AN5184
Application note

SPC58xGx self-test procedures

Introduction

This application note describes how to perform the Logic and Memory BISTs (L/MBIST) implemented on SPC58xGx device.

These self-tests are typically performed during the startup phase to detect latent failures and are transparent for the application.

The user can execute the self test during key-on (offline mode) and key-off (online mode). This guarantees high flexibility to reach for example faster boot time and maintain a high diagnostic coverage against latent failures.

An example of self-tests working in both offline and online modes is available.

This document is focused on SPC58xGx device, but most concepts are valid also for other ST 40 nm devices.

Contents

List of tables

List of figures

1 Overview

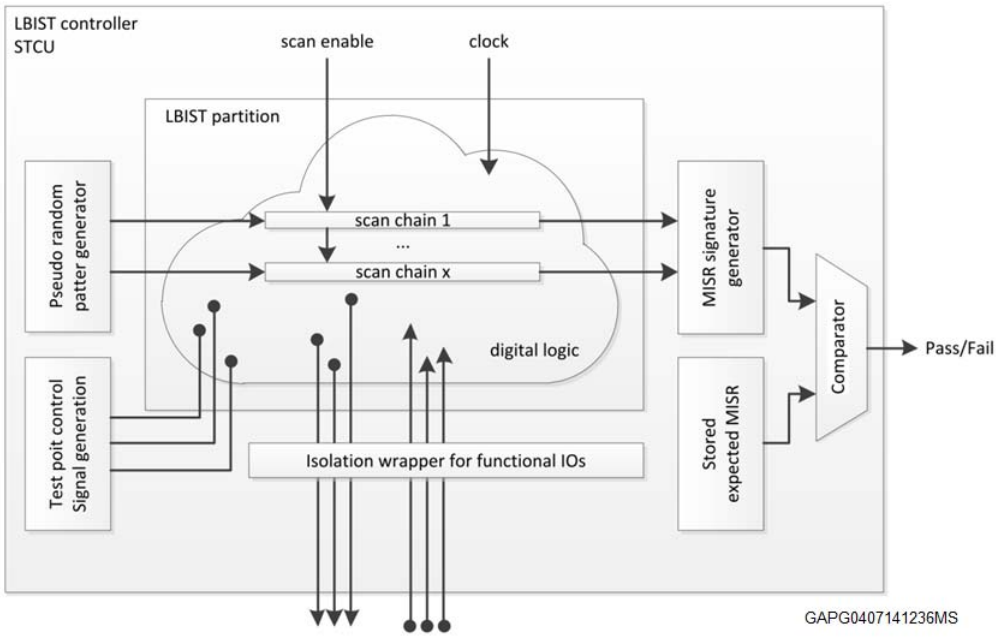
BISTs (Built-In Self-Tests) are periodic tests used to detect latent faults in the functional logic of the device and to guarantee the integrity of the safety mechanisms according to the ISO26262 standard.

Two different types of BIST are included in SPC58xGx device:

- Memory BIST (MBIST) for all volatile memories
- Logic BIST (LBIST) for digital logic

A simplified example of LBIST architecture is shown in the *Figure 1*.

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 to be scanned by a pseudo random pattern generator. Generated signatures (MISR) are compared with the expected ones by the hardware.

SPC58xGx devices consist of one safety LBIST, LBIST0 and 101 MBIST partitions.

Run all 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 MBISTs in 2 groups of self-tests[(a)]:

- One group is executed in offline mode, during the key on phase (offline self-test phase)
- The other one is executed in online mode, during the key off phase (online self-test phase)

BIST execution is controlled by Self-Test Control Unit (STCU) hardware module for both modes, i.e. online and offline.

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

1.1 Offline mode configuration

The STCU shall be configured according to the specific user needs. Its configuration is loaded during the boot phase from UTest sector of the Flash by SSCM (through DCF records).

User saves the right configuration data in the UTest sectors.

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

- L/MBIST scheduling activity
- LBIST setup (including signatures and number of pattern)
- Unrecoverable/recoverable faults management
- CRC
- PLL management

Details on these parameters are provided in the next paragraphs. SPC58xGx device includes one safety LBIST, LBIST0, and 101 MBIST partitions. During offline, the self test strategy on SPC58xGx (designer recommended) is one to perform the LBIST0 together with all MBIST in parallel at 100 Mhz

1.2 Online mode configuration

BIST partitions can be tested also online. A possible strategy is to run the test before the key off event.[(b)]

Test is initiated by software which configures directly the STCU register without using the DCF records.

The results of self-test are stored in NVM (Flash) before shutdown to be used afterwards by the application.

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

- L/MBIST scheduling activity
- LBIST setup (including signatures and number of pattern)
- 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[(c)]. STCU module isn't affected by a functional reset. This gives the possibility to read the self-test results after the reset.

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

b. Also LBIST0 can run in on line mode but is not a mandatory for the safety goal.

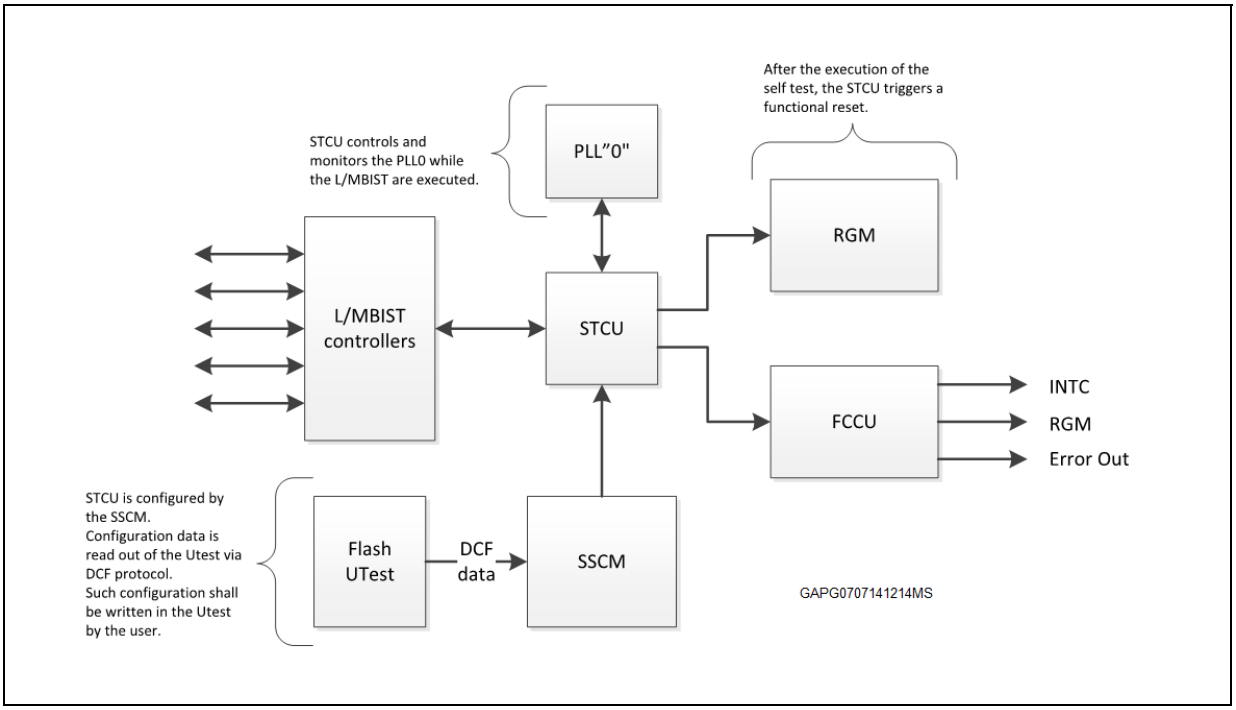
1.3 Modules used in self-test phase.

There are several hardware modules involved in the self-test execution. Their logical interconnection is different according to the self-test modality (either offline or online mode). In offline mode, the modules involved are:

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

Figure 2 shows the interconnection of these modules.

Figure 2. Module interconnections in offline mode



Offline self-test is executed at the start-up by the STCU whose configuration is loaded by the SCCM from UTest sector.

The offline Self-Test can be driven by PLL0 which is directly controlled and configured by the STCU.

Any LBIST/MBIST failure event is forwarded by the STCU to the FCCU which takes the proper reaction. (see *Appendix A: FCCU reaction* for further clarifications).

c. To enable the automatic generation of the functional reset after the execution of the online LBIST, the flag LBRMSWx of the STCU_LBRMSW reset management register shall be set '1'. This reset is a long functional reset which state from PHASE1[FUNC].

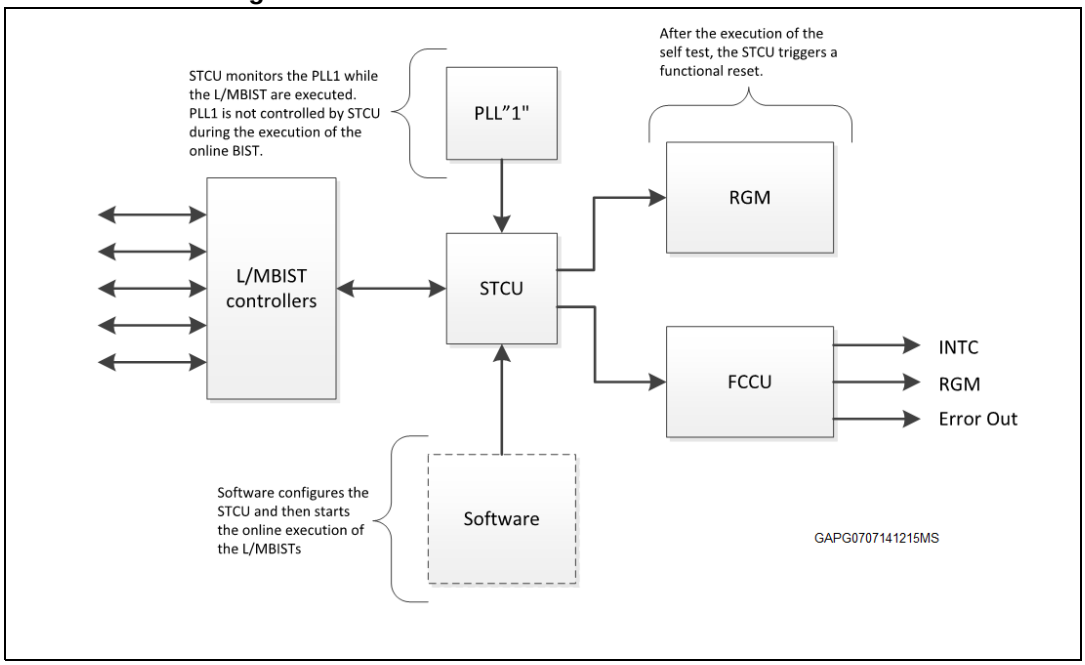
STCU is also connected to the RGM. After the LBIST execution a functional reset is triggered.

In Online mode the interconnection scheme is slightly different. In this case the involved modules are[d]:

- The Reset Generation Module (RGM)
- The Self-Test Control Unit (STCU)
- The Fault Collection and Control Unit (FCCU)
- L/MBIST controllers
- PLL”1”

Figure 3 shows the modules involved.

Figure 3. Module interconnections in online mode



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

In online mode, the action of STCU is limited to PLL Lock signal monitoring[e]. 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 offline mode only PLL0 can be used). *Figure 3* assumes PLL1 has been selected. Additional details on clocking configuration and constraints can be found on *Section 1.5: Clock configurations of the self-test*.

d. 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.

e. During the online self-test, the STCU monitors the lock signals of the PLL1 which in a standard use case drives the system clock.

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

1.4 L/MBIST scheduling activity

STCU is very flexible in terms of scheduling L/MBIST. It allows programming the concurrent or sequential execution of the MBISTs or LBISTs.

Choosing between concurrent and sequential execution has an impact in terms of execution time and current consumption, i.e. concurrent execution means higher consumption but faster execution.

The mechanism used to provide this flexibility is a linked list whose starting pointer is the bit field STCU_CFG[PTR].

The additional pointers are in the bit field 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 bit field STCU_LB_CTRL[CSM] and STCU_MB_CTRL[CSM] provides the flexibility to the chosen set of the LBISTs or MBISTs run concurrently or sequentially or to close the linked list by setting the NIL pointer.

Next example shows how these registers have to be configured to run the first 3 MBISTs concurrently

- STCU_CFG = 0x105A0101 -> PTR = 10h
CLK_CFG = 0x101

It means that the first MBIST is MBIST0. CLK_CFG is equal to sys_clk/6. It isn't the self-test frequency, but related to the memory controller frequency.

- STCU_MB_CTRL_0 = 0x11800000 -> CSM = 1 (concurrent/parallel mode)
PTR = 11h
- STCU_MB_CTRL_1 = 0x12800000> CSM = 1 (concurrent/parallel mode)
PTR = 12h
- STCU_MB_CTRL_2 = 0xFF000000- -> CSM = 0 (This is the last MBIST to be performed, so CSM is set to 0)
PTR = FF (pointer to NIL. No BIST execution)

To run the LBISTs, it is needed to write 0x0 in the PTR field of the STCU_CFG register (first LBIST).

1.5 Clock configurations of the self-test

LBIST and MBIST run with the same sys clock frequency or with different frequencies (in that case the only alternative is the IRC).

1.5.1 Clocking in offline mode

Here after a table that summarizes what are the maximum frequencies involved during offline self test for both devices.

Table 1. Maximum frequencies during offline self test

	LBIST	MBIST
SPC58xGx	100 Mhz	100 Mhz

STCU is able to control the PLL0 depending on the status of the STCU_RUN[MBPLLEN] and STCU_RUN[LBPLLEN].

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)

Following the formula of PLL0 frequency output:

$$fPLLout = \frac{fVCO}{PLLODF \times 2}$$

where:

$$fVCO = fPLLin \times (PLLDF \times 2) = \left(\frac{fRC}{PLLIDF}\right) \times (PLLDF \times 2)$$

fRC is the frequency of internal oscillator that is 16 MHz.

fPLLin should be >= 8 MHz.

1.5.2 Clocking in online mode

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/MBIST clock: represents the clock used by the L/MBISTs

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

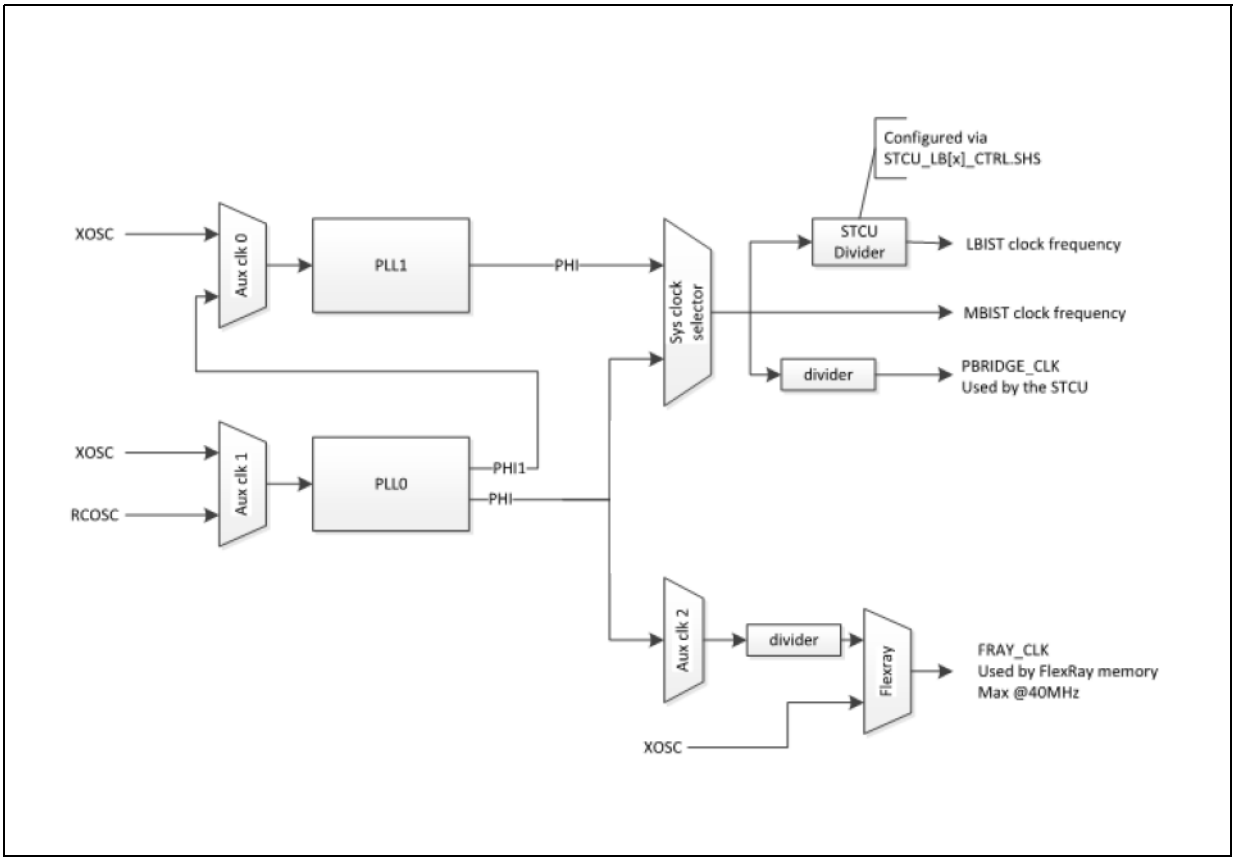
Having a glance at the clock tree of both devices in reference manuals (see *C.1: Reference document*):

- 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 (i.e. PRIDGEA_CLK in *Figure 4*)[f]

Figure 4 underlines the main selectors involved in self-test configuration for SPC58xGx.

f. In case of SPC58xGx this is the PBRIDGE_CLK.

Figure 4. SPC58xGx clock tree for M/LBIST execution



Both PLL0 & PLL1 can be used to run the online L/MBIST, although the typical configuration is to use the PLL1 driven by PLL0 as shown in *Figure 4*.

An additional constraint related to MBIST execution regarding the FlexRay module:

- FlexRay memory works up to 40 MHz

1.6 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 online 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.

Table 2 is referred to SPC58xGx device and summarizes which are STCU registers dedicated to online test and STCU registers dedicated to offline test. (see *C.1: Reference document*).

Table 2. List of STCU on/off line registers

N.	Offline register	Online register	Description
1	STCU_RUN	STCU_RUN_SW	Run Register. Start STCU
2	**STCU_SCK** Offline 0xD3FEA98Bh: Key1 - 0x2C015674h: Key2 Online 0x753F924Eh: Key1 - 0x8AC06DB1h: Key2		SK Code Register.
3	STCU_CFG		Configuration Register. Global configuration of the STCU
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[MBPLLEN] = 1 or STCU_RUN[LBPLLEN] = 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 STCU 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 STCU faults described into the register STCU_ERR_STAT in terms of Unrecoverable or Recoverable Fault.
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

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

N.	Offline register	Online register	Description
12	STCU_LBE	STCU_LBESW	LBIST End Flag Register. It includes the End Flag related to the execution of the selected offline LBIST. 0: LBIST execution not yet completed 1: LBIST execution finished
13		STCU_LBRMSW	LBIST Reset Management. It defines the online 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_MBS1	STCU_MBS1SW	MBIST1 Status Register (from 0 to 31 MBIST). It includes the results corresponding to the execution of the selected Off/On-Line MBIST.
16	STCU_MBS2	STCU_MBS2SW	MBIST2 Status Register (from 32 to 63 MBIST). It includes the results corresponding to the execution of the selected Off/On-Line MBIST.
17	STCU_MBS3	STCU_MBS3SW	MBIST3 Status Register (from 64 to 95 MBIST). It includes the results corresponding to the execution of the selected Off/On-Line MBIST.
18	STCU_MBS4	STCU_MBS4SW	MBIST4 Status register (from 96 to 127 MBIST). It includes the results corresponding to the execution of the selected Off/On-Line MBIST
19	STCU_MBE1	STCU_MBE1SW	MBIST1 End Flag Register (from 0 to 31 MBIST). It includes the results corresponding to the execution of the selected Off/On-Line MBIST
20	STCU_MBE2	STCU_MBE2SW	MBIST2 End Flag Register (from 32 to 63). It includes the End Flag related to the execution of the selected Off/On-Line MBIST.
21	STCU_MBE3	STCU_MBE3SW	MBIST3 End Flag Register (from 64 to 95). It includes the End Flag related to the execution of the selected Off/On-Line MBIST.
22	STCU_MBE4	STCU_MBE4SW	MBIST4 End Flag 4 Register (from 96 to 127) It includes the End Flag related to the execution of the selected Off/On-Line MBIST

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

N.	Offline register	Online register	Description
23	STCU_MBUFM1		MBIST1 Unrecoverable FM Register (from 0 to 31). It defines the fault mapping, in terms of Unrecoverable or Recoverable fault-of the MBIST.
24	STCU_MBUFM2		MBIST2 Unrecoverable FM Register (from 32 to 63). It defines the fault mapping in terms of Unrecoverable or Recoverable fault of the MBIST.
25	STCU_MBUFM3		MBIST3 Unrecoverable FM Register (from 64 to 95). It defines the fault mapping in terms of Unrecoverable or Recoverable fault of the MBIST
26	STCU_MBUFM4		MBIST4 Unrecoverable FM Register (from 96 to 127) It defines the fault mapping, in terms of Unrecoverable or Recoverable fault, of the MBIST.
27	STCU_LB_CTRL		LBIST Control Register. It defines the control setting of each LBIST controller.
28	STCU_LB_PCS		LBIST PC Stop Register. It defines the pattern counter stop of each LBIST controller.
29	STCU_LB_PRPGL		LBIST PRPG Low Register. It defines the LSB part (31...0) of the PRPG start value of the LBIST controller.
30	STCU_LB_PRPGH		LBIST PRPG High Register. It defines the MSB part (32...63) of the PRPG start value of the LBIST controller.
31	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.
32	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.
33	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.

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

N.	Offline register	Online register	Description
34	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.
35	STCU_MB_CTRL		MBIST Control Register. It defines the control setting of MBIST controller.

2 Functional reset sequence in offline mode

After the LBIST execution, a functional reset is generated by the STCU[g]. *Figure 5* shows the reset generation sequence:

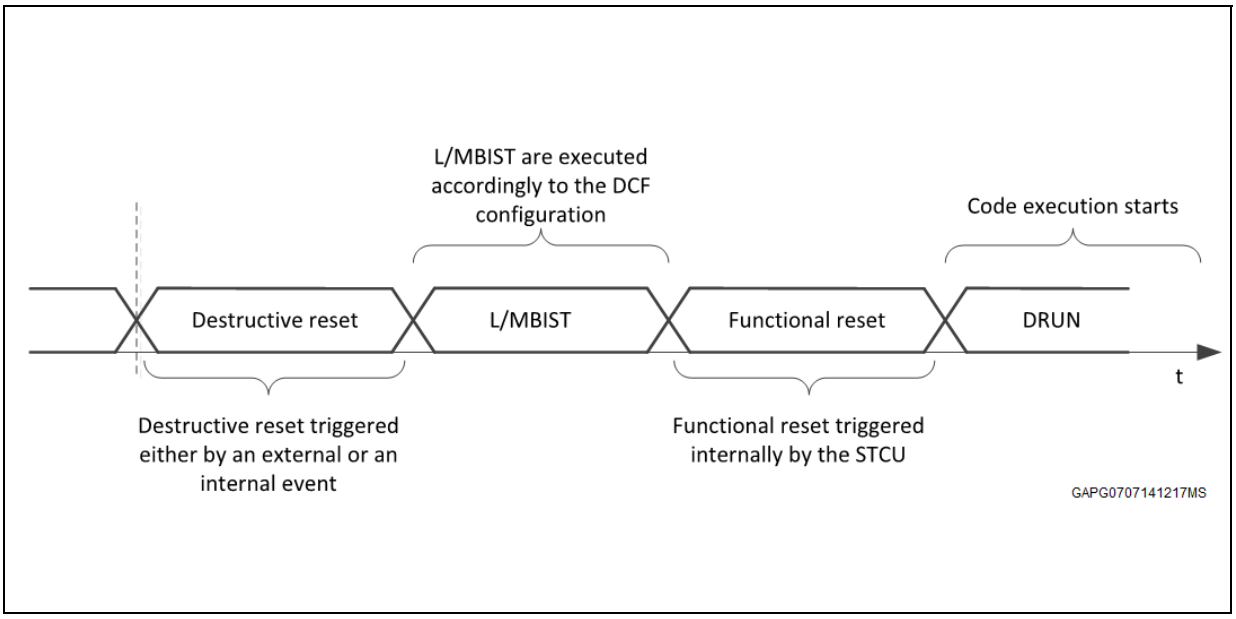
- A destructive reset is triggered by an external or internal event (e.g. POR)
- L/MBIST runs
- After the L/MBIST execution a functional reset is generated by STCU
- The device is ready to execute code in DRUN mode

During the MBIST/LBIST execution:

- A dedicated watchdog is configured to monitor the self-test is completed within an expected time. The length of this phase is variable depending on the self-test configuration.
- 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 5. Destructive and functional RESET sequence in case offline L/MBISTs are enabled.



In case of online mode, a functional reset shall be triggered after execution of the note that an offline self-test won't run after this functional reset.

For more details refer to "Reset and Boot" chapter of SPC58xGx reference manual (see *C.1: Reference document*).

g. This is true only if the correspondent STCU register is set properly.

3 Offline test

Offline self-test is performed during the boot phase after each destructive reset (see *Figure 5*).

3.1 DCF records to program in offline 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:

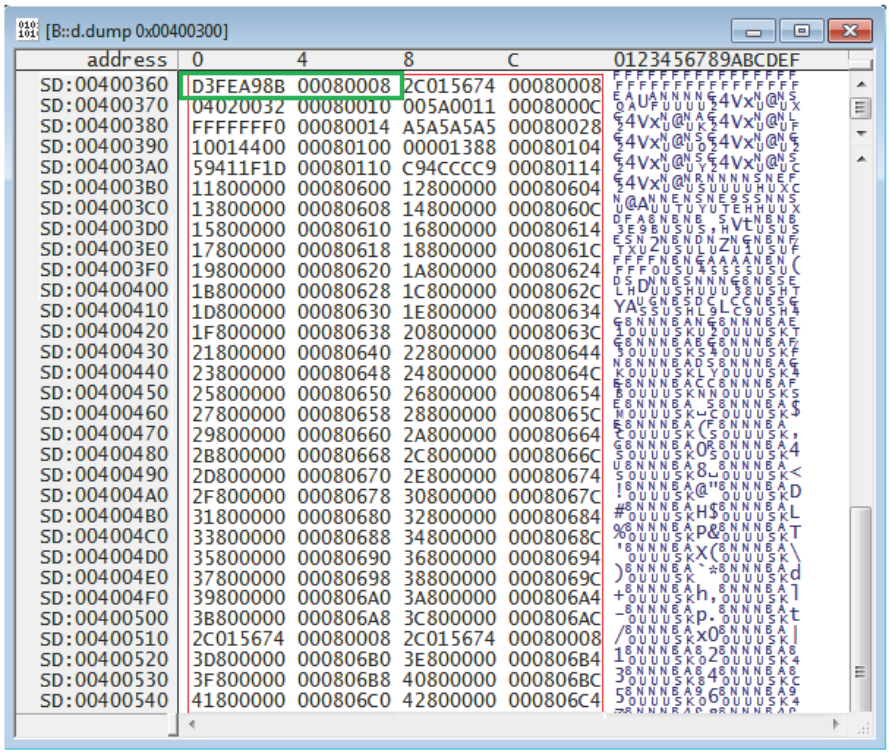
- TEST DCF Records: they are programmed into TEST Flash during production with some information about trimming parameters (e.g. trip points for voltage comparators, adjusting analog to digital voltage supplies parameters, trimming oscillator parameters, frequencies and RAM repair enable). This memory is not visible to the user who can't 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, e.g. 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: the data to be written to a specific register and a pointer to the location of this register[(h)].

All details about DCF and how it shall be used can be found at the section "Device Configuration Format (DCF) Records" of the RM (*C.1: Reference document*).

h. This word includes also additional configuration.

Figure 6. An example of DCF record



An example of DCF record is shown in *Figure 6* in the green box. It contains the value which is automatically copied by the SSCU to the STCU_SCK (key 1) register during the boot process.

The DCF start record is in 0x00400300. According to the device the first location for the user DCF records can change. In case of SPC58xGx the first free location is 0x00400730

Section 3.1.1 describes an example of self-test that includes MBIST and LBIST tests.

3.1.1 Example of DCF for self-test in offline 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: to check if the Self-Test operations (both LBIST than MBIST) have been completed within the assigned time slot or the STCU_RUN or BYPASS bits have been programmed before Watchdog time-out. The DS reports the execution time of L/MBISTs.
- The Watchdog timeout must to be higher than the sum of the execution times of all selected BISTs.
- To estimate the execution time, user has to consider that offline self-test can be driven either by the IRCOSC or PLL0.
- Fault mapping of the STCU faults->STCU_ERR_FM. It defines if a fault is considered either as unrecoverable or recoverable.

- Pointer to the first LBIST or MBIST to 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 with the calculated ones by the STCU.
- Evaluated signatures are saved in some STCU registers, i.e. STCU_LB_MISRRL and STCU_LB_MISRRH.

Start STCU->STCU_RUN[RUN]. This is the software trigger which starts the L/MBIST execution.

This example contains an extract of a Lauterbach script to run 101 MBIST in parallel plus LBIST0 @100 Mhz. For SPC58xGx the Flash programming has to be at 128 bits.

```
;Lock0 ->TSLock enable UTEST memory
PER.S ANC:0xF7FE0010 %LONG 0x3FFFFFFF
print "UTest Unlocked"

print "Unlock UTEST writing STCU Keys"
GOSUB program_128
&current_address 0xD3FEA98B00080008 0x2C01567400080008

&current_address=&current_address+0x10
;PLL set @100Mhz
;LDF=0x32
;IDF=0x2
;ODF=0x4
print "STCU_PLL_CFG and STCU_CFG programming"
GOSUB program_128
&current_address 0x0402003200080010 0x005A00010008000C

&current_address=&current_address+0x10
print "STCU_WDG and STCU_ERR_FM programming"
GOSUB program_128
&current_address 0xFFFFFFF000080014 0xA5A5A5A500080028

&current_address=&current_address+0x10
print "LB_CTRL0 and LB_PCS0 programming"
GOSUB program_128
&current_address 0x1001440000080100 0x0000138800080104

&current_address=&current_address+0x10
print "LB_MISREL_0 and LB_MISRELH_0 programming"
GOSUB program_128
&current_address 0x9CB04E4200080110 0xF5DA16D600080114

&current_address=&current_address+0x10
print "MB_CTRL0 and MB_CTRL1 programming"
GOSUB program_128
&current_address 0x1180000000080600 0x1280000000080604

&current_address=&current_address+0x10
print "MB_CTRL2 and MB_CTRL3 programming"
GOSUB program_128
&current_address 0x1380000000080608 0x148000000008060C
```

```
&current_address=&current_address+0x10
print "MB_CTRL4 and MB_CTRL5 programming"
GOSUB program_128
&current_address 0x1580000000080610 0x1680000000080614

&current_address=&current_address+0x10
print "MB_CTRL6 and MB_CTRL7 programming"
GOSUB program_128
&current_address 0x1780000000080618 0x188000000008061C

&current_address=&current_address+0x10
print "MB_CTRL8 and MB_CTRL9 programming"
GOSUB program_128
&current_address 0x1980000000080620 0x1A80000000080624

&current_address=&current_address+0x10
print "MB_CTRL10 and MB_CTRL11 programming"
GOSUB program_128
&current_address 0x1B80000000080628 0x1C8000000008062C

&current_address=&current_address+0x10
print "MB_CTRL12 and MB_CTRL13 programming"
GOSUB program_128
&current_address 0x1D80000000080630 0x1E80000000080634

&current_address=&current_address+0x10
print "MB_CTRL14 and MB_CTRL15 programming"
GOSUB program_128
&current_address 0x1F80000000080638 0x208000000008063C

&current_address=&current_address+0x10
print "MB_CTRL16 and MB_CTRL17 programming"
GOSUB program_128
&current_address 0x2180000000080640 0x2280000000080644

&current_address=&current_address+0x10
print "MB_CTRL18 and MB_CTRL19 programming"
GOSUB program_128
&current_address 0x2380000000080648 0x248000000008064C
```

```
&current_address=&current_address+0x10
print "MB_CTRL20 and MB_CTRL21 programming"
GOSUB program_128
&current_address 0x2580000000080650 0x2680000000080654

&current_address=&current_address+0x10
print "MB_CTRL22 and MB_CTRL23 programming"
GOSUB program_128
&current_address 0x2780000000080658 0x288000000008065C

&current_address=&current_address+0x10
print "MB_CTRL24 and MB_CTRL25 programming"
GOSUB program_128
&current_address 0x2980000000080660 0x2A80000000080664

&current_address=&current_address+0x10
print "MB_CTRL26 and MB_CTRL27 programming"
GOSUB program_128
&current_address 0x2B80000000080668 0x2C8000000008066C

&current_address=&current_address+0x10
print "MB_CTRL28 and MB_CTRL29 programming"
GOSUB program_128
&current_address 0x2D80000000080670 0x2E80000000080674

&current_address=&current_address+0x10
print "MB_CTRL30 and MB_CTRL31 programming"
GOSUB program_128
&current_address 0x2F80000000080678 0x308000000008067C

&current_address=&current_address+0x10
print "MB_CTRL32 and MB_CTRL33 programming"
GOSUB program_128
&current_address 0x3180000000080680 0x3280000000080684

&current_address=&current_address+0x10
print "MB_CTRL34 and MB_CTRL35 programming"
GOSUB program_128
&current_address 0x3380000000080688 0x348000000008068C
```

```
&current_address=&current_address+0x10
print "MB_CTRL36 and MB_CTRL37 programming"
GOSUB program_128
&current_address 0x3580000000080690 0x3680000000080694

&current_address=&current_address+0x10
print "MB_CTRL38 and MB_CTRL39 programming"
GOSUB program_128
&current_address 0x3780000000080698 0x388000000008069C

&current_address=&current_address+0x10
print "MB_CTRL40 and MB_CTRL41 programming"
GOSUB program_128
&current_address 0x39800000000806A0 0x3A800000000806A4

&current_address=&current_address+0x10
print "MB_CTRL42 and MB_CTRL43 programming"
GOSUB program_128
&current_address 0x3B800000000806A8 0x3C800000000806AC

&current_address=&current_address+0x10
print "MB_CTRL44 and MB_CTRL45 programming"
GOSUB program_128
&current_address 0x3D800000000806B0 0x3E800000000806B4

&current_address=&current_address+0x10
print "MB_CTRL46 and MB_CTRL47 programming"
GOSUB program_128
&current_address 0x3F800000000806B8 0x40800000000806BC

&current_address=&current_address+0x10
print "MB_CTRL48 and MB_CTRL49 programming"
GOSUB program_128
&current_address 0x41800000000806C0 0x42800000000806C4

&current_address=&current_address+0x10
print "MB_CTRL50 and MB_CTRL51 programming"
GOSUB program_128
&current_address 0x43800000000806C8 0x44800000000806CC
```

```
&current_address=&current_address+0x10
print "MB_CTRL52 and MB_CTRL53 programming"
GOSUB program_128
&current_address 0x45800000000806D0 0x46800000000806D4

&current_address=&current_address+0x10
print "MB_CTRL54 and MB_CTRL55 programming"
GOSUB program_128
&current_address 0x47800000000806D8 0x48800000000806DC

&current_address=&current_address+0x10
print "MB_CTRL56 and MB_CTRL57 programming"
GOSUB program_128
&current_address 0x49800000000806E0 0x4A800000000806E4

&current_address=&current_address+0x10
print "MB_CTRL58 and MB_CTRL59 programming"
GOSUB program_128
&current_address 0x4B800000000806E8 0x4C800000000806EC

&current_address=&current_address+0x10
print "MB_CTRL60 and MB_CTRL61 programming"
GOSUB program_128
&current_address 0x4D800000000806F0 0x4E800000000806F4

&current_address=&current_address+0x10
print "MB_CTRL62 and MB_CTRL63 programming"
GOSUB program_128
&current_address 0x4F800000000806F8 0x50800000000806FC

&current_address=&current_address+0x10
print "MB_CTRL64 and MB_CTRL65 programming"
GOSUB program_128
&current_address 0x5180000000080700 0x5280000000080704

&current_address=&current_address+0x10
print "MB_CTRL66 and MB_CTRL67 programming"
GOSUB program_128
&current_address 0x5380000000080708 0x548000000008070C
```

```
&current_address=&current_address+0x10
print "MB_CTRL68 and MB_CTRL69 programming"
GOSUB program_128
&current_address 0x5580000000080710 0x5680000000080714

&current_address=&current_address+0x10
print "MB_CTRL70 and MB_CTRL71 programming"
GOSUB program_128
&current_address 0x5780000000080718 0x588000000008071C

&current_address=&current_address+0x10
print "MB_CTRL72 and MB_CTRL73 programming"
GOSUB program_128
&current_address 0x5980000000080720 0x5A80000000080724

&current_address=&current_address+0x10
print "MB_CTRL74 and MB_CTRL75 programming"
GOSUB program_128
&current_address 0x5B80000000080728 0x5C8000000008072C

&current_address=&current_address+0x10
print "MB_CTRL76 and MB_CTRL77 programming"
GOSUB program_128
&current_address 0x5D80000000080730 0x5E80000000080734

&current_address=&current_address+0x10
print "MB_CTRL78 and MB_CTRL79 programming"
GOSUB program_128
&current_address 0x5F80000000080738 0x608000000008073C

&current_address=&current_address+0x10
print "MB_CTRL80 and MB_CTRL81 programming"
GOSUB program_128
&current_address 0x6180000000080740 0x6280000000080744

&current_address=&current_address+0x10
print "MB_CTRL82 and MB_CTRL83 programming"
GOSUB program_128
&current_address 0x6380000000080748 0x648000000008074C
```

```
&current_address=&current_address+0x10
print "MB_CTRL84 and MB_CTRL85 programming"
GOSUB program_128
&current_address 0x6580000000080750 0x6680000000080754

&current_address=&current_address+0x10
print "MB_CTRL86 and MB_CTRL87 programming"
GOSUB program_128
&current_address 0x6780000000080758 0x688000000008075C

&current_address=&current_address+0x10
print "MB_CTRL88 and MB_CTRL89 programming"
GOSUB program_128
&current_address 0x6980000000080760 0x6A80000000080764

&current_address=&current_address+0x10
print "MB_CTRL90 and MB_CTRL91 programming"
GOSUB program_128
&current_address 0x6B80000000080768 0x6C8000000008076C

&current_address=&current_address+0x10
print "MB_CTRL92 and MB_CTRL93 programming"
GOSUB program_128
&current_address 0x6D80000000080770 0x6E80000000080774

&current_address=&current_address+0x10
print "MB_CTRL94 and MB_CTRL95 programming"
GOSUB program_128
&current_address 0x6F80000000080778 0x708000000008077C

&current_address=&current_address+0x10
print "MB_CTRL96 and MB_CTRL97 programming"
GOSUB program_128
&current_address 0x7180000000080780 0x7280000000080784

&current_address=&current_address+0x10
print "MB_CTRL98 and MB_CTRL99 programming"
GOSUB program_128
&current_address 0x7380000000080788 0x748000000008078C
```

```
      &current_address=&current_address+0x10
      print "MB_CTRL100 and MB_CTRL101 programming"
      GOSUB program_128
      &current_address 0x7580000000080790 0xFF00000000080794

      print " RUN  and Dummy DCF"
      &current_address=&current_address+0x10
      GOSUB program_128
      &current_address 0x0000030100080000 0x00000000010003FC


program_128:
      entry &address &data &data1

      ;MCR->PGM =1 enable program memory
      PER.S ANC:0xF7FE0000 %LONG 0x610

      D.S EA:(&address) %BE %QUAD (&data)
      print "written 0x" &data " at 0x" &address

      D.S EA:(&address+0x8) %BE %QUAD (&data1)
      print "written 0x" &data " at 0x" &address

      ;MCR->EHV=1 program memory
      PER.S ANC:0xF7FE0000 %LONG 0x611

      ;while (FLASH.MCR.B.DONE == 0);
      WHILE ((Data.Long(ea:0xF7FE0000)&0x0200)==0);

      ;MCR->EHV=0 program memory
      PER.S ANC:0xF7FE0000 %LONG 0x610

      ;MCR->PGM =0
      PER.S ANC:0xF7FE0000 %LONG 0x600

      RETURN
```

4 On line test case

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

Online BIST can be used to perform LBIST of partitions not yet tested in offline mode and to re-execute the MBIST with an algorithm different from the one used during offline mode[(i)]. In such a way advanced methodologies can be implemented, e.g. splitting the execution of L/MBISTs in multiple sub-groups. LBIST0 can be perfomed also in online mode.

4.1 Self-test flow

LBISTs and MBISTs are destructive and the state of tested modules is not recovered automatically by the hardware to the one before the test execution.

A functional reset must be executed after the online LBIST execution. To enable the automatic generation of this functional reset, the flag LBRMSWx of the STCU_LBRMSW reset management register shall be set ‘1’.

Since STCU, which reports the results of L/MBISTs, is not affected by this functional reset, software can read its registers to access the self-test result.

Figure 7 reports an example of online 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 only some LBISTs. The important point is not to forget to setup the functional reset after LBIST execution[(j)].

After the MBIST execution the RAM contains random data[(k)] and a read access can trigger ECC events. Before starting executing application code, the RAM shall be initialized[(l)].

Considering the provided example, 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 on *Figure 7*.

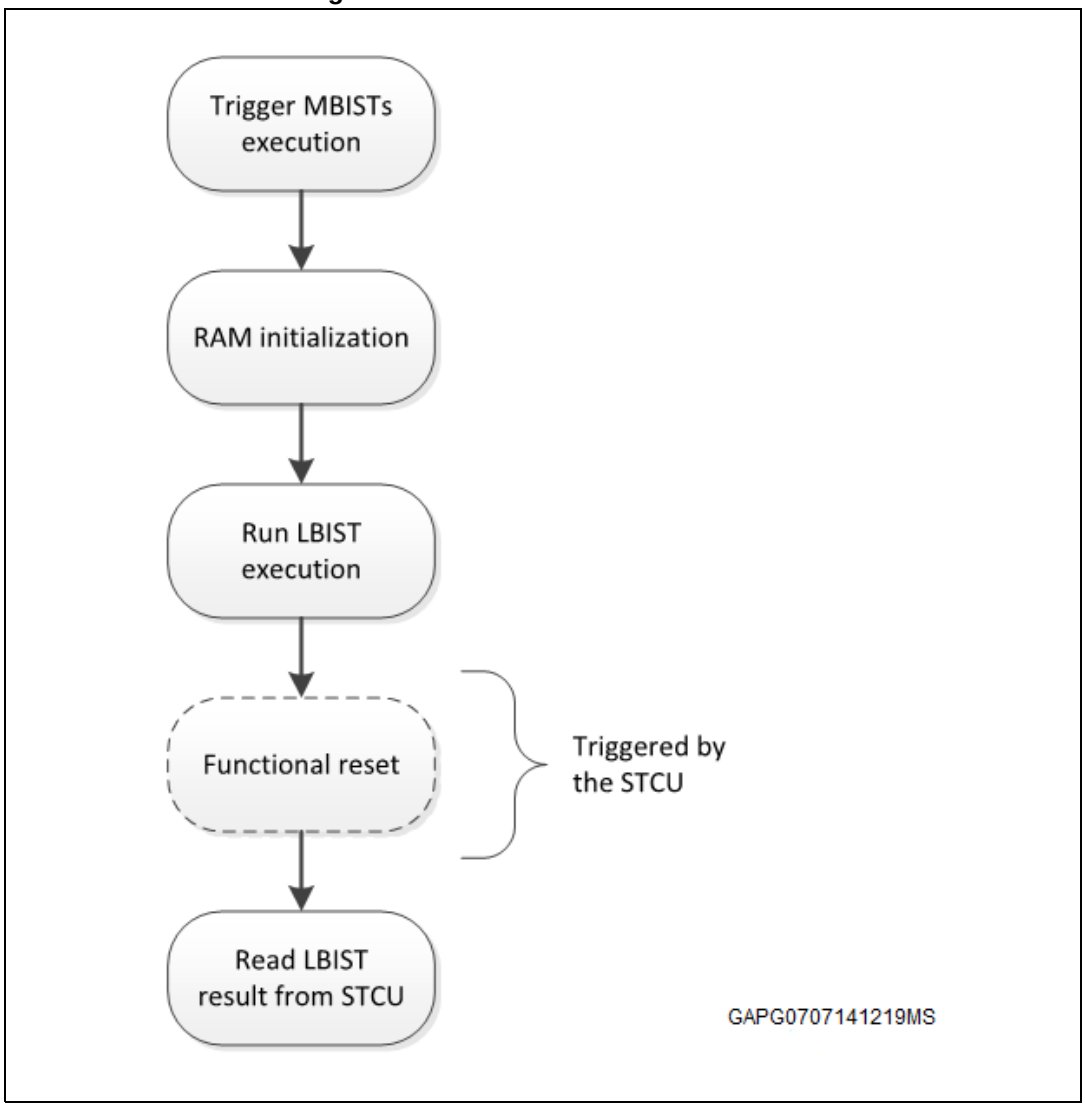
i. Executing MBIST algorithms, which are different between offline and online mode, guarantees a higher coverage against different types of *failure modes*.

j. By the STCU_LBRMSW register.

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

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

Figure 7. Self test flow in online mode



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

4.2 MEMU results

MBIST errors are collected by the MEMU module. MEMU implements multiple error tables used to collect memory errors coming from different memories.

MEMU saves additional details about ECC errors than the ones reported by STCU (e.g address, bit error and so on). It can trigger a fault to the FCCU.

The integrity of the MEMU is verified by the execution of the LBIST3. For this reason it is suggested to perform the LBIST3 during offline mode at boot time. If MEMU is LBISTed during the offline mode, it is not usable for collecting MBIST failures during the offline mode.

4.3 Online configuration

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

Afterwards the user shall unlock the write protection to the STCU registers using the following couple of keys:

- 0x753F_924Eh Key1
- 0x8AC0_6DB1h Key2

Similar settings as the ones used for the offline self-test shall be configured in online mode.

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

Following the most important:

- Watchdog timer (WDG)-> STCU_WDG: to check if the self-test operations (both LBIST than 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: it runs @ 100 MHz by PLL0.
- Fault mapping of the STCU 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 the *Figure 1: Basic single LBIST architecture*, this couple of registers has 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 online mode are different that ones used in offline mode.
- Start STCU->STCU_RUNSW[RUN].

The Flash section (NVM) used to store MBIST result is the DATA FLASH BLOCK of SPC58xGx device.

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

5 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.

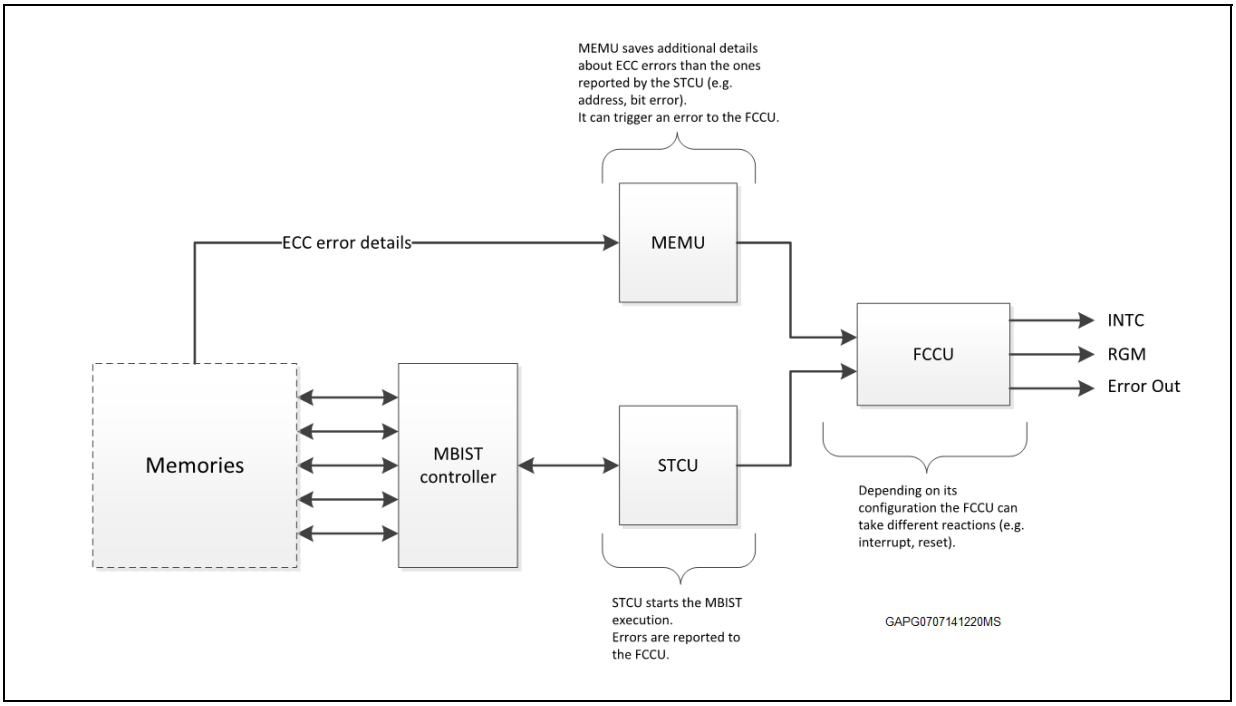
These sections are valid for both online and offline modes.

5.1 MBIST reaction

Figure 8 shows the MBIST reaction scheme. Two independent modules collect results of the MBISTs:

- STCU which give a passed/not passed result for each MBIST, and MEMU which gives additional information related possible ECC events (e.g. address of the faulty location).

Figure 8. 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 takes the proper reaction(m), e.g. interrupt, an external signal or a short/long functional reset.

m. The “proper” reaction of the FCCU is application dependent and shall be configured by the user before launching the online self test.

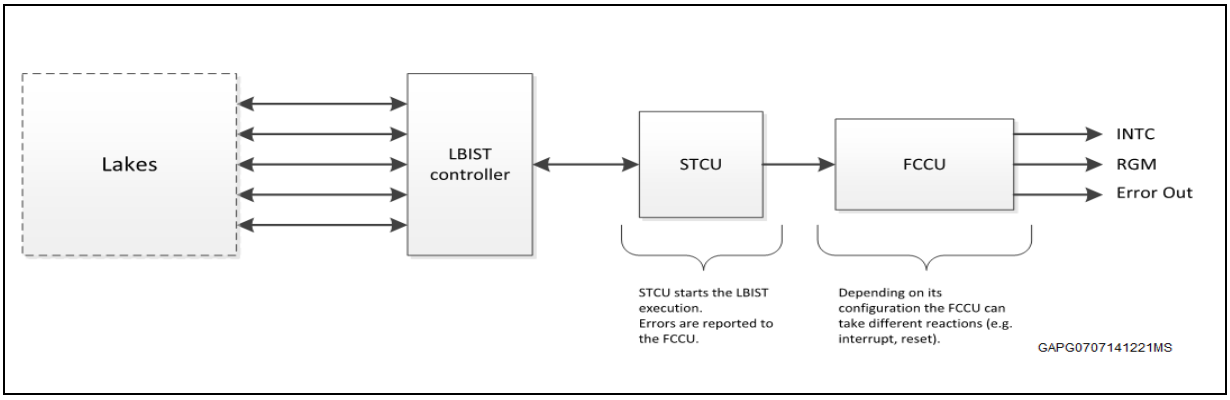
Here the list of STCU registers related to MBIST results:

- STCU_CRCR: this register for SPC58xGx device reports the value obtained at the end of the offline/online 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_MBS1/STCU_MBS1SW: It is a status register. It contains the result of MBIST Offline test. Each bit is associated to a single MBIST, that is bit0 with MBIST0, bit1 with MBIST1 and so on (up to 31).
- STCU_MBS2/STCU_MBS2SW: as above, from 32 to 63 bits.
- STCU_MBS3/ STCU_MBS3SW: as above, from 64 to 95 bits.
- STCU_MBS4/STCU_MBS4SW: as above m from 96 to 127 bits
- STCU_MBE1/ STCU_MBE1SW: It is an end-flag register. Each bit has an MBIST associated and gives information about if it has been performed or not.
- STCU_MBE2/STCU_MBE2SW: It is an end-flag register.
- STCU_MBE3/STCU_MBE3SW: It is an end-flag register.
- STCU_MBE4/STCU_MBE4SW: it is an end-flag register
- STCU_MBUFML1t defines the fault mapping, in terms of Unrecoverable or Recoverable fault, of the MBIST in the range NMCUT = 0...31.
- STCU_MBUFM2: It defines the fault mapping, in terms of Unrecoverable or Recoverable fault, of the MBIST in the range NMCUT = 32...63.
- STCU_MBUFM3: It defines the fault mapping, in terms of Unrecoverable or Recoverable fault, of the MBIST in the range NMCUT = 64...95.
- STCU_MBUFM4: it defines the fault mapping in terms of Unrecoverable or Recoverable fault, of the MBIST in the range NMCUT = 96..127

5.2 LBIST reaction

Here follows an LBIST reaction scheme:

Figure 9. LBIST reaction scheme



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

Any failure detected by the LBIST is reported to FCCU which collects all errors (in this case related to LBIST execution) and takes the proper reaction(m), e.g. interrupt, an external signal or a short/long functional reset.

Here the list of STCU register related to LBIST results:

- STCU_CRCR: 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 Offline 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.

6 Summary

This document describes how to perform the L/MBISTs on SPC58xGx devices.

To speed up the boot process, it is possible to split the self-test in two (or even more) sub-tests: some BISTs (L/MBISTS) to be executed during the offline mode and the remaining ones during the online mode.

The MBISTs run in both modes but with different algorithms.

STCU configuration and connections with other modules are described.

Described concepts can be easily reused for other ST devices of 55 and 40 nm family.

To facilitate the implementation of the offline and online 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 online LBIST with main focus on the error-out pins.The *Figure 10* shows the connections involved during the execution of the online self-test.

If the STCU detects a failure, the Recoverable Fault #7, i.e. RF[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[n]:

- RMG and STCU are not impacted by this functional reset
- FCCU is impacted by this functional reset

Hereafter the main default configurations of the FCCU[o]:

1. RF[7], i.e. the one related to the STCU, is disabled
2. Error-out pins are disabled, i.e. they are in high impedance state

Software explicitly enables the RF[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 and FCCU. EOUT_SIG_EN[x] and FCCU.NCFE = 0x0000_0080

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

- Enables the RF[7]
- Configures RF[7]'s reaction to move the FCCU FSM to FAULT state and Activates the error-out pins

FCCU is a flexible peripheral whose 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 10* assumes no fault is detected and
- *Figure 11* assumes at least a fault is detected.

Figure 10 and *Figure 11* show 4 parameters:

- Application phases is 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 message sent by the STCU to the FCCU

n. no functional reset is triggered after the MBIST execution.

o. this is the FCCU configuration after its reset.

With reference to *Figure 10*, first it is described the flow in case no fault is detected:

a) Device goes out of reset (or POR).
The FCCU is in its default state, i.e. NORMAL.
Error-out functionality has not been enabled yet and its pins are in Hi-Z.

b) 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.

c) Safety function runs.
FCCU is in NORMAL state and error-out pins signal Normal phase.

d) STCU is configured to run one of more LBIST
FCCU and error-out pins are kept unchanged.

e) LBIST is executing.
Two cases shall be differentiated depending whether LBIST #2, which includes the FCCU, is executed[p]:

– LBIST #2 in 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.

f) A functional reset is triggered by the STCU.
FCCU goes back to its default status as phase (a)

g) SIUL and FCCU are configured as in phase (b)

h) Safety function runs as in phase (c)

The flow does not change till phase (e), in case the LBIST detects a fault. With reference to *Figure 11* only phases starting from (f) are described in this case:

f) A functional reset is triggered by the STCU.
Since the LBIST detects a fault, the STCU trigger the RF[7] to the FCCU.
FCCU goes back to its default status as phase (a).
Since by default the reaction to RF[7] is disabled, FCCU remains in NORMAL state and error-out pins in Hi-Z.

g) SIUL and FCCU are configured and, depending on the fault management of the application, the STCU and FCCU can be cleaned.

h) Two different cases can be distinguished depending on the application fault management.

– Status of STCU and FCCU is cleaned on phase (g).
FCCU is in NORMAL state, error-out pins signals Normal phase and STCU stops triggering RF[7] to the FCCU.

– Status of STCU and FCCU is not cleaned on phase (g).
As result the FCCU goes to FAULT state and error-out pins start signaling the Error phase.
STCU keeps signaling the RF[7] to the FCCU.

This is a basic example on how the STCU, STCU and RGM work together during the online LBIST execution. Other configurations can be implemented, e.g. the FCCU may go into

p. User decides which LBIST run during the STCU configuration.

ALARM state and trigger an interrupt in case of fault detected by the LBIST instead of going directly to the FAULT state.

Figure 10. Behavior of the FCCU in case NO fault is detected by LBIST

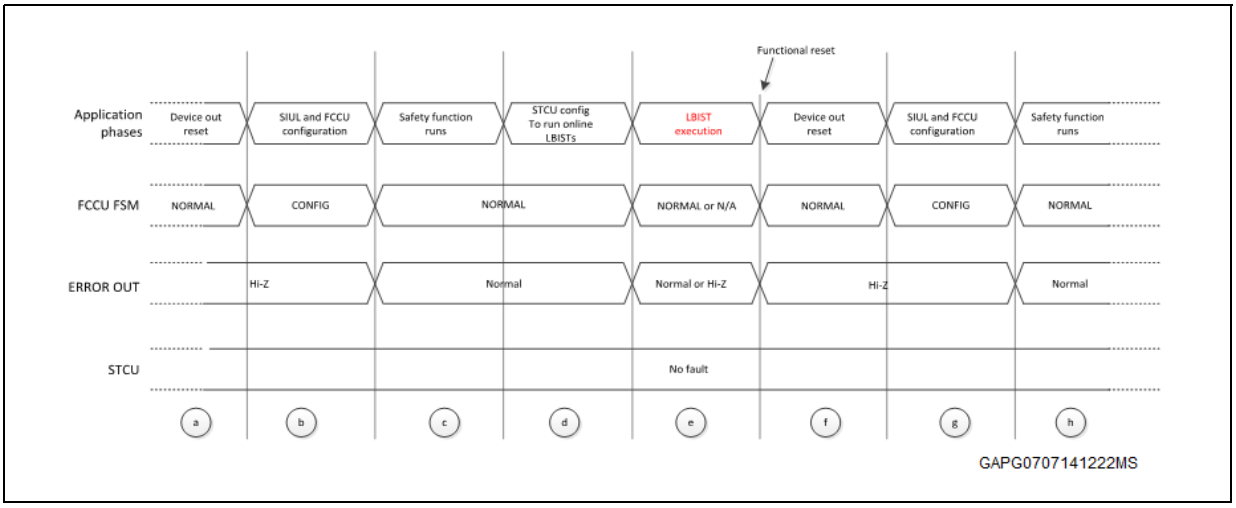
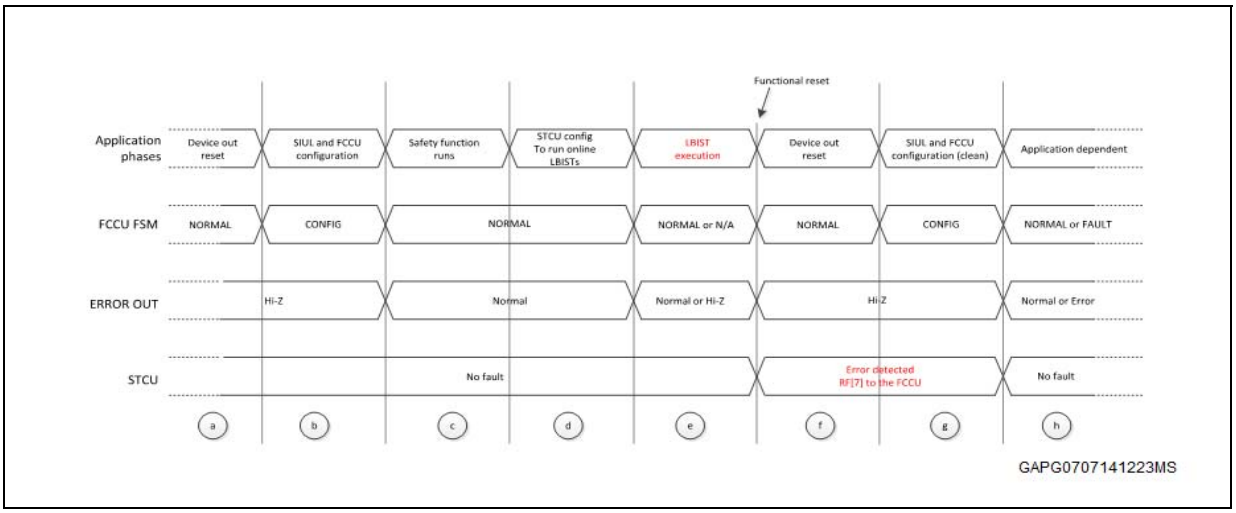


Figure 11. Behavior of the FCCU in case at least a fault is detected by LBIST



Appendix B Reference code for SPC58xGx

In this section it is described the self-test algorithm in online mode applied on SPC58xGx.

Only MBISTs run in online mode.

The software used is:

- Compiler: GHS version 6.1.6
- Debugger: Lauterbach Trace32

MBISTs run from 0 to 101 at 180 Mhz in the IOP core (HSM run at 90 Mhz consequently, i.e. HSM divider fixed by 2).

In order to enable the core of the HSM by Mode Entry a handshake is setup between core 2 (IOP) and core HSM.

The MBIST algorithm is the Reduced Run BIST mode (PMOS and MBU set at 0 in the STCU_CFG register).

```
void main_HSM(void){
      /* Set flag to signal Host that clocks can be changed */
      HSM_HOST.HSM2HTF.B.FLAG31 = 1;

      /* Wait for main platform to be fully initialized */
      while(HSM_HOST.HT2HSMF.B.FLAG31 != 1);

      /* Configure HSM System Bus Interface */
      HSM_SBI.BCR.R = 0x3;    /* Enable Read & Write buffers */
      HSM_SBI.CCR.B.INV = 1;  /* Invalidate SBI cache */
      while(HSM_SBI.CCR.B.INV == 1){
            asm("nop");
      }

      /* Wait for the end of invalidation (66 clock cycles) */
      HSM_SBI.CCR.B.CE = 0;   /* Disable SBI cache */
      HSM_SBI.CCR.B.CE = 1;   /* Enable SBI cache */
      while(1);
}

int main(){
      /* HSM enable */
      MC_ME.CADDR4.R = ((vuint32_t)(&_start_HSM))|1;

      /*! Set HSM Start Address | ME.CADDR[4] */
      MC_ME.CCTL4.R = 0x00FE;

      /*! Execute mode change: Re-enter the DRUN mode to start
      cores, clock tree dividers, PCS, and PLL1 */
```

```
        /*! Write Mode and Key | MC_ME.MCTL */
        MC_ME.MCTL.R = 0x30005AF0;

        /*! Write Mode and Key inverted | MC_ME.MCTL */
        MC_ME.MCTL.R = 0x3000A50F;

        /*! Wait for mode entry complete | MC_ME.GS[S_MTRANS] */
        while(MC_ME.GS.B.S_MTRANS == 1);

        /*! Check DRUN mode entered |MC_ME.GS[S_CURRENT_MODE] */
        while(MC_ME.GS.B.S_CURRENT_MODE != 0x3);

        /* Configure Clocks and Modules Via Mode Entry */
        MC_MODE_INIT();

        /* Set flag in HSM to say all inits are done */
        HSM2HST.HT2HSMF.B.FLAG31 = 1;
        run_MBIST();

        /*------------------- Main While Loop -----------------*/
        /* Main While Loop */
        while(1){
              asm("nop");
        }
}

void run_MBIST(void) {
        STCU2.SKC.R = 0x753F924E; //key1
        STCU2.SKC.R = 0x8AC06DB1; //key2
        /* start MBIST */
        STCU2.CFG.R = 0x10000000;
        STCU2.WDG.R = 0xFFFFFFFA;
        STCU2.MB_CTRLN[0].R = 0x11800000;
        STCU2.MB_CTRLN[1].R = 0x12800000;
        STCU2.MB_CTRLN[2].R = 0x13800000;
        STCU2.MB_CTRLN[3].R = 0x14800000;
        STCU2.MB_CTRLN[4].R = 0x15800000;
        STCU2.MB_CTRLN[5].R = 0x16800000;
        STCU2.MB_CTRLN[6].R = 0x17800000;
        STCU2.MB_CTRLN[7].R = 0x18800000;
        STCU2.MB_CTRLN[8].R = 0x19800000;
        STCU2.MB_CTRLN[9].R = 0x1A800000;
        STCU2.MB_CTRLN[10].R = 0x1B800000;
        STCU2.MB_CTRLN[11].R = 0x1C800000;
        STCU2.MB_CTRLN[12].R = 0x1D800000;
```

```
STCU2.MB_CTRLN[13].R = 0x1e800000;
STCU2.MB_CTRLN[14].R = 0x1f800000;
STCU2.MB_CTRLN[15].R = 0x20800000;
STCU2.MB_CTRLN[16].R = 0x21800000;
STCU2.MB_CTRLN[17].R = 0x22800000;
STCU2.MB_CTRLN[18].R = 0x23800000;
STCU2.MB_CTRLN[19].R = 0x24800000;
STCU2.MB_CTRLN[20].R = 0x25800000;
STCU2.MB_CTRLN[21].R = 0x26800000;
STCU2.MB_CTRLN[22].R = 0x27800000;
STCU2.MB_CTRLN[23].R = 0x28800000;
STCU2.MB_CTRLN[24].R = 0x29800000;
STCU2.MB_CTRLN[25].R = 0x2A800000;
STCU2.MB_CTRLN[26].R = 0x2B800000;
STCU2.MB_CTRLN[27].R = 0x2C800000;
STCU2.MB_CTRLN[28].R = 0x2D800000;
STCU2.MB_CTRLN[29].R = 0x2E800000;
STCU2.MB_CTRLN[30].R = 0x2F800000;
STCU2.MB_CTRLN[31].R = 0x30800000;
STCU2.MB_CTRLN[32].R = 0x31800000;
STCU2.MB_CTRLN[33].R = 0x32800000;
STCU2.MB_CTRLN[34].R = 0x33800000;
STCU2.MB_CTRLN[35].R = 0x34800000;
STCU2.MB_CTRLN[36].R = 0x35800000;
STCU2.MB_CTRLN[37].R = 0x36800000;
STCU2.MB_CTRLN[38].R = 0x37800000;
STCU2.MB_CTRLN[39].R = 0x38800000;
STCU2.MB_CTRLN[40].R = 0x39800000;
STCU2.MB_CTRLN[41].R = 0x3A800000;
STCU2.MB_CTRLN[42].R = 0x3B800000;
STCU2.MB_CTRLN[43].R = 0x3C800000;
STCU2.MB_CTRLN[44].R = 0x3D800000;
STCU2.MB_CTRLN[45].R = 0x3E800000;
STCU2.MB_CTRLN[46].R = 0x3F800000;
STCU2.MB_CTRLN[47].R = 0x40800000;
STCU2.MB_CTRLN[48].R = 0x41800000;
STCU2.MB_CTRLN[49].R = 0x42800000;
STCU2.MB_CTRLN[50].R = 0x43800000;
STCU2.MB_CTRLN[51].R = 0x44800000;
STCU2.MB_CTRLN[52].R = 0x45800000;
STCU2.MB_CTRLN[53].R = 0x46800000;
STCU2.MB_CTRLN[54].R = 0x47800000;
STCU2.MB_CTRLN[55].R = 0x48800000;
STCU2.MB_CTRLN[56].R = 0x49800000;
STCU2.MB_CTRLN[57].R = 0x4A800000;
```

```
STCU2.MB_CTRLN[58].R = 0x4B800000;
STCU2.MB_CTRLN[59].R = 0x4C800000;
STCU2.MB_CTRLN[60].R = 0x4D800000;
STCU2.MB_CTRLN[61].R = 0x4E800000;
STCU2.MB_CTRLN[62].R = 0x4F800000;
STCU2.MB_CTRLN[63].R = 0x50800000;
STCU2.MB_CTRLN[64].R = 0x51800000;
STCU2.MB_CTRLN[65].R = 0x52800000;
STCU2.MB_CTRLN[66].R = 0x53800000;
STCU2.MB_CTRLN[67].R = 0x54800000;
STCU2.MB_CTRLN[68].R = 0x55800000;
STCU2.MB_CTRLN[69].R = 0x56800000;
STCU2.MB_CTRLN[70].R = 0x57800000;
STCU2.MB_CTRLN[71].R = 0x58800000;
STCU2.MB_CTRLN[72].R = 0x59800000;
STCU2.MB_CTRLN[73].R = 0x5A800000;
STCU2.MB_CTRLN[74].R = 0x5B800000;
STCU2.MB_CTRLN[75].R = 0x5C800000;
STCU2.MB_CTRLN[76].R = 0x5D800000;
STCU2.MB_CTRLN[77].R = 0x5E800000;
STCU2.MB_CTRLN[78].R = 0x5F800000;
STCU2.MB_CTRLN[79].R = 0x60800000;
STCU2.MB_CTRLN[80].R = 0x61800000;
STCU2.MB_CTRLN[81].R = 0x62800000;
STCU2.MB_CTRLN[82].R = 0x63800000;
STCU2.MB_CTRLN[83].R = 0x64800000;
STCU2.MB_CTRLN[84].R = 0x65800000;
STCU2.MB_CTRLN[85].R = 0x66800000;
STCU2.MB_CTRLN[86].R = 0x67800000;
STCU2.MB_CTRLN[87].R = 0x68800000;
STCU2.MB_CTRLN[88].R = 0x69800000;
STCU2.MB_CTRLN[89].R = 0x6A800000;
STCU2.MB_CTRLN[90].R = 0x6B800000;
STCU2.MB_CTRLN[91].R = 0x6C800000;
STCU2.MB_CTRLN[92].R = 0x6D800000;
STCU2.MB_CTRLN[93].R = 0x6E800000;
STCU2.MB_CTRLN[94].R = 0x6F800000;
STCU2.MB_CTRLN[95].R = 0x70800000;
STCU2.MB_CTRLN[96].R = 0x71800000;
STCU2.MB_CTRLN[97].R = 0x72800000;
STCU2.MB_CTRLN[98].R = 0x73800000;
STCU2.MB_CTRLN[99].R = 0x74800000;
STCU2.MB_CTRLN[100].R = 0xFF000000;
STCU2.RUNSW.R = 0x00000201;
```

Appendix C Further information

C.1 Reference document

- SPC58xEx/SPC58xGx - 32-bit Power Architecture® microcontroller for automotive ASILD applications (RM0391, DocID027214)

C.2 Acronyms and abbreviations

Table 3. Acronyms and abbreviations

Terms	Meaning
BISTs	Built-In Self-Tests
MBIST	Memory BIST
LBIST	Logic BIST
STCU	Self-Test Control Unit
WDG	Watchdog
RGM	Reset Generation Module
SSCM	System Status and Configuration Module
STCU2	Self-Test Control Unit
FCCU	Fault Collection and Control Unit

Revision history

Table 4. Document revision history

Date	Revision	Changes
09-Jul-2018	1	Initial release.