.LB[5].MISRELSW.R = 0xae48deec;
STCU2.LB[5].MISREHSW.R = 0xa446dd7c;

STCU2.LB[6].CTRL.R = 0xFF004400;
STCU2.LB[6].PCS.R = 0xE3D; /*3645*/
STCU2.LB[6].LB_PRPGL.R = 0xFFFFFFFF;
STCU2.LB[6].MISRELSW.R = 0xB4002B88;
STCU2.LB[6].MISREHSW.R = 0x4EA80A8E;

STCU2_LBRMSW = 0x0000007E;
STCU2.RUNSW.R = 0x00000101;
}

```

## B.1 Document reference

*SPC58xNx 32-bit Power Architecture® microcontroller for automotive ASILD applications (RM0421, DocID028528).*

## B.2 Acronyms

**Table 5. Acronyms**

Acronyms	Meaning
DCF	Device Configuration Format (DCF) Records
Utest	User Test flash block
STCU	Self-Test Control Unit
SSCM	System Status and Configuration Module
RGM	Reset Generation Module
FCCU	Fault Collection and Control Unit

## Revision history

**Table 6. Document revision history**

Date	Revision	Changes
26-Jun-2017	1	Initial release

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2017 STMicroelectronics – All rights reserved

