AN4684
Application note

How to split self-test (Key-On & Key-Off)

Introduction

This application note describes a possible strategy on how to perform self-test on SPC574K72 device.

Safety analysis requires that L/MBIST runs at least once per trip time. The easiest solution is to run all BISTs during the boot phase and it requires almost no software (for example Key-On phase).

In some application the time needed for the execution of all BISTs during the Key-On phase is too long. In order to speed up this phase, a possible solution is splitting the self-tests in two subsets during Key-On (also known as off line mode), and during Key-Off (also known as on line mode).

This document shows how to implement the Key-On/-Off self-tests with an associated example code. Other approaches can be implemented but it depends on the specific needs of the final application.

Obviously the reader knows the usage of self-tests but for further information see (*Section Appendix B: Reference documents*)

Contents

1 Overview 5
1.1 LBIST strategy for Fast Self-Test algorithm 7
1.2 MBIST strategy for Fast self-test algorithm 7
1.3 Fast self -test algorithm use cases 9
1.4 Storing results 10
1.5 Error reaction 11
1.6 Self-test execution integrity 12
1.7 Fast Self-Test algorithm. 12
1.8 Normal Flow (ST_STAT = 0x0000_C1A0) 16
1.8.1 Running the self-test during Key-Off 18
1.8.2 Storing LBIST results (ST_STAT = 0x0000_5555) 19
1.9 Virgin Flash (ST_STAT = 0xFFFF_FFFF) 20
1.10 Key-Off failures (ST_STAT = 0x0000_0A1C) 21
1.11 Unexpected abort 24

2 Summary 25

Appendix A Appendix. 26
A.1 Fast Self-test algorithm (Key-Off) 26

Appendix B Reference documents 41

Revision history 42

List of tables

Table 1. SPC574K72 self-tests use-cases 9
Table 2. Document revision history 42

List of figures

Figure 1. Example of Self-test performed during Key-On and Key-Off 5
Figure 2. Temporal diagram about L/MBISTs performing 6
Figure 3. Fast self-test flow 6
Figure 4. A possible LBIST partition 7
Figure 5. MBIST execution 8
Figure 6. PMOS and MBU bit for MBIST algorithms 8
Figure 7. Data Flash section 10
Figure 8. Data storing in data Flash 11
Figure 9. Online and offline self-test flows 13
Figure 10. Fast self-test algorithm flow 15
Figure 11. Start of fast self test agorithm 16
Figure 12. First check after starting algorithm 16
Figure 13. Reading of ST_STAT value from Flash 16
Figure 14. Normal flow 18
Figure 15. Data storing, case ‘C’ of Fast Self-Test flow 20
Figure 16. Virgin Flash case 21
Figure 17. Key-Off with failures 23
Figure 18. Unexpected abort 24

1 Overview

The *Fast self-test* algorithm described in this document is based on the idea to split the Self-tests in two subsets in order to perform them in two different phases.

The Fast self-Test could be organized as described below:

- LBISTs partitions split in two subsets
  - The 1st subset executed during the Key-On (off line mode)
  - The 2nd subset executed during the Key-Off (on line mode)
- MBISTs are not split
  - To reach a high coverage all MBISTs run during both Key-On and Key-Off, but using different algorithms(a) .

As requirement:

- The first subset of LBIST and all MBIST, which are executed during Key-On (STARTUP), meets the start-up timing constraints. Modules tested in this phase are the ones involved in the boot process (e.g. IOP, Flash controller)
- The second subset of LBIST and all MBIST, which are executed during Key-Off (SHUTDOWN), can take longer time.

Figure 1 describes the composition of these two subsets.

Figure 1. Example of Self-test performed during Key-On and Key-Off

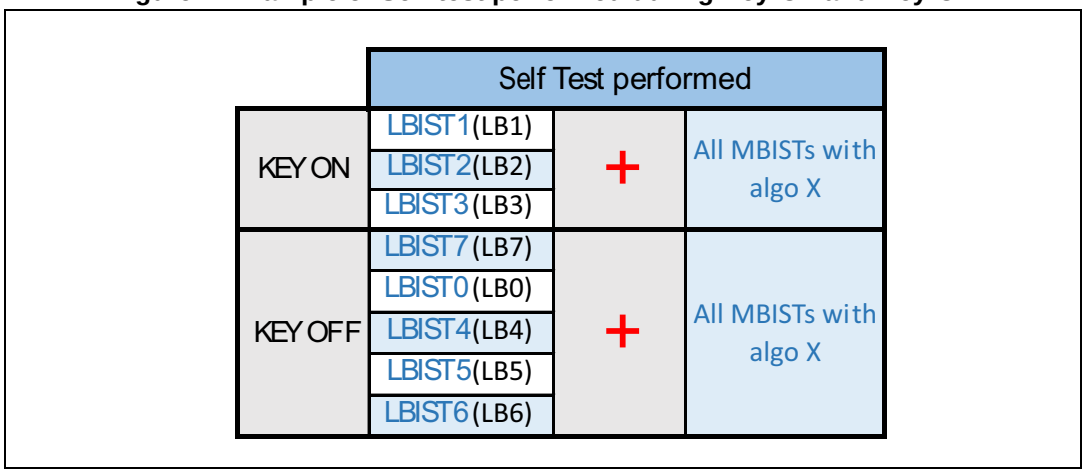
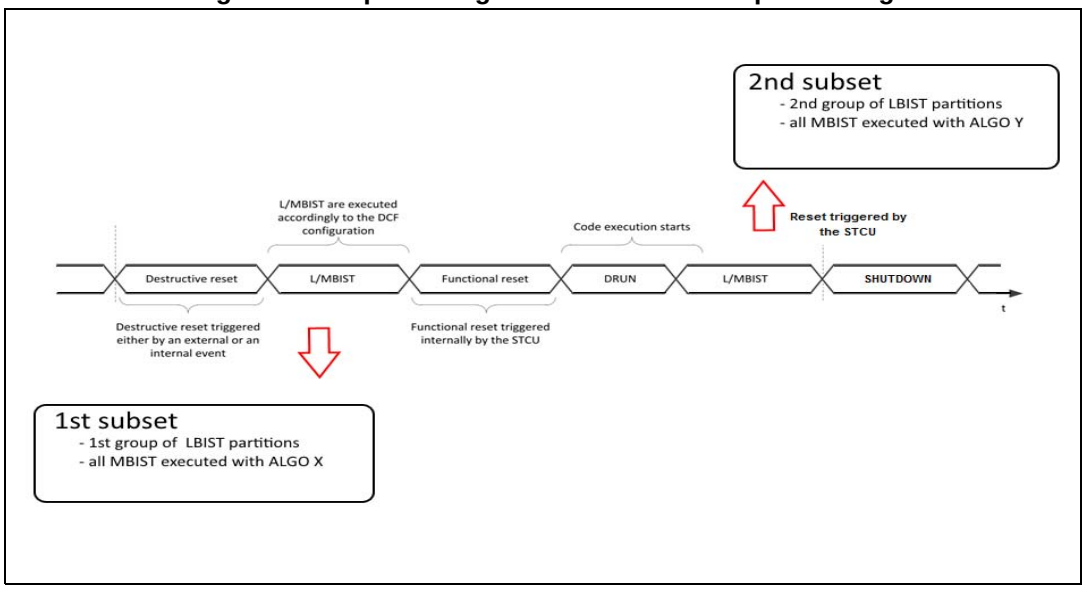


Figure 1 provides some details about how peripherals are split between Key-On and Key-Off phases.

Figure 2 is a temporal diagram that explains better how the two subsets are organized.

a. One algorithm during the Key-On and different one during the Key-Off

Figure 2. Temporal diagram about L/MBISTs performing



The reaction to errors detected by either offline or online self-tests is application dependent. The reference code reported in *Section A.1: Fast Self-test algorithm (Key-Off)* shows an example of possible reaction.

Figure 3. *Fast self-test* flow

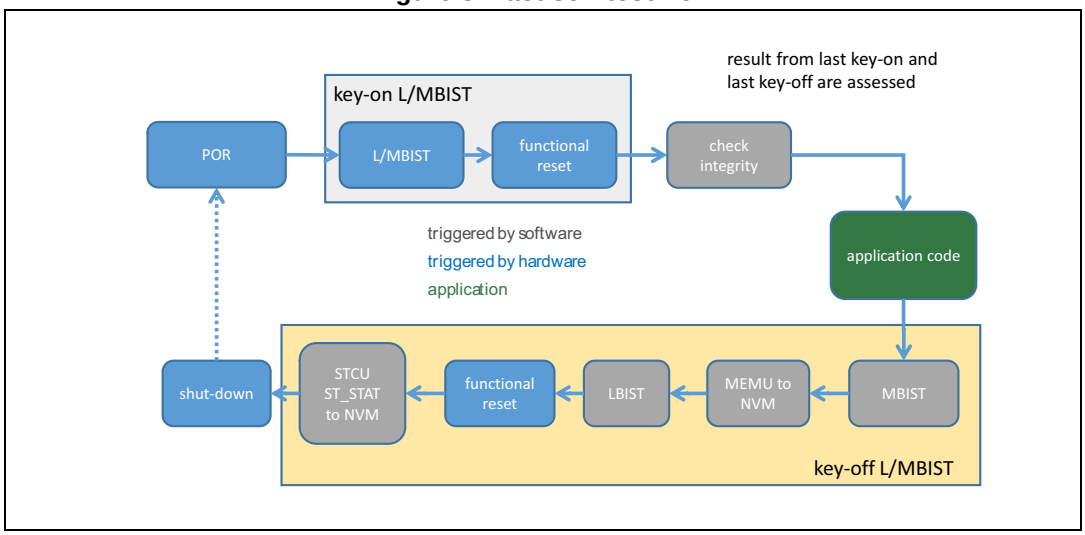


Figure 3 gives additional details on the self-test flow:

- L/MBIST at Key-On performed after POR[b], during the boot phase before the execution of any code[c];
- L/MBIST at Key-Off performed after the execution of the application code, before shutdown.
  After functional reset, LBIST results are saved in the not volatile memory (e.g. data

b. or after a destructive reset.

c. grey block in *Figure 3: Fast self-test flow*

sector of the flash). MBIST results, coming from MEMU, are saved the not volatile memory before LBIST execution[d].

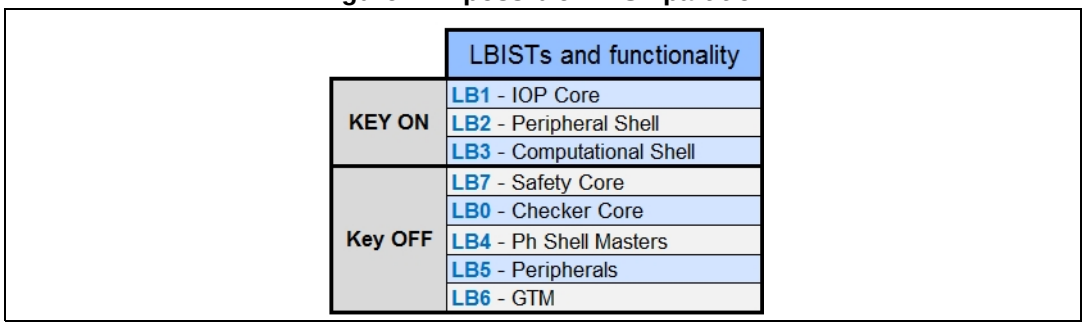
After next POR, startup code checks for errors and prevents the system from starting in case of detected error[e].

1.1 LBIST strategy for *Fast Self-Test* algorithm

SPC574K72 cut 2.0 device includes 8 LBISTs, from 0 to 7. Refer to the reference manual (see RM0334) to have a list of peripherals tested by each LBIST.

Figure 4 shows how different LBISTs are split between Key-On/off.

Figure 4. A possible LBIST partition



This partition depends on some considerations:

1. Integrity of the IOP core (LBIST1) shall be verified during the Key-On, because the IOP executes the BAF and handles error reaction to Key-On/-off error events.
2. Peripheral Shell (LBIST2) contains some modules involved also during the boot phase, such us the slow crossbar used by IOP
3. Computation Shell (LBIST 3) includes the MEMU module[f].
   MEMU saves details about error detected by the MBIST. For this reason, it's recommended to check its integrity during Key-On phase, but before the execution of MBIST[g].
4. In order to speed up the self-test at Key-On, the number of LBIST executions shall be minimized. Only LB1, LB2 and LB3 are executed in this phase.

1.2 MBIST strategy for *Fast self-test* algorithm

The suggested strategy for the MBISTs is not to execute some MBISTs during the Key-On and others during the Key-Off (as done for the LBISTs). The idea is to perform all MBISTs in both phases, but with different available algorithms (see *Figure 4*)

d. yellow block in *Figure 3: Fast self-test flow*.

e. *check integrity* block in *Figure 3: Fast self-test flow*.

f. In the McK device, MEMU module belongs to LBIST5

g. Otherwise MBIST failures found during Key-Off phase cannot be saved by MEMU, because the LBIST3 would destroy the contents of the MEMU.

Figure 5. MBIST execution

	MBIST	Algorithm
KEY ON	All	AutoTest Mode
KEY OFF	All	Reduced Run MBIST

These algorithms can be configured via two flags of STCU_CFG registers, PMOS and MBU.

Figure 6. PMOS and MBU bit for MBIST algorithms

Mode	PMOS bit	MBU bit
Autotest	0	1
	1	1
Reduced RunBIST	0	0
Full RunBIST	1	0

The recommended configuration is:

- Key-On -> Autotest mode: it verifies also the presence of multi bit errors in the RAM arrays
- Key-Off -> two choices:
  - Reduced RunBIST Mode: it verifies also the integrity of the RAM addressing logic, or
  - Full RunBISTMode[h]: It includes the first 2 algorithms

The 3 algorithms showed in *Figure 6: PMOS and MBU bit for MBIST algorithms* are different in terms of covered faults and diagnostic coverage. In general *Autotest mode* has a lower coverage than *RunBIST mode* algorithms.

h. It is not used by the suggested *Fast Self-Test* Algorithm.

1.3 *Fast self -test* algorithm use cases

According to the time available to the application to perform the boot phase, two different uses cases have been analyzed:

- Ultra-short self-test
- Short self-test.

They are different in terms of execution time and diagnostic coverage. In both cases, tests are split between Key-On and Key-Off phases.

Table 1. SPC574K72 self-tests use-cases

Case	Self-test	Num of Patterns/ STCU_LB_PCS	Time budget	Coverage	MBISST mode	LBIST
Ultra Soft	Off-line	0x12 C	8 ms	≥ 80%	Auto test	L1, L2,L3
	On-line		200 ms	≥ 80%	Reduced RunBIST	L7 L0 L4 L5 L6
Short	Off-line	0x1130	28 ms	≥ 90%	Auto test	L1, L2,L3
	On-line		100 ms	≥ 90%	Reduced RunBIST	L7 L0 L4 L5 L6

Table 1: SPC574K72 self-tests use-cases shows the differences between the two cases. They differ mainly in term of executed LBIST vectors (number of patterns contained in STCU_LB_PCS register).

Short case guarantees higher coverage, but higher execution time than Ultra-short case.

In both cases, MBIST time execution during offline is lower than MBIST execution during online mode because used algorithms are different.

In term of clock:

- during *Offline self test*: STCU controls directly the PLL0. Self-test can run up to 40MHz
- during *Online self test*: STCU doesn't control any PLLs, but it only monitors PLL1. SPC574K72Self-test can run up to 160Mhz

All LBIST partition are generally executed in sequential mode in order to avoid any possible IR drop[(i)] issue

Key-Off time budget doesn't include the time needed to save the test result in to the not volatile memory.

i. If multiple LBISTs runs in parallel, many flip-flops change their status at the time. This could cause a drop of voltage.

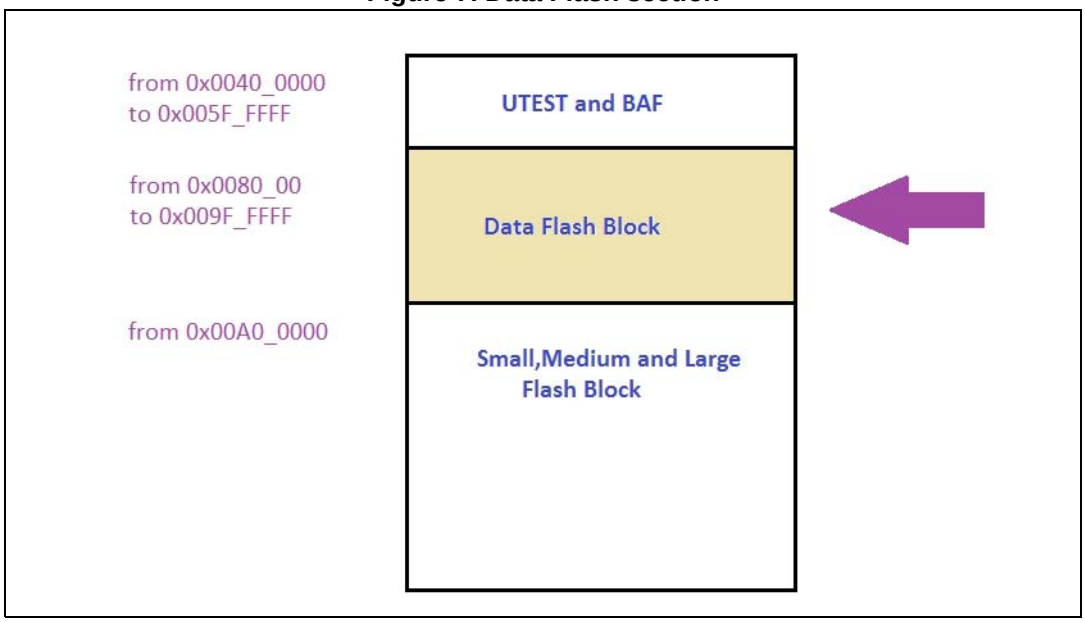
1.4 Storing results

During Key-Off procedure, immediately before shutdown, it is necessary to store the L/MBIST results in flash in order to avoid to miss some information[j].

Storing a single PASSED/NOT PASSED flag is considered not to be enough from a safety point of view. Other details coming from control and monitoring modules, such as STCU and MEMU, shall be stored.

Flash section chosen for storing these details is the DATA FLASH block[k] (see *Figure 7: Data Flash section*).

Figure 7. Data Flash section



Refer to the reference manual to have all details about how to save data into flash (see RM0334).

Writing operation in Flash operates on any Double Word (64-bits).

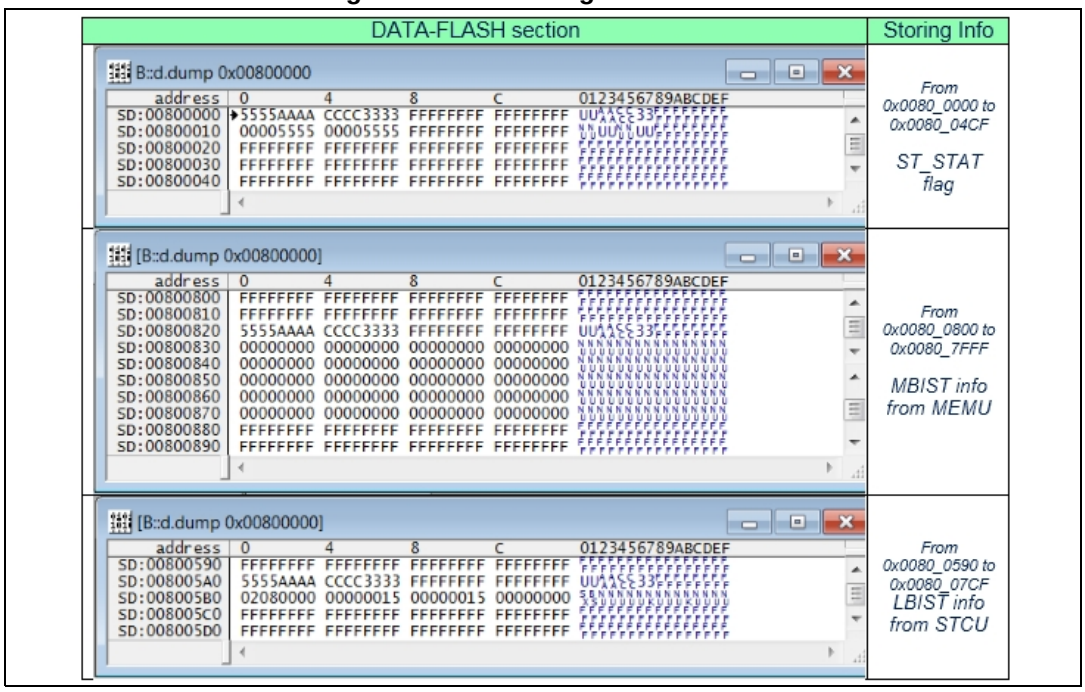
For the *Fast self-test* algorithm, Data Flash section has been virtually split in three parts, to store different information (refer to *Figure 8: Data storing in data Flash*):

- one for ST_STAT flag information (passed/not passed flag)
- one for STCU detailed results after LBIST execution
- one for MEMU detailed results after MBIST execution

j. After shutdown self-test results are not maintained by STCU. Test result are read out of the flash and verified by the software during next code execution (see *Figure 3: Fast self-test flow*).

k. Other Flash sectors can be used.

Figure 8. Data storing in data Flash



ST_STAT is a 64bit field which reports the status of the test. It can assume the following values with the respective meanings:

- 0x0000_5555 - device is ready for Self –Test at Key-Off
- 0xFFFF_FFFF – self-test never executed at Key-Off
- 0x0000_C1A0 - test executed during last Key-Off is “passed”
- 0x0000_0A1C - test executed during last Key-Off is “not passed”.

Result of first MBIST execution is saved from 0x0080_0830 to 0x0080_0870. In Data storing in data FlashMEMU result shows all ‘0’, to indicate that the MBIST is passed without errors.

The Result of first LBIST execution is saved from 0x0080_05A0 to 0x0080_05B0. Values of some STCU registers are copied starting from location 0x0080_05B0[I].

The 0x5555_AAAA_CCCC_3333 is a keyword that indicates the end of the correct writing operation in each single block.

1.5 Error reaction

Key-Off error reactions are application dependent. Hereafter the two possibilities:

- Error reaction by HW via FCCU: in this case external resources are available to receive and process the error indication.
  The FCCU interface is able to indicate eventual occurrence of Failure (RF) during the self-test sequence and, if configured, takes other reaction as reset, interrupt. See

I. STCU registers that are saved are STCU.ERR_STAT, STCU.LBSSW and STCU.LBESW.

Section A.1: Fast Self-test algorithm (Key-Off) for more detail about the application note *SPC574K72_HowToRun_self-Test* (AN4551).

- Error reaction by SW: eventual errors detected during Key-Off phase are managed by software.

The *Fast Self-Test* algorithm given as example uses the reaction by Software.

1.6 Self-test execution integrity

Although the STCU module itself can be source of latent failure, it is not included on LBIST list:

- A CRC[(m)] monitors the internal STCU critical signals during the Self-Test run to increase the STCU2's Self-Checking capability. It is a self-test configuration dependent.
- A programmable Watchdog timer (WDG) to provide a protection mechanism in case of dead-lock or end-less condition during the Self-Test procedures.
  For this reason, its timeout shall be loaded with a value that is a greater than the expected execution time.

Check STCU chapter in the RM to have all needed details.

1.7 *Fast Self-Test* algorithm

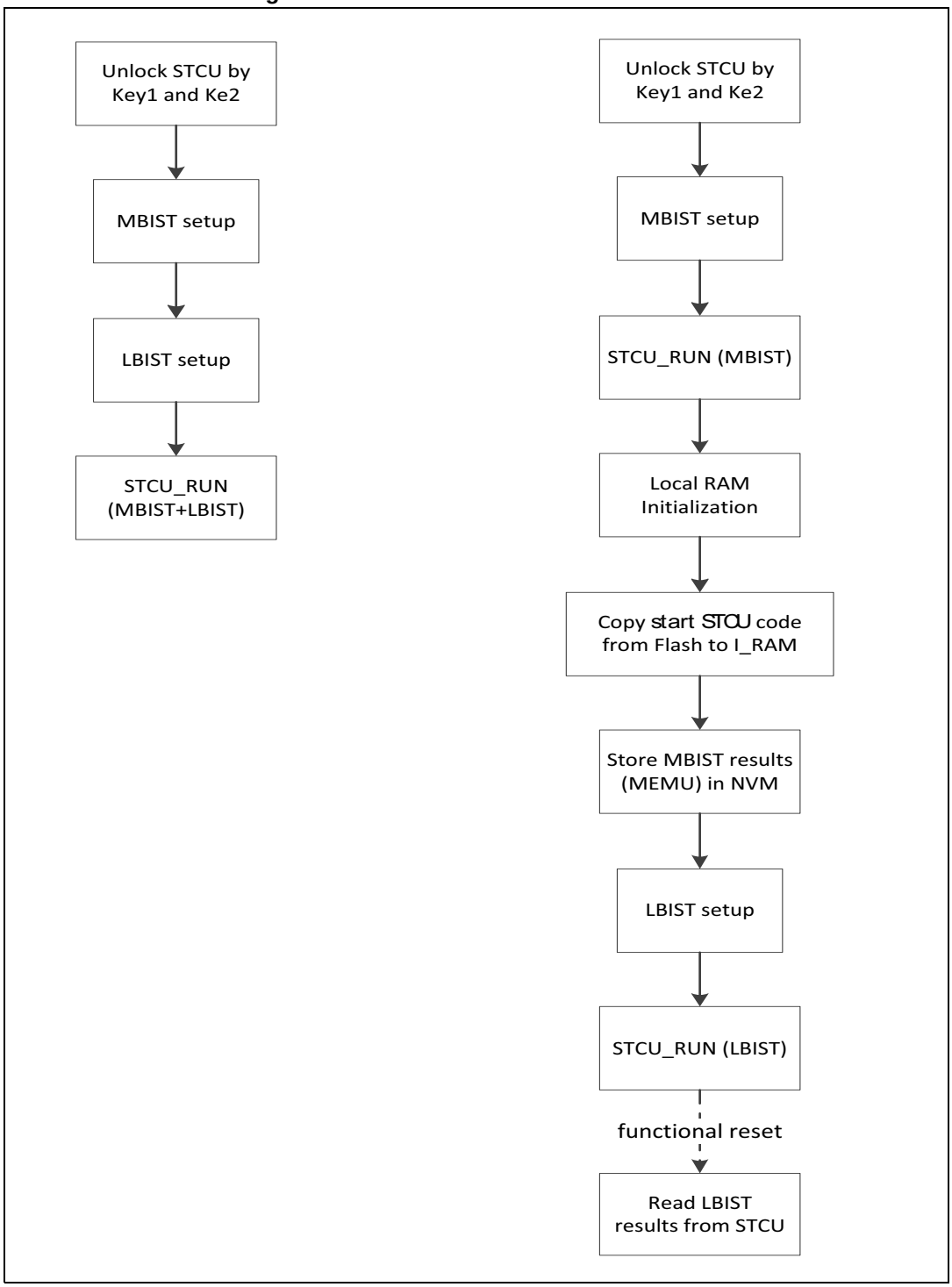
This section describes the *Fast Self-Test* algorithm.

At this point, the reader should have a clear understanding about starting L/MIST in both modes, i.e. online (programming the STCU by SW) and offline (programming STCU by DCF record in the UTEST). Refer to the reference manual and to the SPC574K72_HowToRun_self-Test AN if anything is not clear

Figure 9: Online and offline self-test flows summarizes these concepts.

m. This functionality can be optionally enabled by the CRCEN flag in the STCU_CFG register.

Figure 9. Online and offline self-test flows



Hereafter some considerations related to online mode self-test execution.

1. Password to unlock the STCU, i.e. Key1 and Key2, are different in online and offline mode (see RM for more details)
2. In online mode, the code which starts the *LBIST* is saved in a local memory (I_RAM of Core2).

The reasons is that in some versions of the device, FLASH memory is forced in Power Down as soon any LBIST partition is run. To avoid unwanted behavior, code can't be executed out of the FLASH.

3. Local RAM initialization after MBIST execution: after MBIST, all RAMs under test contain random data. To avoid possible ECC[n] errors, RAMs used to execute the next LBIST must be initialized[o].
4. MEMU results storing: MEMU information related to MBIST result is stored in the NVM to be evaluated after next boot (see *Figure 3: Fast self-test flow*).
5. After LBIST execution, a global functional reset is generated[p].

Figure 9: Online and offline self-test flows represents a very simplified version than the real implementation of the whole Fast Self-Test algorithm. In addition to what is showed in *Figure 10*, the Fast Self-Test algorithm implements the additional actions described below:

1. It checks the results of the Key-On phase before Key-Off performing
2. It checks the result s of the previously Online Self-Test (last Key-Off) if performed
3. It manage eventual errors detected during point 2

Figure 9 shows the flow of *Fast Self-Test* algorithm. 'A', 'B' and 'C' letters indicate the main path of the diagram in case of

- 'A' one or more error found during previous Key-On/Key-Off execution
- 'B' no error have been found in previous Key-On/Key-Off execution
- 'C' Key-Off self-test executed, its result can be stored in the flash.

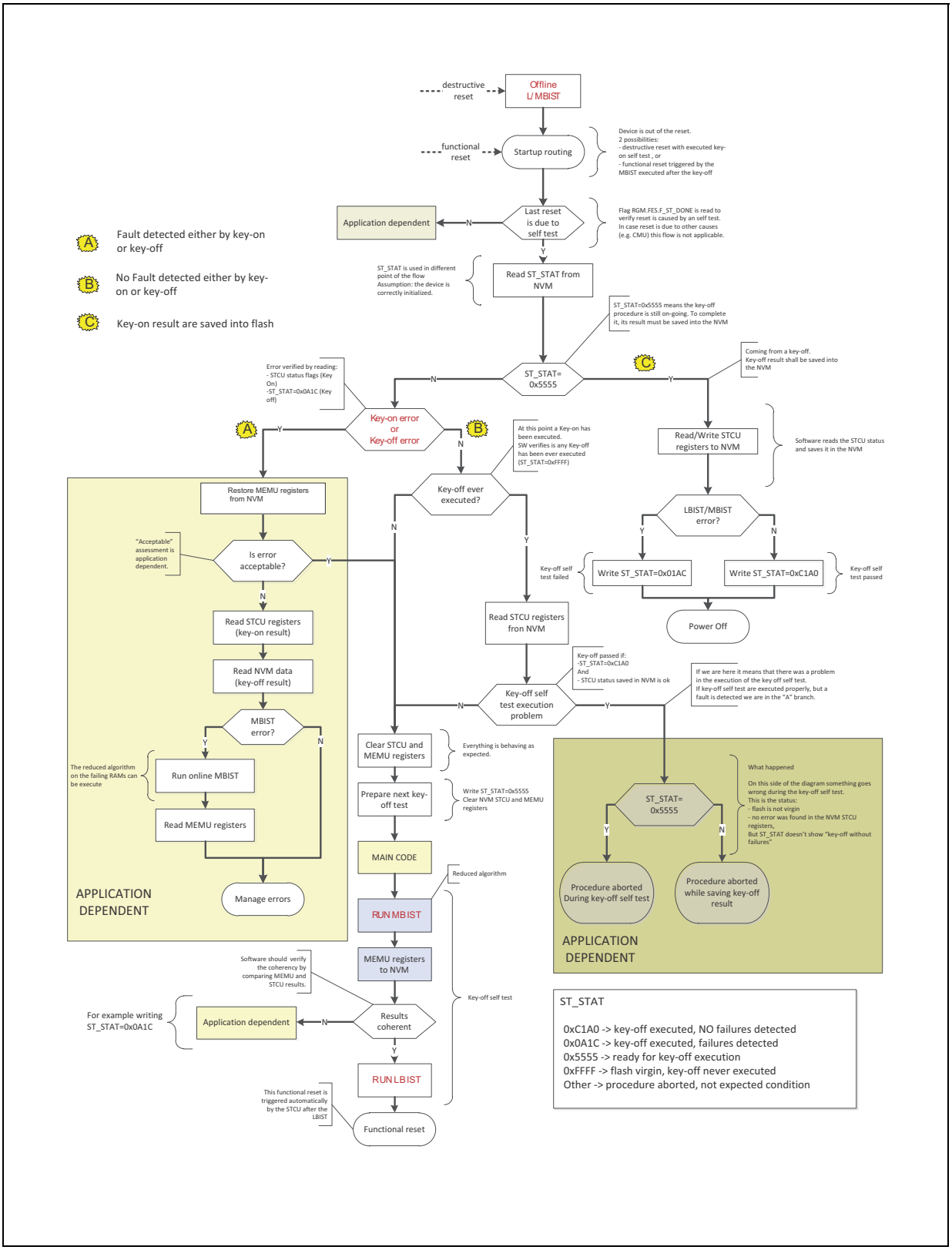
Next sections describe details of this flow.

n. According to the kind of RAM, they support 64-bit data (system ram) + 8-bit of checksum or 32-bit data (local ram) + 7-bit of checksum. After MBIST execution, as data could be not coherent with checksum A write of 64/32 bit shall be done.

o. During the Key-Off, the order of BIST execution is MBIST and after LBIST.

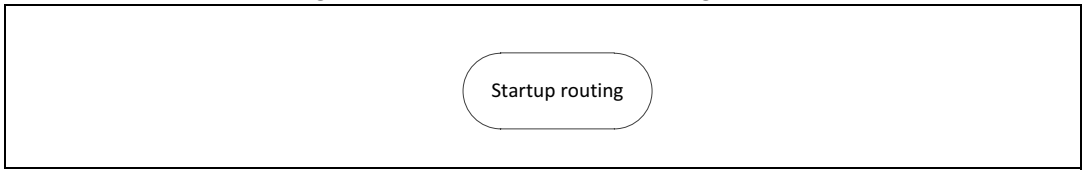
p. Depending of the configuration of STCU_LBRMSW register. This register shall be configured to have the global reset triggered by the STCU.

Figure 10. Fast self-test algorithm flow



Hereafter it is a description of the steps of the flow:

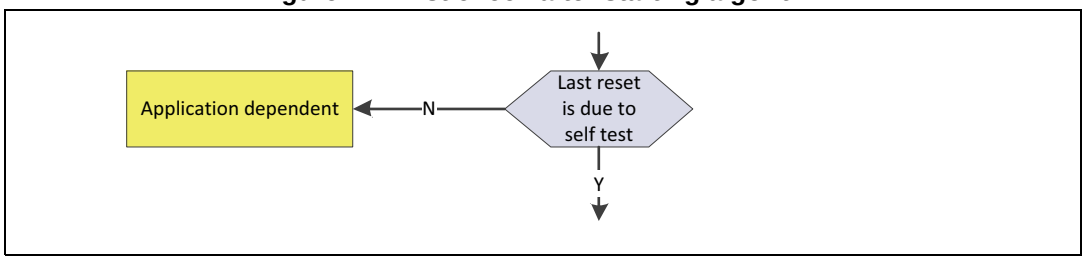
Figure 11. Start of fast self test agorithm



Self-Test software algorithm starts when device is out of the reset due to:

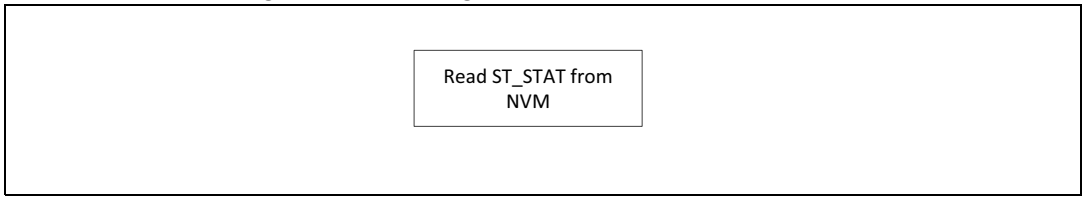
- destructive reset, (in this case the key on self- test is executed) or
- functional reset triggered by STCU after LBIST ending during the previous key-off

Figure 12. First check after starting algorithm



RGM.FES.F_ST_DONE is read in order to verify whether the previous reset is caused by a self-test. This AN focuses only of reset caused by self-test. Reaction to other source of reset are not considered.

Figure 13. Reading of ST_STAT value from Flash



The ST_STAT flag, which is saved in the data flash, is read in order to understand if:

- Key - off algorithm hasn't been performed yet (ST_STAT = 0xFFFF_FFFF).
- Key - off algorithm has been executed and no failures have been detected (ST_STAT = 0x0000_C1A0)
- Key - off algorithm has been executed and some failures have been detected (ST_STAT = 0x0000_0A1C)
- Key-off algorithm has been executed, but LBIST results has not been stored in data flash yet (ST_STAT = 0x0000_5555).

Next sections describe the main branches of the software flow.

1.8 Normal Flow (ST_STAT = 0x0000_C1A0)

Branch ‘B’ in *Figure 10* is part of the normal flow.

In this case, both offline and last online self-tests have been completed and no error failures have been detected(q).

Result of Key-On self-test is verified by reading the following STCU registers:

- STCU2_ERRSTAT (STCU Error register)
- STCU2_LBE (STCU Off-line LBIST End Flag register)
- STCU2_LBS (STCU Off-line LBIST status register)
- STCU_MBEL (STCU Off-line MBIST End Flag low register)
- STCU_MBEM (STCU Off-line MBIST End Flag medium register)
- STCU_MBSL (STCU Off-line MBIST status low register)
- STCU_MBSL (STCU Off-line MBIST status medium register)

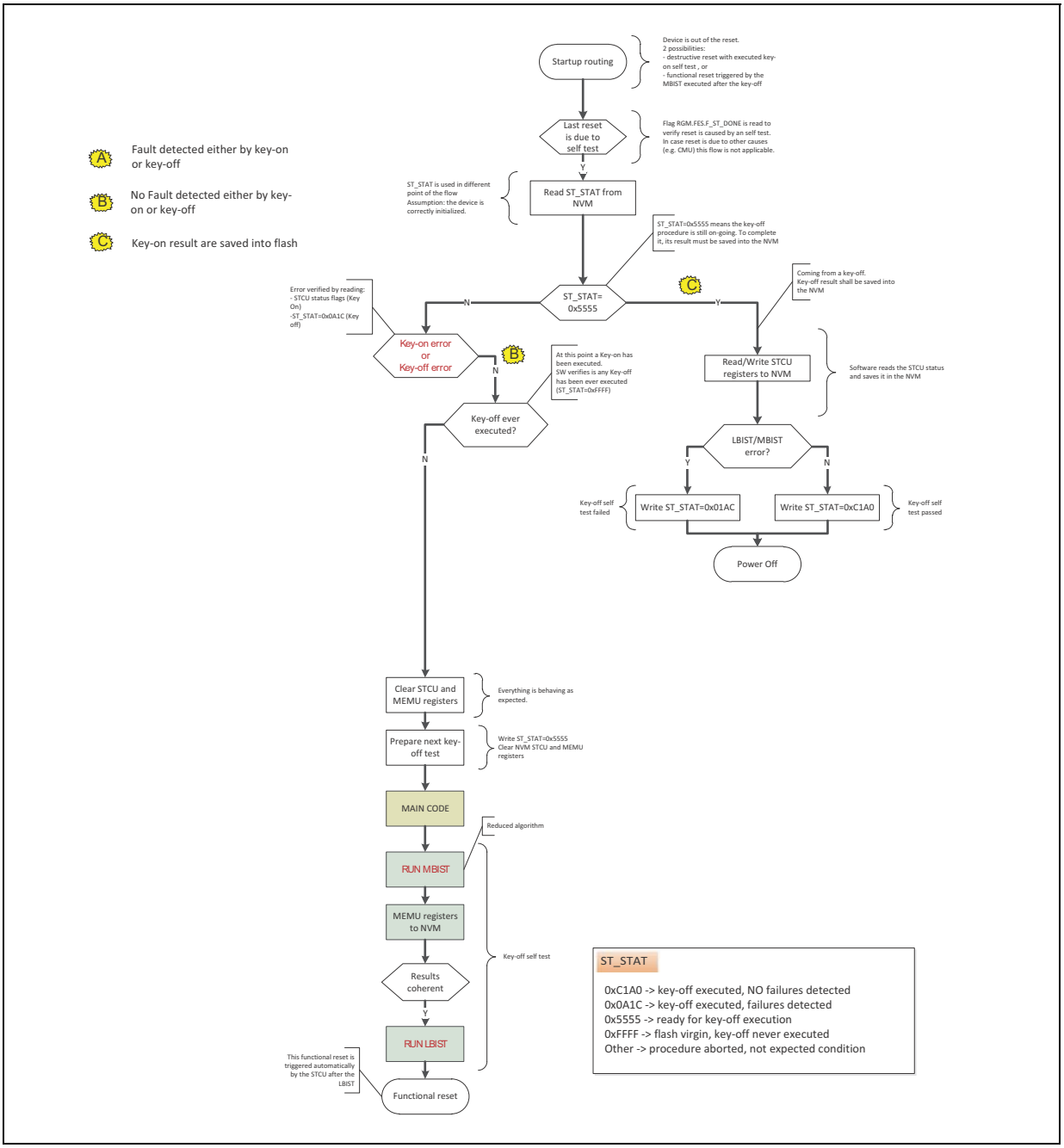
Result of Key-Off self-test is verified by reading the following STCU registers:

- STCU2_ERRSTAT (STCU Error register)
- STCU2_LBESW (STCU On-line LBIST End Flag register)
- STCU2_LBSSW (STCU On-line LBIST status register)
- MEMU registers stored in NVM after MBIST execution:
  - MEMU_ERR_FLAG
  - MEMU_SYS_RAM_CERR_STS[n] n:0 to 10
  - MEMU_SYS_RAM_UNCERR_STS
  - MEMU_SYS_RAM_OFLW[n] n:0 to 3
  - MEMU_PERIPH_RAM_CERR_STS[n] n:0,1
  - MEMU_PERIPH_RAM_UNCERR_STS
  - MEMU_PERIPH_RAM_OFLW

In case no Key-On/Key-Off error has been found, system is ready to execute the application code and afterward to perform next Key-Off self-test (see *Section 1.8: Normal Flow (ST_STAT = 0x0000_C1A0)*).

q. Or Key-Off self test has been never executed yet.

Figure 14. Normal flow



1.8.1 Running the self-test during Key-Off

If software verifies that no error has been detected, it can start configuring the execution of the next Key-Off procedure. Hereafter the next steps:

1. Clearing MEMU registers. MEMU data tables about error found in volatile memories are clean
2. Preparing for next Key-Off test: writing ST_STAT = 0x5555h on specific location of Data flash
3. Executing Main Code. This block represents the application code. After its execution and before shutdown, the online self-test is executed.
4. Running Key-Off self-test:
   a) Running MBIST. In this phase all MBISTsSPC574K72 are executed using a different algorithm respect the Key-On selt test.
   b) Local RAM Initializing and saving structure for info MEMU and starting code of STCU on them (D_RAM2 and I_RAM2). For more detail about flow see Online and offline self-test flows right side)
   c) MEMU registers to NVM.
   Results of the MBIST are stored in the MEMU module. Content of this module is erased during LBIST3 which is executed during the next step. It's important to store in the NVM the MEMU information related to MBIST results. These results are read after next reset.
   d) LBIST running. Only LBIST not executed during Key-On are executed at this step[(r)]SPC574K72.
   NOTE: Before running the LBIST it's important to set all flags of the STCU.LBRMSW register to '1'. In this way, a global functional reset is automatically triggered at the end of LBIST execution.

1.8.2 Storing LBIST results (ST_STAT = 0x0000_5555)

After global reset, algorithm goes toward branch 'C' of *Figure 10*.

Software verifies that RGM.FES.F_ST_DONE flag=1 and the ST_STAT field is equal to 0x5555[(s)].

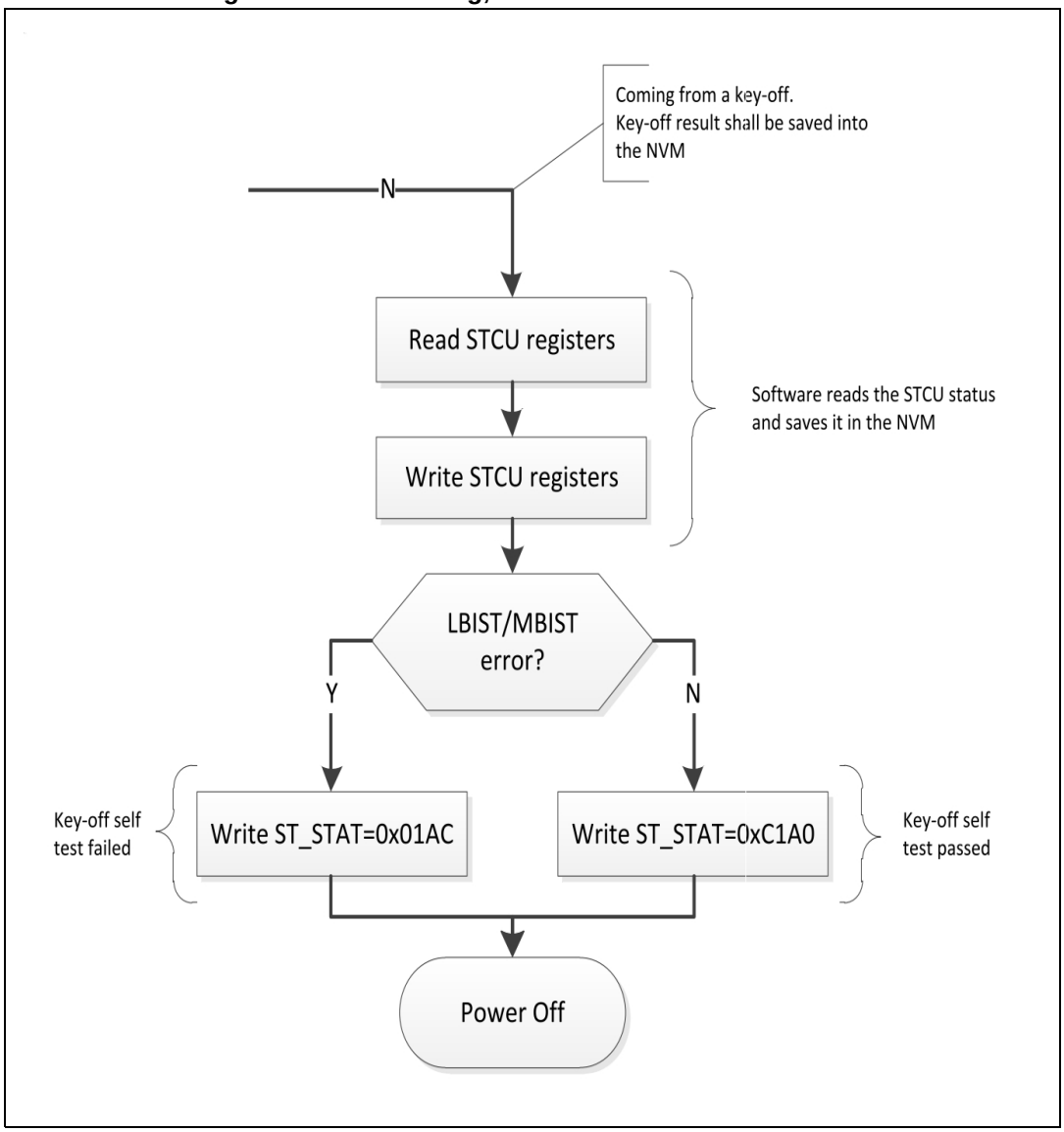
The result of the online LBIST is stored in the NVM. Afterwards software verifies whether any error is detected by either the MBIST or LBIST. If at least an error is found, the new ST_STAT value saved in flash is 0xC1A0h, otherwise 0x01AC.

At this point the system can shut down the microcontroller.

r. for K2 cut2.2 in online mode they are:LB7,LB0,LB3…LB6

s. ST_STAT =0x5555 means that the Key-Off procedure has not been completed yet. In this case results need to the be saved into the not volatile memory.

Figure 15. Data storing, case 'C' of *Fast Self-Test* flow



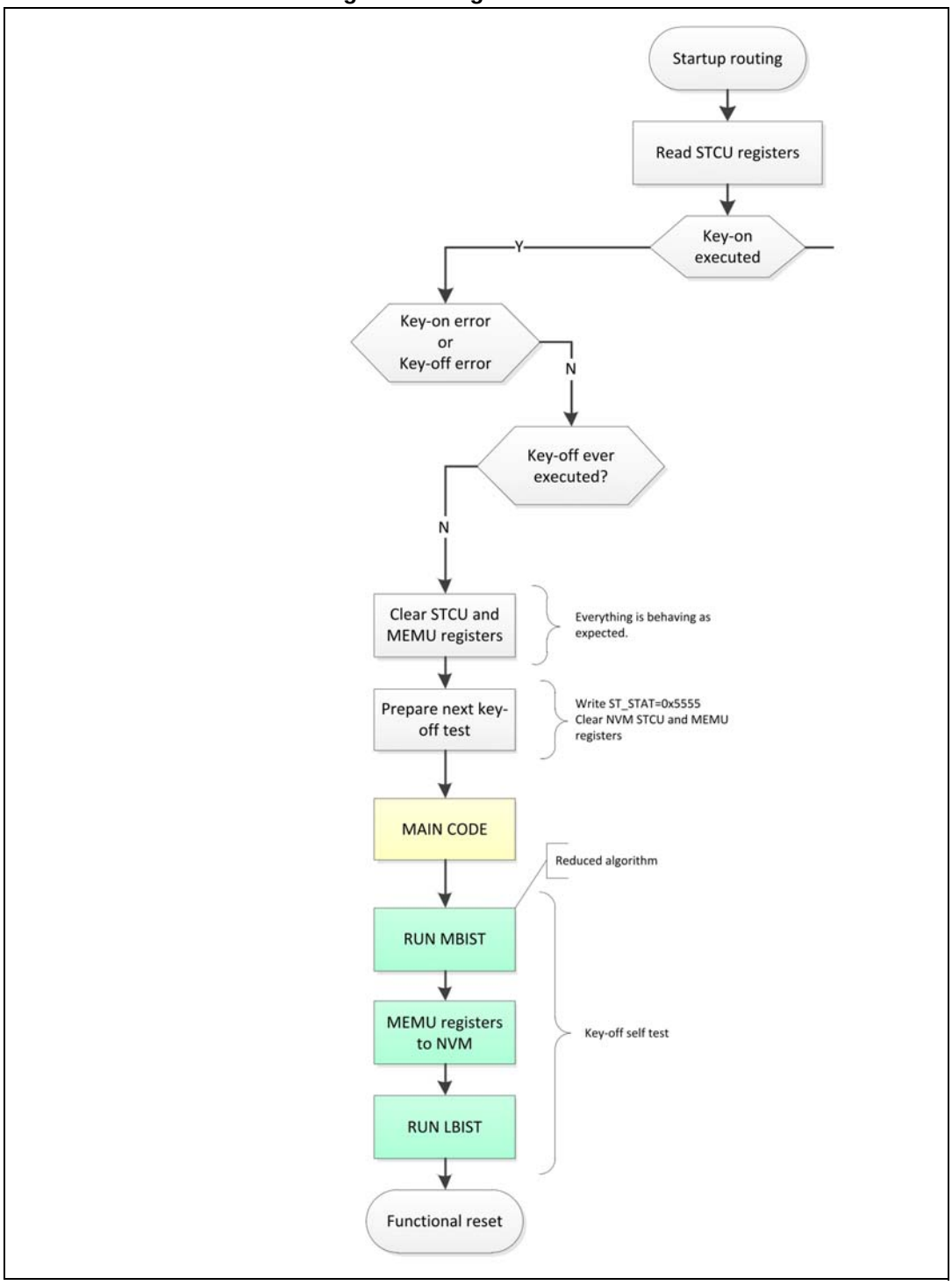
1.9 Virgin Flash (ST_STAT = 0xFFFF_FFFF)

The Virgin flash case is a "sub-case" of branch 'B' (Virgin Flash case). It represents on the situation in which no Key-Off self -est has ever run. Key-Off algorithm has to be performed for the first time and data sector related to the self test is empty

Next steps consist of reading the STCU register results to understand how is gone the Key-On test performed in offline.

Basically test can continue as branch 'B' (see*Section 1.8.1: Running the self-test during Key-Off* section).

Figure 16. Virgin Flash case



1.10 Key-Off failures (ST_STAT = 0x0000_0A1C)

This section of the flow is executed when one or more errors are found during last executed Key-On/-off self-tests (i.e. the branch ‘A’ of *Figure 10: Fast self-test algorithm flow*).

Software can discriminate if errors are detected by either Key-On or Key-Off self-test, the user shall read the STCU registers and the ST_STAT saved in the data flash.

In details, to verify whether errors have been detected by the online self-test, the ST_STAT field is read out of the NMV:

- 0xC1A0h means last executed Key-Off self-test passed
- 0x01Ach means last executed Key-Off self-test failed.

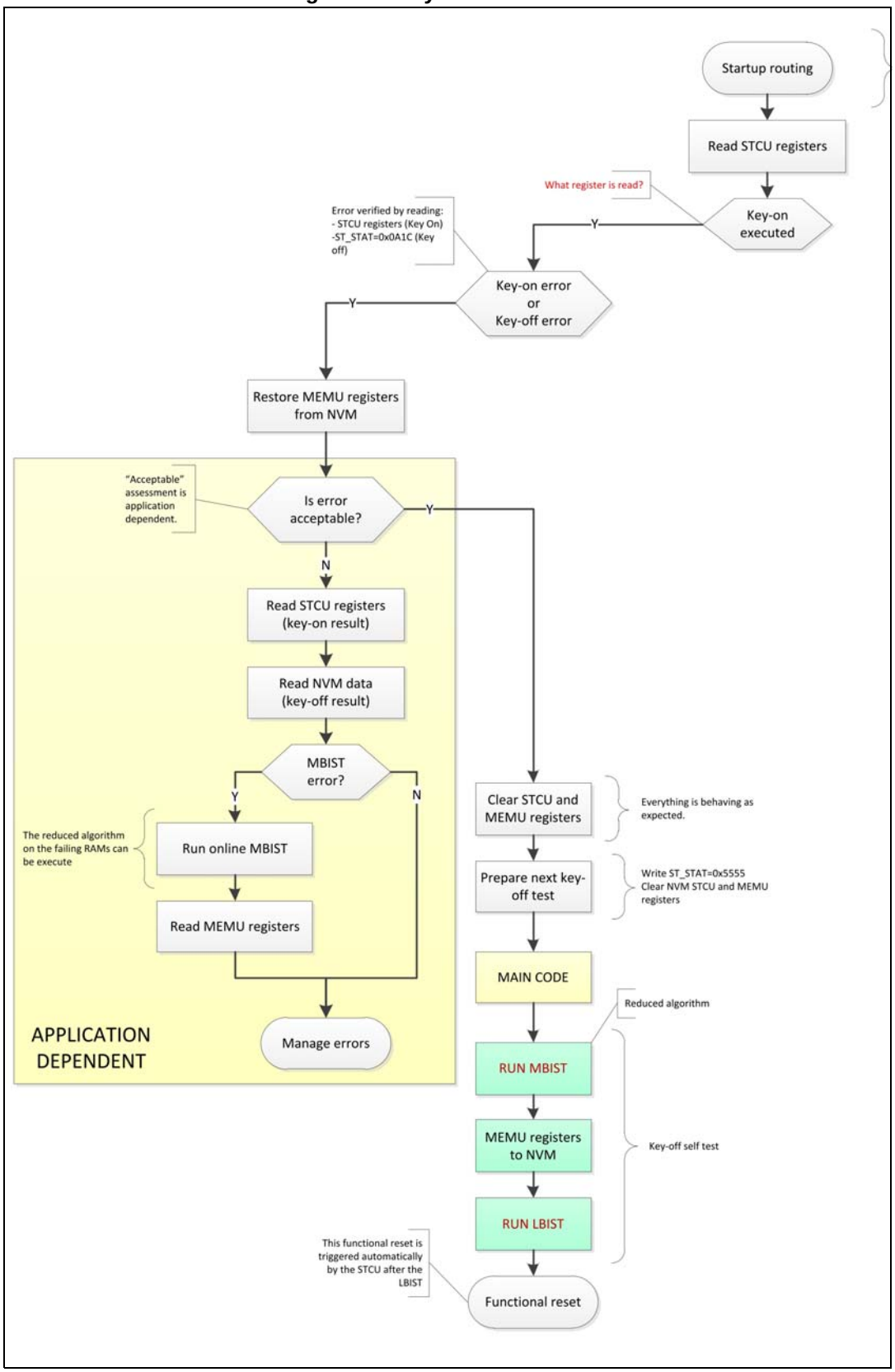
To verify whether an error has been detected by the offline self-test, software shall read the STCU result registers. Firstly the MBIST and LBIST end flag register are read (STCU.LBE, STCU.MBEL, STCU.MBEM and STCU.MBEH). If BISTs have been successfully performed, MBIST and LBIST status flag registers are set to '1' (STCU.LBS, STCU.MBSL, STCU.MBSM and STCU.MBSH).

At this point, if some errors have been found during Key-On e/o previous Key-Off, MEMU registers from Data Flash are restored.

Reaction to faults detected by L/MBIST is an application dependent. For example, software can quantify how many MBIST errors are detected. If this number is still acceptable (for example under specific threshold value), flow follows the steps described in section *Section 1.8.1: Running the self-test during Key-Off*. Otherwise MBIST test is executed again[t] and the application takes the proper reaction (*Figure 17: Key-Off with failures*).

t. MBIST is re-executed to verify if detected errors are caused by either a permanent or a transient failures.

Figure 17. Key-Off with failures



1.11 Unexpected abort

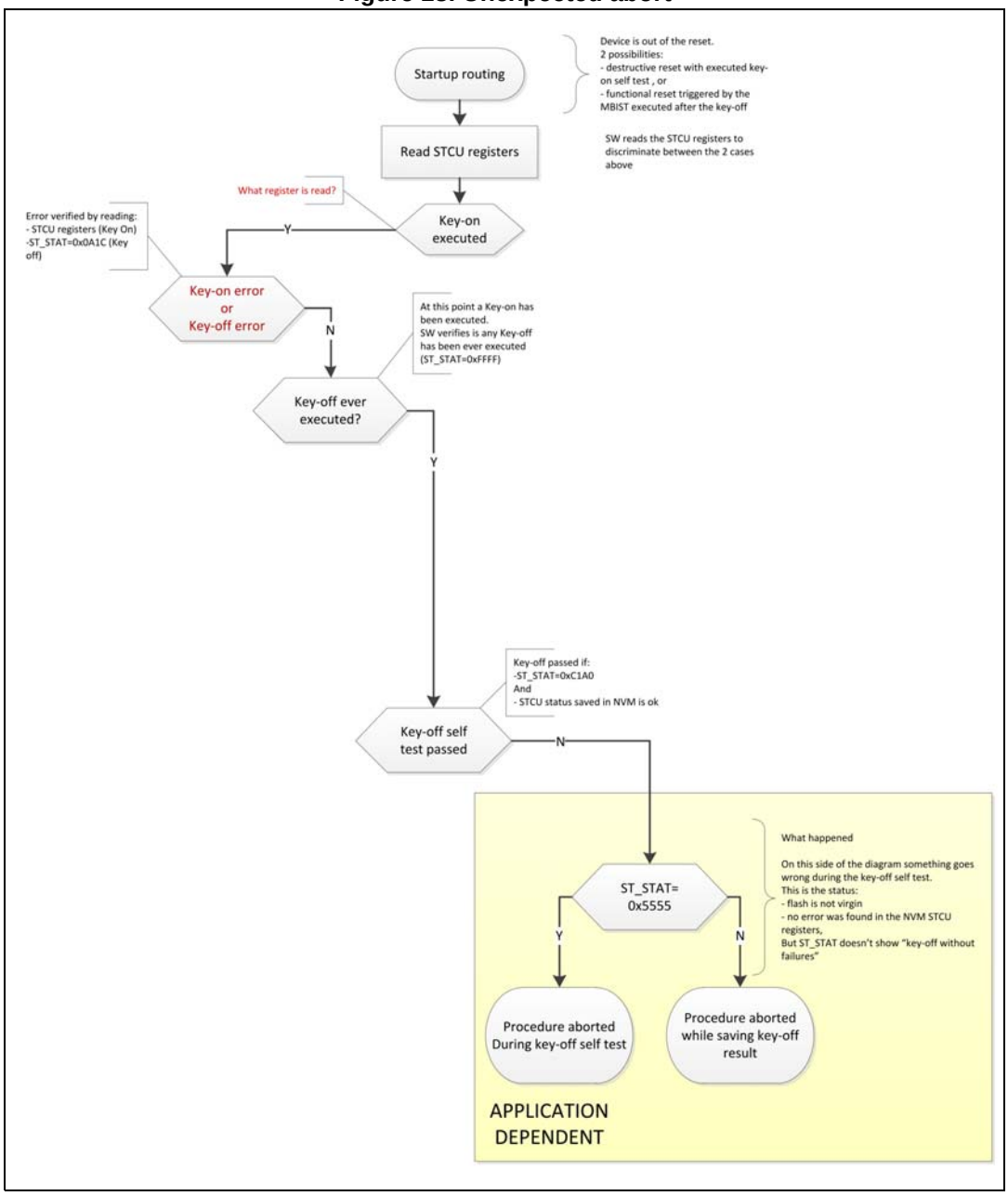
Fast Self-Test Algorithm should manage the corner case in which the Key-Off self-test is aborted while running, for example because the supply is removed before the end of the Key-Off procedure.

Reaction to an aborted online self-test is the application showed in *Figure 18*.

In order to understand when test was aborted, the ST_STAT flag in the data flash is read.

If ST_STAT = 0x5555h, it means that procedure aborted either during Key-Off self-test execution or while saving Key-Off results.

Figure 18. Unexpected abort



2 Summary

Depending on the requirements of the application, running the full self-test during the boot phase may require a time larger than the available one.

This AN describes a solution on how to implement the full self-test in case a short boot time is required. This algorithm is based on two important strategies:

- splitting LBIST execution in two parts, one executed during Key-On and one during Key-Off
- performing two different algorithms for MBIST execution: Autotest mode during Key-On and Reduced RunBIST Mode during Key-Off.

Appendix A Appendix

In this section is reported code used to implement *Fast Self* algorithm on SPC574K72 cut 2.2, reporting the more significant subroutines used. For safety reasons it is suggested to run the application using the safety core, that is core_0.

In addition it is provided the script to be used during Key-On to program STCU register by DCF in the UTEST.

A.1 Fast Self-test algorithm (Key-Off)

```
volatile uint32_t u32DebugFlag = 0;
volatile trKey-Onstatus rKey-Onstatus;
volatile trKey-Offstatus rKey-Offstatus;
int main(){
   init_status();
   save_RGM_FCCU_status();
   while(u32DebugFlag!=0xCCAA) { asm("nop"); };
   init_resetEscalation();
   rKey-Onstatus.bKey_ON_executed = 0;   /* var initialization */
   check_Key_ON_status();
   MCU_Init();        /* Configure Clocks and Modules via Mode Entry */
   FCCU_Init();
   UNLOCK_Flash();
   KEY_OFF_Routine();
   LOCK_Flash();

   while(1){
        DELAY(2000000);
        OUTPAD_Toggle(PAD_LED_CORE2);   /* PC[10]  */
   }
}
/***** init_status *******/
//! It initializes the rKey-Onstatus variable defined in a specific RAM
section
/*!
\return void
*/
void init_status(void) {
rKey-Onstatus.bKey_ON_executed = 0;
rKey-Onstatus.bEnd_Key-On = 0;
rKey-Onstatus.u8Fail_Key-On = 0;
rKey-Offstatus.bEnd_Key-Off = 0;
rKey-Offstatus.bFail_LBIST_Key-Off = 0;
STCU_register_value = LBIST_1_2_3;
```

```
STCU_register_valueSW = LBISTSW_0_4upto7;
}
/***** init_resetEscalation*******/
//! A destructive reset will happen if the RGM_FRET register isn't written
to beforehand.
//! 'F' is the number of functional resets which will cause
/*!
\return void
*/
void init_resetEscalation(void) {
        RGM_FRET = 0x0F000000;
}
/***** key_off_routine *******/
//! This is the main algorithm.
//!Some part of it is application dependent
/*!
\return void
*/
void KEY_OFF_Routine(void)    {

tU8 u8MEMU_LIST_error = 0,  u8threshold = 3,u8Error_offline = 0,
u8_MBIST_error = 0, u8num_errors = 0, u8num_LBIST_fails = 0;


tU32 u32current_ST_STAT;

/*as first time is necessary to check the NVM in order to understand the
value of ST_STAT
ST_STAT = 0x5555 -  it means that Key-Off procedure has been performed and
writing procedure it will follow
ST_STAT = 0xFFFF - it means that Key-Off procedure was never executed
ST_STAT = 0xC1A0 - it means that  last Key-Off was ok
ST_STAT = 0x0A1C - it means that last in the Key-Off procedure some errors
were found */



if (rKey-Onstatus.bKey_ON_executed==1) {
    /*read -ST FLAGG from NVM*/
    u32current_ST_STAT = read_ST_STAT(ST_STAT_BLOCK);
    if (u32current_ST_STAT == 0x5555)  {
       u8num_LBIST_fails = storing_LBIST_results_OnNVM();
       clear_MEMU_registers();
       startMBIST_byFlash(u8num_LBIST_fails);
    }
    else {
           read_STCU_registers();
```

```
          if ((rKey-Onstatus.u8Fail_Key-On != 0) || (u32current_ST_STAT ==
0x0A1C)) {

/* START APPLICATION DEPENDENT CODE */
/* At this point device has detected a failure during either Key-On or Key-
Off self-test.
The reaction is application dependent. An acceptance test can be used to
decide how to read.
As example a very basic threshold acceptance test is implemented.  */

          u8num_errors = cont_errors(u32current_ST_STAT,
                         rKey-Onstatus.u8Fail_Key-On);
          if (u8num_errors>u8threshold) {/*it means error isn't acceptable
*/
             ST_error = read_MBIST_Key-On_register();
             if (u32current_ST_STAT != 0xFFFF)
                   u8MEMU_LIST_error = restore_MEMU_registers();
                 u8_MBIST_error = u8_MBIST_error + u8MEMU_LIST_error;
                   /*read NVM data*/
                   if (u8_MBIST_error > 0)
                              MBIST_online();
                   manage_MBISTerror();
/* END APPLICATION DEPENDENT CODE */
          }
          else {
             WriteBlock(ST_STAT_BLOCK, 0x00005555);/* ST_STAT = 0x5555 -
ready for  Key-Off ST */
             startLBIST_byDRAM()
          }
       }
else {
       if (u32current_ST_STAT != 0xFFFF) {
                rKey-Offstatus.bFail_LBIST_Key-Off = read_flash_section();
/*read STCU register for Key-Off procedure*/
                 if ((u32current_ST_STAT == 0xC1A0) && (rKey-
Offstatus.bFail_LBIST_Key-Off == 0)) {
                      WriteBlock(ST_STAT_BLOCK, 0x00005555);/* ST_STAT =
0x5555 - ready for Key-Off ST */
                      startLBIST_byDRAM();
                 } else key_off_abort_routine();
       }
       else {
             WriteBlock(ST_STAT_BLOCK, 0x00005555);/* ST_STAT = 0x5555 -
ready for Key-Off ST */
             startLBIST_byDRAM();
      }
  }
```

```
 }
} else  /* bKey_ON_executed=0 */
        asm ("nop");
      asm("nop");
}
/***** startLBIST_byDRAM *******/
//! LIBIST in on line mode (Key-Off) are being started by DRAM2 section.
/*!
\return void
*/
void startLBIST_byDRAM(void){
          STCU_Init(0xFFDDEEAA);           /*parameter is size of window
watchdog*/
          RUN_LBIST();
 }
/***** startMBIST_byFlash *******/
//! Results of MBIST are being stored in MEMU before executing LBIST.
//! In STCU_Init is set the watchdog window size
/*!
\return void
*/
void startMBIST_byFlash(tU8 fails){
tU8 num_lbist_fails = 0;

          num_lbist_fails = fails;
          STCU_Init(0xFFFEFFFA);
          RUN_MBIST();
          DELAY(1000000);
          SRAM_initialization();
          writeMEMU_in_NVM(num_lbist_fails);

 }
/***** RUN_LBIST *******/
//! Here is defined clock configuration and the list of LBIST to be run.
//!They are 0,2 and 4 in sequential mode.
/*!
\return void
*/
void RUN_LBIST(void) {


     STCU2.CFG.R = 0x005A0008; //MBU 1
     STCU2.LB[0].LB_CTRL.R = 0x04035410;  /* Link to LBIST4 - sequential*/
     STCU2.LB[0].LB_PCS.R = 0x00000380;
     STCU2.LB[0].LB_MISRELSW.R = 0x20FCE2E2;
```

```
      STCU2.LB[0].LB_MISREHSW.R = 0xF20508A0;
STCU2.LB[4].LB_CTRL.R = 0x05035410;
      STCU2.LB[4].LB_PCS.R = 0x00000380;
      STCU2.LB[4].LB_MISRELSW.R = 0xFCBB5F2D;
      STCU2.LB[4].LB_MISREHSW.R = 0xD79E356B;

      STCU2.LB[5].LB_CTRL.R = 0x06035410;
      STCU2.LB[5].LB_PCS.R = 0x00000380;
      STCU2.LB[5].LB_MISRELSW.R = 0x2C9A3E88;
      STCU2.LB[5].LB_MISREHSW.R = 0x6DDD0220;

     STCU2.LB[6].LB_CTRL.R = 0x07035410;
      STCU2.LB[6].LB_PCS.R = 0x00000380;
      STCU2.LB[6].LB_MISRELSW.R = 0x6A99E0F0;
      STCU2.LB[6].LB_MISREHSW.R = 0x3FE47B6E;

      STCU2.LB[7].LB_CTRL.R = 0x7F035410;
      STCU2.LB[7].LB_PCS.R = 0x00000380;
      STCU2.LB[7].LB_MISRELSW.R = 0xD6D336FB;
      STCU2.LB[7].LB_MISREHSW.R = 0x1E0F68D8;

      STCU2.LBRMSW.R = 0xF1;

{
void (* volatile pStart_STCU)(void) = Start_STCU;
pStart_STCU();
}

}

/***** SRAM_initialization *******/
//! After the MBIST test the RAM contains random data. Before LBIST
execution, it's necessary to
//! initialize the RAM
/*!
\return void
*/
void SRAM_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 */

        dram2_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 dram2_init_loop;  /* Loop for all of SRAM */


         #pragma endasm
}
/***** 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)  {

    STCU2.SKC.R = 0x753F924E;   //key1
    STCU2.SKC.R = 0x8AC06DB1;   //key2
    STCU2.ERR_FM.R = 0x00000000; // NCF faults

    STCU2.WDG.R = window_size;

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

```
void RUN_MBIST(void) {

     STCU2.CFG.R = 0x105A0008;  /* set Initial MBIST pointer - #16
*/
      STCU2.MB[0].MB_CTRL.R = 0x915A5A5A;
      STCU2.MB[1].MB_CTRL.R = 0x925A5A5A;
      STCU2.MB[2].MB_CTRL.R = 0x935A5A5A;

      STCU2.MB[3].MB_CTRL.R = 0x945A5A5A;
      STCU2.MB[4].MB_CTRL.R = 0x955A5A5A;
      STCU2.MB[5].MB_CTRL.R = 0x965A5A5A;
      STCU2.MB[6].MB_CTRL.R = 0x975A5A55;
      STCU2.MB[7].MB_CTRL.R = 0x985A5A5A;
      STCU2.MB[8].MB_CTRL.R  = 0x995A5A5A;
      STCU2.MB[9].MB_CTRL.R = 0x9A5A5A5A;
      STCU2.MB[10].MB_CTRL.R = 0x9B5A5A5A;

      STCU2.MB[11].MB_CTRL.R = 0x9C5A5A5A;
      STCU2.MB[12].MB_CTRL.R = 0x9D5A5A5A;
      STCU2.MB[13].MB_CTRL.R= 0x9E5A5A5A;

 STCU2.MB[14].MB_CTRL.R = 0x9F5A5A5A;
      STCU2.MB[15].MB_CTRL.R = 0xA05A5A5A;
      STCU2.MB[16].MB_CTRL.R = 0xA15A5A5A;
      STCU2.MB[17].MB_CTRL.R = 0xA25A5A5A;
      STCU2.MB[18].MB_CTRL.R = 0xA35A5A5A;
      STCU2.MB[19].MB_CTRL.R = 0xA45A5A5A;
      STCU2.MB[20].MB_CTRL.R = 0xA55A5A5A;
      STCU2.MB[21].MB_CTRL.R = 0xA65A5A5A;
      STCU2.MB[22].MB_CTRL.R = 0xA75A5A5A;
      STCU2.MB[23].MB_CTRL.R = 0xA85A5A5A;
      STCU2.MB[24].MB_CTRL.R = 0xA95A5A5A;
      STCU2.MB[25].MB_CTRL.R = 0xAA5A5A5A;
      STCU2.MB[26].MB_CTRL.R = 0xAB5A5A5A;
      STCU2.MB[27].MB_CTRL.R = 0xAC5A5A5A;
```

```
      STCU2.MB[28].MB_CTRL.R = 0xAD5A5A5A;
      STCU2.MB[29].MB_CTRL.R = 0xAE5A5A5A;
      STCU2.MB[30].MB_CTRL.R = 0xAF5A5A5A;
      STCU2.MB[31].MB_CTRL.R = 0xB05A5A5A;

      STCU2.MB[32].MB_CTRL.R = 0xB15A5A5A;
      STCU2.MB[33].MB_CTRL.R = 0xB25A5A5A;

      STCU2.MB[34].MB_CTRL.R = 0xB35A5A5A;
      STCU2.MB[35].MB_CTRL.R = 0xB45A5A5A;
      STCU2.MB[36].MB_CTRL.R = 0xB55A5A5A;
      STCU2.MB[37].MB_CTRL.R = 0xB65A5A5A;
      STCU2.MB[38].MB_CTRL.R = 0xB75A5A55;
      STCU2.MB[39].MB_CTRL.R = 0xB95A5A55;

      STCU2.MB[41].MB_CTRL.R = 0x7F5A5A5A;

 STCU2.RUNSW.R = 0x00000001;
}
Offline script (Key-On)

AREA.RESET
AREA.CREATE log
AREA.SELECT log
AREA.VIEW log

DIALOG.YESNO "Flash programming prepared. Program new DCF now?"
ENTRY &progflash

  &UTEST_STCU_DCF_base=0x00400308
&current_address=&UTEST_STCU_DCF_base

WHILE ((Data.Long(ea:&current_address))!=0xFFFFFFFF)
(
         &current_address=&current_address+0x8
```

```
)
IF &progflash
(
    ;Lock0 ->TSLock enable UTEST memory
    PER.S ANC:0xFFFE0010 %LONG 0x3FFFFFFF

 ;1 - SET_DCF_STCU_STCU_SCK 0xD3FEA98B   key1 correct
    GOSUB program_word &current_address 0xD3FEA98B00080008

    ;2 - SET_DCF_STCU_STCU_SCK 0x2C015674   key2 correct
    &current_address=&current_address+0x8
    GOSUB program_word &current_address 0x2C01567400080008

    ;3 - SET_DCF_STCU_PLL_CFG 0x0C01001E    PLL set to 40Mhz
    &current_address=&current_address+0x8
    GOSUB program_word &current_address 0x0C01001E00080010

    ;4 - SET_DCF_STCU_STCU_CFG 0x015A0008   ;pointer LBIST1
    &current_address=&current_address+0x8
    GOSUB program_word &current_address 0x015A00080008000C

    ;5 - SET_DCF_STCU_STCU_WDG 0xFFFFFFF0
    &current_address=&current_address+0x8
    GOSUB program_word &current_address 0xFFFFFFF000080014

    ;6 - SET_DCF_STCU_STCU_CRCE 0xA5A5A5A5
    &current_address=&current_address+0x8
    GOSUB program_word &current_address 0xA5A5A5A50008001C

    ;7 - SET_DCF_STCU_STCU_ERR_FM 0xA5A5A5A5
    &current_address=&current_address+0x8
    GOSUB program_word &current_address 0xA5A5A5A500080028

 ;LBIST1
```

```
;8 - SET_DCF_STCU_STCU_LB_CTRL_1 0x02015410
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x0201541000080140

;9 - SET_DCF_STCU_STCU_LB_PCS_1 0x00000380
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x0000038000080144

;10 - SET_DCF_STCU_STCU_LB_MISREL_1 0xC3615561
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xC361556100080150

;11 - SET_DCF_STCU_STCU_LB_MISREH_1 0x087939CE
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x087939CE00080154

;LBIST2

;12 - SET_DCF_STCU_STCU_LB_CTRL_2 0x03015410
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x0301541000080180

;13 - SET_DCF_STCU_STCU_LB_PCS_2 0x00000380
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x0000038000080184

;14 - SET_DCF_STCU_STCU_LB_MISREL_2 0xD330EC46
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xD330EC4600080190

;15 - SET_DCF_STCU_STCU_LB_MISREH_2 0xE4E7BA34
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xE4E7BA3400080194

;LBIST3

;16 - SET_DCF_STCU_STCU_LB_CTRL_3 - 0x10015410
&current_address=&current_address+0x8
```

```
GOSUB program_word &current_address 0x10015410000801C0

;17 - SET_DCF_STCU_STCU_LB_PCS_3 0x00000380
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x00000380000801C4

;18- SET_DCF_STCU_STCU_LB_MISREL_3 0xC1605CAE
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xC1605CAE000801D0

;19 - SET_DCF_STCU_STCU_LB_MISREH_3 0x9B169C75
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x9B169C75000801D4

;20 - SET_DCF_STCU_STCU_MB_CTRL_0 0x915A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x915A5A5A00080600

;21 - SET_DCF_STCU_STCU_MB_CTRL_1 0x925A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x925A5A5A00080604

;22 - SET_DCF_STCU_STCU_MB_CTRL_2  0x935A5A5A
current_address=&current_address+0x8
GOSUB program_word &current_address 0x935A5A5A00080608

;23 - SET_DCF_STCU_STCUMB_CTRL_3 0x945A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x945A5A5A0008060C

;24 - SET_DCF_STCU_STCUMB_CTRL_4 0x955A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x955A5A5A00080610

 ;25 - SET_DCF_STCU_STCUMB_CTRL_5 0x965A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x965A5A5A00080614

;26 - SET_DCF_STCU_STCUMB_CTRL_6 0x975A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x975A5A5A00080618

;27 - SET_DCF_STCU_STCUMB_CTRL_7 0x985A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x985A5A5A0008061C
```

```
;28 - SET_DCF_STCU_STCUMB_CTRL_8 0x995A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x995A5A5A00080620

;29 - SET_DCF_STCU_STCUMB_CTRL_9 0x9A5A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x9A5A5A5A00080624

;30 - SET_DCF_STCU_STCUMB_CTRL_10 0x9B5A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x9B5A5A5A00080628

;31 - SET_DCF_STCU_STCUMB_CTRL_11 0x9C5A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x9C5A5A5A0008062C

;32 - SET_DCF_STCU_STCUMB_CTRL_12 0x9D5A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x9D5A5A5A00080630

;33 - SET_DCF_STCU_STCUMB_CTRL_13 0x9E5A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x9E5A5A5A00080634

;34 - SET_DCF_STCU_STCUMB_CTRL_14 0x9F5A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0x9F5A5A5A00080638

;35 - SET_DCF_STCU_STCUMB_CTRL_15 0xA05A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xA05A5A5A0008063C

;36 - SET_DCF_STCU_STCUMB_CTRL_16 0xA15A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xA15A5A5A00080640

;37 - SET_DCF_STCU_STCUMB_CTRL_17 0xA25A5A5A00080640
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xA25A5A5A00080644

;38 - SET_DCF_STCU_STCUMB_CTRL_18 0xA35A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xA35A5A5A00080648
```

```
;39 - SET_DCF_STCU_STCUMB_CTRL_19 0xA45A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xA45A5A5A0008064C

;40 - SET_DCF_STCU_STCUMB_CTRL_20 0xA55A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xA55A5A5A00080650

;41 - SET_DCF_STCU_STCUMB_CTRL_21 0xA65A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xA65A5A5A00080654

;42 - SET_DCF_STCU_STCUMB_CTRL_22 0xA75A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xA75A5A5A00080658

;43 - SET_DCF_STCU_STCUMB_CTRL_23 0xA85A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xA85A5A5A0008065C

;44 - SET_DCF_STCU_STCUMB_CTRL_24 0xA95A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xA95A5A5A00080660

;45 - SET_DCF_STCU_STCUMB_CTRL_25 0xAA5A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xAA5A5A5A00080664

;46 - SET_DCF_STCU_STCUMB_CTRL_26 0xAB5A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xAB5A5A5A00080668

;47 - SET_DCF_STCU_STCUMB_CTRL_27 0xAC5A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xAC5A5A5A0008066C

;48 - SET_DCF_STCU_STCUMB_CTRL_28 0xAD5A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xAD5A5A5A00080670

;49 - SET_DCF_STCU_STCUMB_CTRL_29 0xAE5A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xAE5A5A5A00080674

;50 - SET_DCF_STCU_STCUMB_CTRL_30 0xAF5A5A5A
```

```
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xAF5A5A5A00080678

;51 - SET_DCF_STCU_STCUMB_CTRL_31 0xB05A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xB05A5A5A0008067C

;52 - SET_DCF_STCU_STCUMB_CTRL_32 0xB15A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xB15A5A5A00080680

;53 - SET_DCF_STCU_STCUMB_CTRL_33 0xB25A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xB25A5A5A00080684

;54 - SET_DCF_STCU_STCUMB_CTRL_34 0xB35A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xB35A5A5A00080688

;55 - SET_DCF_STCU_STCUMB_CTRL_35 0xB45A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xB45A5A5A0008068C

;56 - SET_DCF_STCU_STCUMB_CTRL_36 0xB55A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xB55A5A5A00080690

;57 - SET_DCF_STCU_STCUMB_CTRL_37 0xB65A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xB65A5A5A00080694

;58 - SET_DCF_STCU_STCUMB_CTRL_38 0xB75A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xB75A5A5A00080698

;59 - SET_DCF_STCU_STCUMB_CTRL_39 0xB85A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xB85A5A5A0008069C

;60 - SET_DCF_STCU_STCUMB_CTRL_40 0xB95A5A5A
&current_address=&current_address+0x8
GOSUB program_word &current_address 0xB95A5A5A000806A0

;61 - SET_DCF_STCU_STCUMB_CTRL_41 0x7F5A5A5A
&current_address=&current_address+0x8
```

```
      GOSUB program_word &current_address 0x7F5A5A5A000806A4

      ;RUN
      ;62 - SET_DCF_STCU_STCU_RUN  ->LBPLLEN + MBPLLEN + RUN
      &current_address=&current_address+0x8
      GOSUB program_word &current_address 0x0000030100080000

  )
  ELSE
  (
    FLASH.List
  )
  SYStem.BdmClock 4MHz
ENDDO

program_word:
      entry &address &data

      ;MCR->PGM =1 enable program memory
      PER.S ANC:0xFFFE0000 %LONG 0x610

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

      PER.S ANC:0xFFFE0000 %LONG 0x611  ;MCR->EHV=1 program memory

       WHILE ((Data.Long(ea:0xFFFE0000)&0x0200)==0);     ;while
(FLASH.MCR.B.DONE == 0);

       PER.S ANC:0xFFFE0000 %LONG 0x610   ;MCR->EHV=0 program memory

       PER.S ANC:0xFFFE0000 %LONG 0x600  ;MCR->PGM =0

       RETURN
```

Appendix B Reference documents

SPC574K72xx self-test procedures (AN4551, DocID026636).

Revision history

Table 2. Document revision history

Date	Revision	Changes
17-Jul-2015	1	Initial release.