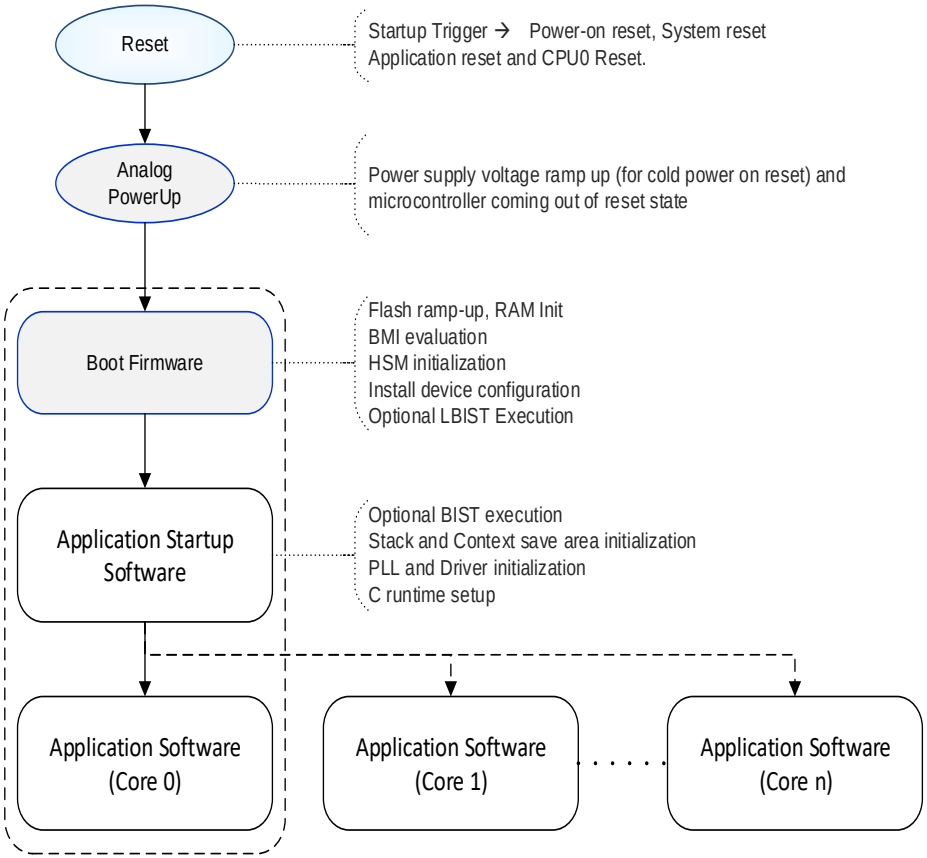
Built-In Self Test (BIST) on the TC3XX

TC3XX BIST context: Start-up Sequence



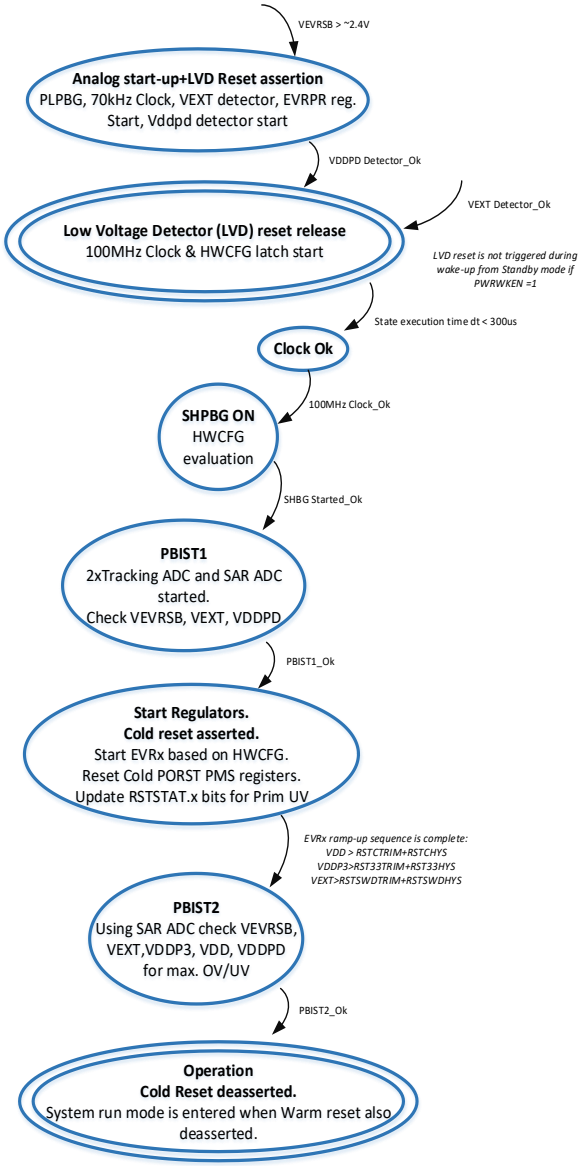
- On powering-up the microcontroller it goes through a brief power-up sequence and comes out of reset state.
- Boot Firmware (SSW) is the first instruction sequence executed immediately after reset. This prepares the hardware for the generic application needs of the targeted domain where the controller is being used. The execution control is then passed to the application start-up software.
- Application start-up software further initializes the hardware and software for the specific application.
- Execution and control is handed over to the application software.
- Execution of BISTs ensures a safe state for application software to run.

TC3XX BIST Overview

- PBIST – SM[HW]:PMS:PBIST
  - Checks supply levels, power functions and voltage monitors before Cold Power On Reset release.
- MONBIST - SM[HW]:PMS:MONBIST
  - Checks Secondary Voltage Monitors and Alarm paths
- LBIST - SM[HW]:MCU:LBIST
  - Checks most of the digital logic in the chip.
- MBIST - ESM[SW]:VMT:MBIST
  - Checks SRAMs for any errors.

PBIST (Power Built In Self Test) at start-up on TC3XX

Analog Startup and PBIST



- Executed automatically (by Hardware) on every cold-power on reset.
- Nothing to be done by application software
- Primary Low Power Bandgap (PLPBG) is checked against Secondary Bandgap (SHPBG) (PBIST1)
- VEVRSB & VEXT are checked using secondary monitor ADC – before starting the regulators (PBIST1)
- After regulator start-up, the supplies are checked for limits (PBIST2):
  - VEVRSB = 5,84V / 2,75V ± 5%
  - VEXT = 5,84V / 2,75V ± 5%
  - VDDP3 = 3,81V / 2,0V ± 5%
  - VDD = 1,46V / 1,0V ± 5%
- Device reset is not de-asserted until PBIST passes.

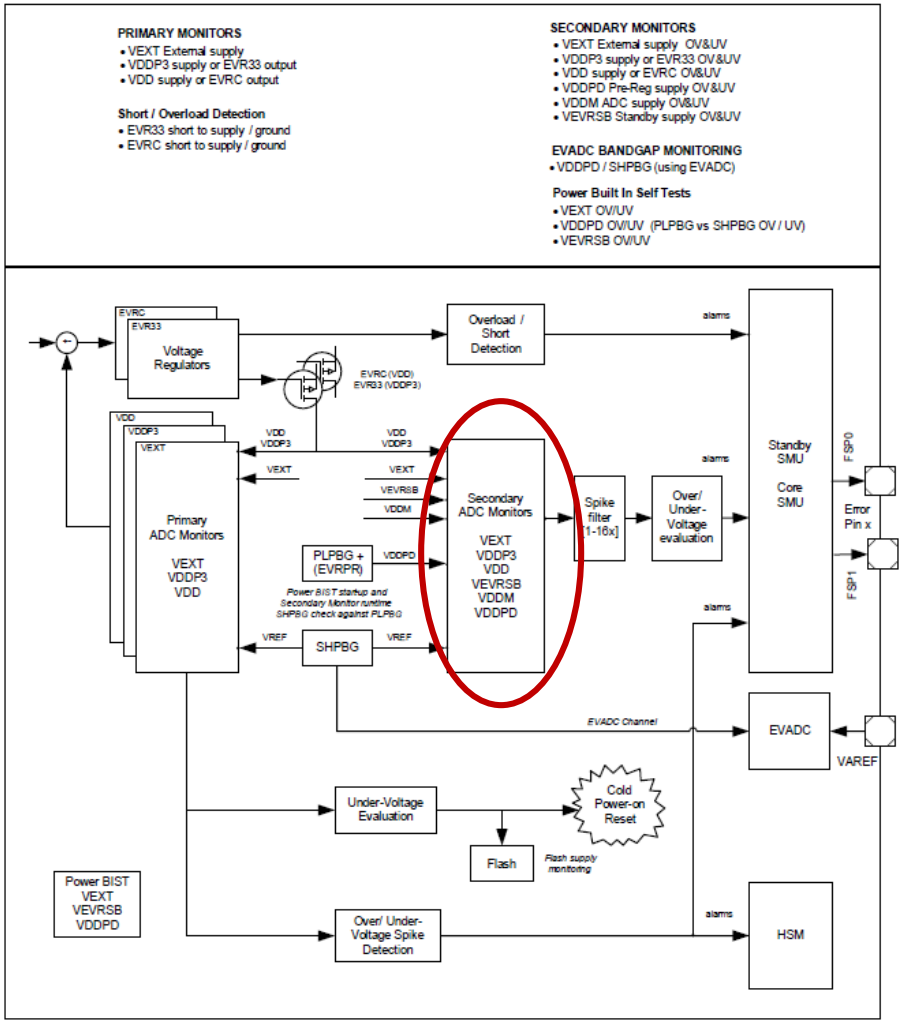
MONBIST

(Secondary Monitor and Standby SMU BIST)

On TC3XX

MONBIST

- Secondary monitor:
  - Safety monitor for multiple supply rails.
  - Under- and Over- voltage alarms sent to Standby-SMU.
- MONBIST:
  - Checks the secondary monitors, alarm paths and error pin fault logic (standby SMU).



Triggering the MONBIST

- To be triggered during Application Startup.
- Set the TSTEN bit of the MONBISTCTRL register.
- Wait as long as MONBISTSTAT.TSTRUN is set (indicates MONBIST is currently running – ~25us run-time).
- Result of the MONBIST can be read from MONBISTSTAT.TSTOK.

LBIST

(Logic Built-in Self Test) on TC3XX

Agenda

1. WHAT is LBIST
2. WHICH modules are affected by LBIST
3. HOW to configure the LBIST
4. WHEN to run the LBIST
5. SUMMARY

Agenda

1. WHAT is LBIST
2. WHICH modules are affected by LBIST
3. HOW to configure the LBIST
4. WHEN to run the LBIST
5. SUMMARY

LBIST Overview: Two line summary

- The Logic BIST (LBIST) executes structural tests (Test using Scan Chains) and checks the digital logic of the MCU to detect hard errors.

- In the TC3XX, the LBIST is used as a Safety Mechanism and contributes to achieving the Latent Fault Metrics (LFM) target.

Agenda

1. WHAT is LBIST
2. WHICH modules are affected by LBIST
3. HOW to configure the LBIST
4. WHEN to run the LBIST
5. SUMMARY

The modules affected by LBIST

- All digital modules in the TC3XX, except for the PMS sub-block are covered by the LBIST scan chains.
  - i.e. All CPUs, peripherals, ports (excl. Pads), HSM are tested by LBIST.
- PMS modules (EVR, SCR WUT, SMU_STDBY e.t.c.) are not covered by LBIST. But the PMS register interface is covered.
- All the safety mechanisms (e.g. SRAM & PFLASH ECC, MPUs, Bus SMs, and all SMU alarms & the SMU_CORE) are covered via LBIST.
  - **Therefore, there is no need to test the safety mechanisms again (e.g. by software) after proper LBIST execution.**
- Analog and mixed signal modules are not covered by LBIST.
  - Analog areas of ADCs, CONVCTRL, PLLs, HSCT/GIGETH-PHY e.t.c.
- Note: Simply repeating the LBIST (e.g. with the same or different seed) does not improve the coverage.

LBIST & SRAM/FLASH

- LBIST **does not** test the SRAMs & PFLASH & DFLASH areas.
- The RAMs have to be tested separately by the MBIST.
- However:
  - The SRAM redundancy registers are part of the scan chain and hence corrupted.
  - Therefore SRAMs contents are **not reliable** after LBIST and shall be initialized after LBIST, prior to usage.

- Similarly LBIST does not test the flash itself.
  - But the surrounding / digital logic (e.g. FSI or ECC blocks) are tested.

LBIST v/s SafeTlib

- Since we recommend to run LBIST, we don't have to test the safety mechanisms again.
- Therefore we do not need the SafeTlib anymore (in the TC3XX B-step).
- However: The LBIST is not a 1:1 replacement of SafeTlib.

LBIST	SafeTlib
H/W based test – tests the digital logic directly.	S/W based test: Tests the SMs & alarms via software based error injection.
Helps to achieve LFM targets directly based on the HW test coverage.	Does not directly contribute quantitatively to the LFM targets. The argumentation is complex.
Runtime (ASIL-D configuration): 6ms @ application startup.	Runtime: 10-25ms (depending on configuration) @ application startup.
Simple to enable: Enabled via bit in the BMI.	Complexity to enable: Need to integrate the software into customer application.
Software needed only for result analysis – much simpler.	Complex software: High development, support and maintenance effort (e.g. updated releases, different compilers).
During LBIST run, the MCU is not available.	SafeTlib runs on the CPU, so the system could be available for any communication/reaction.

Agenda

1. WHAT is LBIST
2. WHICH modules are affected by LBIST
3. HOW to configure the LBIST
4. WHEN to run the LBIST
5. SUMMARY

LBIST Configuration -1

- The LBIST configuration and status via 3 registers in the SCU:
  - LBISTCTRLx (x = 0-3)

Logic BIST Control 0 Register

LBISTCTRL0
Logic BIST Control 0 Register **(0164_H)** **Reset Value: Table 424**

31	30	29	28	27–20	19–16
LBISTREQRED	LBISTERRINJ	0	LBISTDONE	0	PATTERNS
w	rwh	r	rh	r	rw

15–2	1	0
PATTERNS	LBISTRES	LBISTREQ
rw	w	w

Logic BIST Control 1 Register

LBISTCTRL1
Logic BIST Control 1 Register **(0168_H)** **Reset Value: Table 425**

31–30	29–28	27	26–25	24–23	22–21	20–19	18–17	16
LBISTFREQU		BODY	SPLITSH		0		SEED	
rw		rw	rw		r		rw	

15–0
SEED
rw

Logic BIST Control 2 Register

LBISTCTRL2
Logic BIST Control 2 Register **($016C_H$)** **Reset Value: Table 426**

31–16
0
r

15–12	11–0
0	LENGTH
r	rwh

Logic BIST Control 3 Register

LBISTCTRL3
Logic BIST Control 3 Register **(0170_H)** **Reset Value: Table 427**

31–16
SIGNATURE
rh

15–0
SIGNATURE
rh

LBIST Configuration - 2

- Via these registers, The LBIST in TC3XX is configurable via the following parameters.
  - SEED: The seed of the random pattern generator.
  - PATTERNS: Number of random patterns driven in the test.
  - LBISTFREQU: Frequency of LBIST.
    - Note: LBIST runs on 100MHz backup clock with a divider.
  - SPLITSH: Number of partitions into which the scan chains are divided, and executed sequentially.
    - Helps to trade –off power jumps against total test time.
  - BODY: Pad behavior during LBIST (Weak pull-up or tristate).
  - LBISTREQ(RED): Starts the execution of the LBIST.
  - LBISTDONE: Indicates completion of LBIST.
  - LENGTH: Scan Chain length (Automatically loaded by FW).
  - SIGNATURE: Reflects the MISR signature of the LBIST execution.

Recommended configurations

- 2 Configurations are specified currently in the device appendix UM. (eg. TC39x BC-step LBIST configuration on the right)
- Please refer: SCU Appendix Chapter.
  - Config A runs within 6ms.
  - Config B runs within 50ms.
  - These runtimes are specified in the datasheet.
- Config A is sufficient for upto ASIL-D applications.
- Config A is the recommended configuration, and also used by the Firmware.
- Config B is provided currently as an alternate configuration. But Config A is the recommended configuration.

LBIST Configuration A

LBISTCTRL0 = 0x200;

LBISTCTRL2 = 0x86;

With LBISTCTRL1.BODY = 0:

- LBISTCTRL1 = 0x54000007
- LBISTCTRL3 = 0x01432DEB

With LBISTCTRL1.BODY = 1:

- LBISTCTRL1 = 0x5C000007
- LBISTCTRL3 = 0xF8097B38

LBIST Configuration B

LBISTCTRL0 = 0x1400;

LBISTCTRL2 = 0x86;

With LBISTCTRL1.BODY = 0:

- LBISTCTRL1 = 0x54000007
- LBISTCTRL3 = 0x80835418

With LBISTCTRL1.BODY = 1:

- LBISTCTRL1 = 0x5C000007
- LBISTCTRL3 = 0xDD5B50C5

Agenda

1. WHAT is LBIST
2. WHICH modules are affected by LBIST
3. HOW to configure the LBIST
4. WHEN to run the LBIST
5. SUMMARY

LBIST execution.

- To achieve LFM targets – the LBIST has to be run at least once per driving cycle.
  - Typically this is after each Cold Power-on Reset.

- LBIST can be triggered by
  - Application software

  or
  - Automatically by the firmware after each cold PORST (TC38xAB, 39xBB onwards).

- Triggering LBIST by application software is not recommended.
  - Reason: Additional startup time, software complexity.

- **Automatic execution via Firmware is the recommended approach to run LBIST on the TC3XX.**

Enabling LBIST execution via Firmware

- LBISTENA is part of the BMI.
- The user must set the LBISTENA bit to 1 in the BMI/BMHD.
- The LBIST Configuration A is used by firmware.
- Firmware will execute the LBIST after each Cold PORST (valid for qualified products).
  - SRAM initialization enabled via PROCONRAM after Cold or Warm PORST will be executed correctly.
- After LBIST execution an internal cold reset is triggered.
- **User software can start executing from this internal reset like after a normal cold PORST.**

Table 32 Boot Mode Header (BMHD) structure

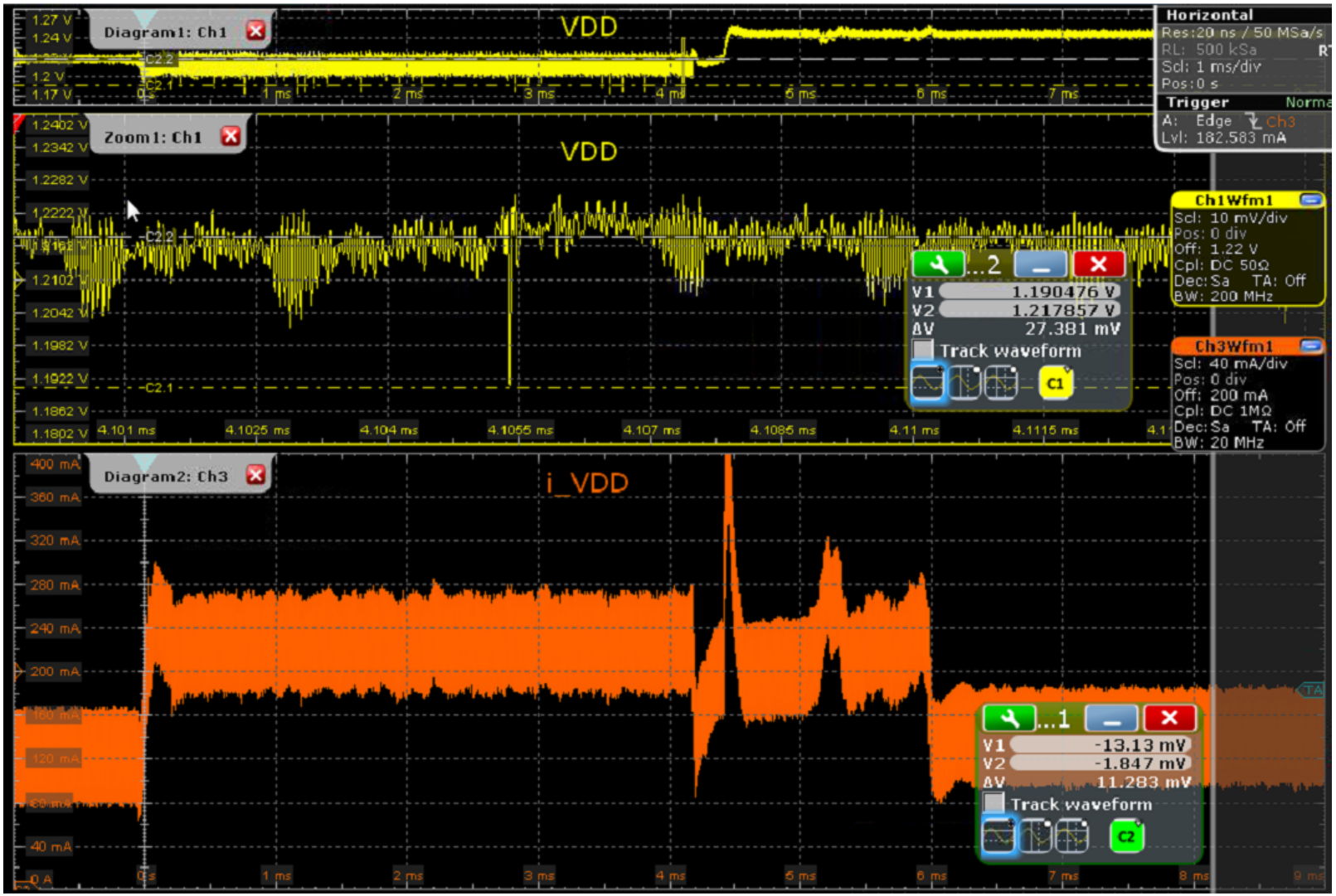
Field Name	Subfield	Description
BMI	Boot Mode Index - 16 bit	
	PINDIS bit [0]	Mode selection by configuration pins: 0B Mode selection by HWCFG pins is enabled 1BMode selection by HWCFG pins is disabled
	HWCFG bits [3:1]	Start-up mode selection: 111B Internal start from Flash 110BAlternate Boot Mode (ABM) 100B Generic Bootstrap Loader Mode (ASC/CAN BSL) 011BASC Bootstrap Loader Mode (ASC BSL) elseinvalid
	LSENA0 bit [4]	Lockstep monitoring control by SSW for CPU0: 0B Lockstep monitoring for CPU0 is disabled 1BLockstep monitoring for CPU0 is enabled
	LSENA1 bit [5]	Lockstep monitoring control by SSW for CPU1: 1) 0B Lockstep monitoring for CPU1 is disabled 1BLockstep monitoring for CPU1 is enabled
	LSENA2 bit [6]	Lockstep monitoring control by SSW for CPU2: 1) 0B Lockstep monitoring for CPU2 is disabled 1BLockstep monitoring for CPU2 is enabled
	LSENA3 bit [7]	Lockstep monitoring control by SSW for CPU3: 1) 0B Lockstep monitoring for CPU3 is disabled 1BLockstep monitoring for CPU3 is enabled
	LBISTENA bit [8]	LBIST execution start by SSW: 0B LBIST execution start by SSW is disabled 1BLBIST execution start by SSW is enabled
	CHSWENA bits [11:9]	Checker Software (CHSW) execution after SSW: 2) 101_B CHSW execution after SSW is disabled else CHSW execution after SSW is enabled
	reserved bits [15:12]	Reserved for future extensions, must be configured to 0 in UCB_BMHDx

During LBIST Run triggered by Firmware

- LBIST with config A executes for < 6ms.
- The complete MCU is unavailable during this time.
- The GPIO behavior is either Tristate (BODY = 1) or weak pull-up (BODY = 0). This includes also the FSP.
  - But: ESR0 & 1 have weak pull down during LBIST.
- No external communication is possible during the LBIST run, until the reset after the LBIST is complete. . E.g. cannot send or receive CAN messages during this time.
- When LBIST is started, roughly 200mA current jump occurs on i_VDD.
- With the default EVR settings (i.e. after FW trimming), around 50-90mV drop is seen for a very short time (40ns).
- There is no risk of undervoltage reset or DCDC instability.

LBIST current Jump during FW execution

12	Rovacs board VIH	Socket	DCDC	No sense	High Z	100nF	1.225-35mV	TC39xB#9	



LBIST result handling after FW execution

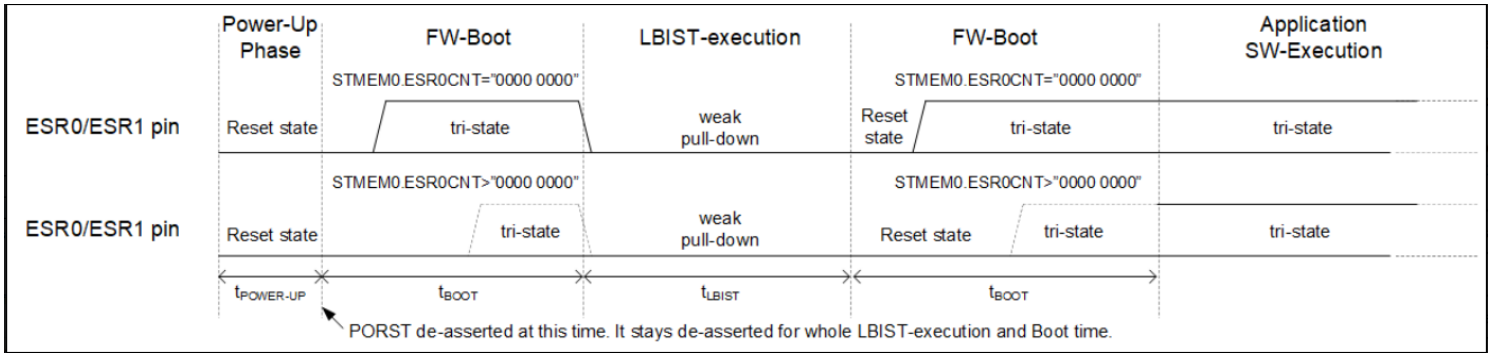
- When user software starts after the internal reset after LBIST, it should check the LBIST status.
- Check proper execution completion via LBISTDONE, LBPORST & LBTERM bits.
  - If LBIST is not properly terminated (e.g. due to warm PORST triggered by external regulator):
    - Next FW run will automatically retrigger the LBIST..
- Check that the signature matches the expected value in the appendix (for config A).
  - If the signature does not match: It shall be assumed to be not safe to start the application.
  - Strategy to retry with e.g. „n" cold PORSTs depending on the customer application.

LBIST Monitoring
(ESM[HW]:DFX:LBIST_MONITOR)

- In order to avoid that the chip hangs up during LBIST execution and external monitoring of the ERS0/ESR1 pins is necessary. For that purpose ESM[HW]:DFX:LBIST_MONITOR has been defined. The ESR0/1 behavior will show the following behavior:

1. The ESR0/1 pins are expected to show a weak pull-down behavior during the power-up phase of the device.
2. During FW execution ESR0/1 pins will either show a tri-state or a weak pull-down behavior depending on the STMEM0.ESR0CNT setting in UCB.
3. As soon LBIST operation has been started, a weak pull-down behavior will be visible again.
4. After LBIST operation is finished an internal PORST will be triggered followed by another FW-boot sequence (ESR0/1 pins will be tri-state or pull-down again depending on the STMEM0.ESR0CNT setting).

- In case ESR0/ESR1 pins are still showing a weak pull-down behavior after $t_{POWER-UP} + t_{LBIST} + 2 \times t_{BOOT}$ it has to be assumed that LBIST-operation did not finish as expected ⇒ the chip can be reset through an external PORST.



- Other startup monitoring mechanisms like external safety monitoring are implemented to ensure that startup time required by application is maintained.

- **Recommendation:** An external WDT can be used for this purpose as highlighted in ESM[HW]:SYS:WATCHDOG_FUNCTION. After PORST de-assertion, the external power supply waits for a first SPI message from the [MCU] which will serve the external WDT. If the wait time exceed the allowed time window because of a problem during start-up sequence, the appropriate reaction shall be taken at system level.

Agenda

1. WHAT is LBIST
2. WHICH modules are affected by LBIST
3. HOW to configure the LBIST
4. WHEN to run the LBIST
5. SUMMARY

SUMMARY

Summary.

- LBIST is a HW safety mechanism that helps to achieve LFM targets.
- LBIST covers all digital modules and not the analog modules.
- LBIST is not a 1:1 replacement for safeTlib. Rather, due to LBIST execution we do not need to test the safety mechanisms separately.
- Recommended triggering of LBIST is by enabling FW execution in the BMI.
  - Firmware triggers the LBIST only after a cold PORST.
- The <6ms configuration run by Firmware is sufficient for upto ASIL-D applications.

MBIST

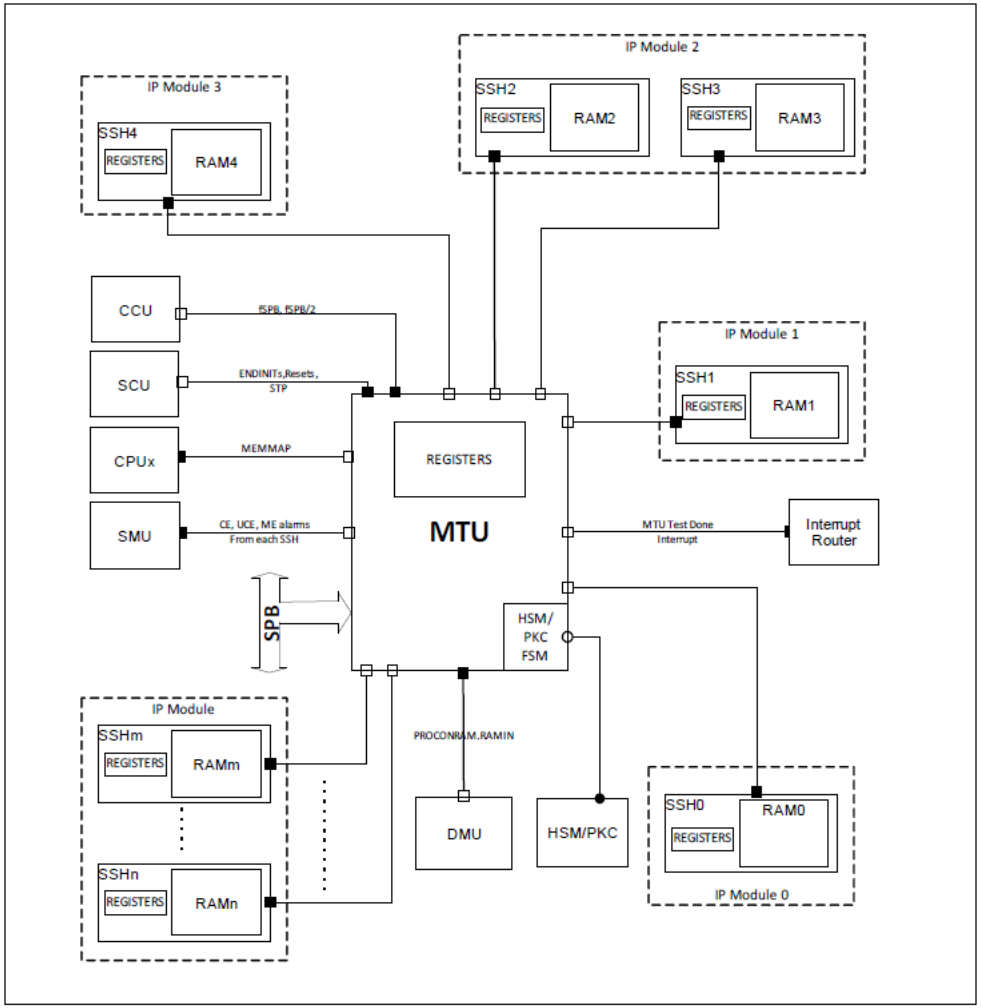
(Memory Built-in Self Test)

On TC3XX

MBIST in TC3XX

- TC3XX has a configurable MBIST engine (called SSH), supporting multiple test patterns on the SRAMs.
- For up-to ASIL-D applications, we recommend to run a single type of test: the 4N Non-Destructive-Test .
  - This test uses the ECC mechanisms.
  - This does not destroy the data in the memory.
  - When run on an initialized SRAM (i.e. ECC correct data)- the data remains ECC-correct after the test as well.

TC3XX Memory Controllers
MTU & SSH: System View



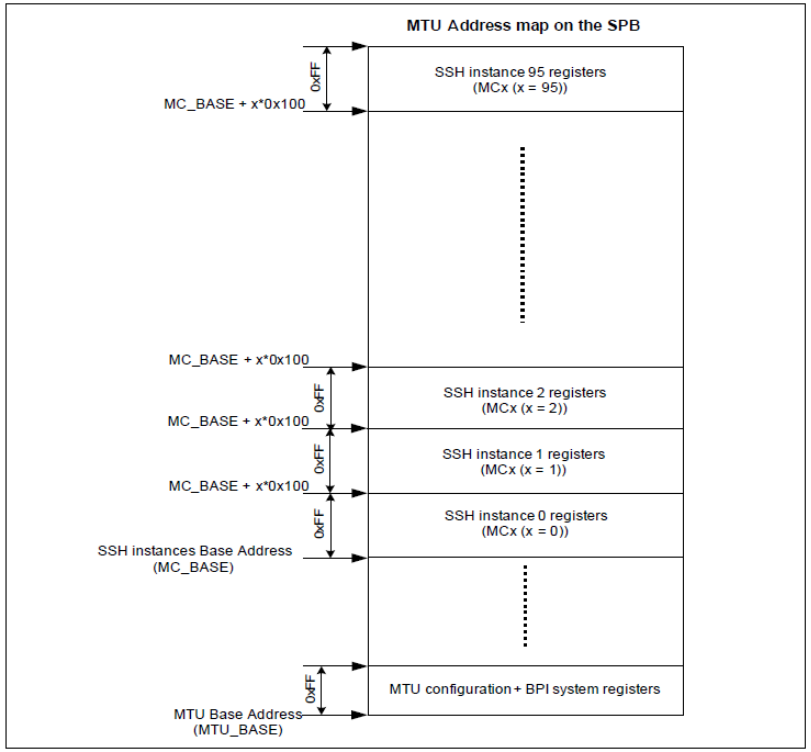
SSHs in the TC3XX & Mapping to Module SRAMs.

- SSH instance table in the Appendix chapter shows all the SSHs implemented in the device.
- In general, we do not recommend to try and map the implemented SSHs to logical SRAMs in the modules.
  - This is too complex and device specific.
  - e.g. DSPR + DCACHE are implemented within the same SRAM/SSH.
- Recommendation is to test / initialize and handle each and every SSH in the system.

Table 79 SSH instances

(MCx) x =	Module
0	CPU0_DMEM
1	CPU0_DTAG
2	CPU0_PMEM
3	CPU0_PTAG
4	CPU0_DLMU_STBY
5	CPU1_DMEM
6	CPU1_DTAG
7	CPU1_PMEM
8	CPU1_PTAG
9	CPU1_DLMU_STBY
10	CPU2_DMEM
11	CPU2_DTAG
12	CPU2_PMEM
13	CPU2_PTAG
14	CPU2_DLMU
15	CPU3_DMEM
16	CPU3_DTAG
17	CPU3_PMEM

MTU+SSH Programmers‘ View



Name	Base Address	Description
MTU_BASE	0xF0060000	Base Address of the MTU.
MC_BASE	0xF0061000	Base Address of the SSH Instances.

Short Name	Long Name	Offset Address
CLC	Clock Control Register	0000_H
ID	Identification Register	0008_H
MEMTESTi	Memory MBIST Enable Register i	0010_H+i*4
MEMMAP	Memory Mapping Enable Register	$001C_H$
MEMSTATi	Memory Status Register i	0038_H+i*4
MEMDONEi	Memory Test Done Status Register i	0050_H+i*4
MEMFDAi	Memory Test FDA Status Register i	0060_H+i*4
ACCEN1	Access Enable Register 1	$00F8_H$
ACCEN0	Access Enable Register 0	$00FC_H$

MTU Registers

MCi_CONFIG0	Configuration Registers	1000_H+i*100_H
MCi_CONFIG1	Configuration Register 1	1002_H+i*100_H
MCi_MCONTROL	MBIST Control Register	1004_H+i*100_H
MCi_MSTATUS	Status Register	1006_H+i*100_H
MCi_RANGE	Range Register, single address mode	1008_H+i*100_H
MCi_REVID	Revision ID Register	$100C_H$+i*100_H
MCi_ECCS	ECC Safety Register	$100E_H$+i*100_H
MCi_ECCD	Memory ECC Detection Register	1010_H+i*100_H
MCi_ETRRx	Error Tracking Register x	1012_H+i*100_H+x*2
MCi_RDBFLy	Read Data and Bit Flip Register y	1060_H+i*100_H+y*2
MCi_ALMSRCS	Alarm Sources Configuration Register	$10EE_H$+i*100_H
MCi_FAULTSTS	SSH Safety Faults Status Register	$10F0_H$+i*100_H
MCi_ERRINFOx	Error Information Register x	$10F2_H$+i*100_H+x*2

SSH Registers
(in each SSH)

Programming to run an MBIST test in TC3XX

- MBIST is required to find faults in the SRAMs.
  - ESM[SW]:VMT:MBIST.

- The SSH allows to configure and run different MBIST algorithms.
- Configurability includes:
  - Number & type of accesses.
  - Address modes.
  - Row/column toggle
  - Checkerboard
  - Direction up/down.

- The programming is done via the CONFIG0, CONFIG1 & MCONTROL register.

4N Non-Destructive Test (NDT).

- NDT works by reading the SRAM (DATA+ECC), inverting them and writing it back.
  - Inverted values of DATA+ECC are also a valid combination.
  - When done an even number of times, the data returns to the original value.

- NOTE: The SRAM has to be initialized to ECC correct values before running this test.

> A simple example of such a sequence is: {r, w*, r*, w}.
>
> Steps (Apply all the steps one after the other on each word, and then move to the next word, until the complete address range is covered):
>
> 1. r: Read data word including check bits.
> 2. w*: Write back all bits inverted.
> 3. r*: Read data word including check bits (All bits are inverted compared to original SRAM contents).
> 4. w: Write back all bits inverted (The Data is now same as the original SRAM contents).
>
> After this sequence, the user data is undisturbed and every bit would have seen '0' and '1'.

- This example is a 4N NDT test. N indicates the number of accesses to each location. Here it is 4.

Programming the NDT

- Sequence to program the NDT is specified in the UM.
  - The test time as well as current jumps corresponding to the 4N NDT test are described in the datasheet.

Test Programming Sequence:

1. Ensure that error detection is enabled, via the ALMSRCS and ECCS registers, and check for errors already present before the test (this is not required for the test, it is just a hint to software).
2. Enter memory test mode (set the corresponding bit in the MTU_MEMTEST register)
3. RANGE register assumed to have default reset value.
4. CONFIG0 = 0x4005
5. CONFIG1 = 0x5008
6. Set MCONTROL.DIR, RCADR and EN_DESCR and start the test by writing a 1 to register bit MCONTROL.START.
   MCONTROL := 4009_H (direction up, ROW first, EN_DESCR = 0, START = 1)
7. Clear MCONTROL.START
   MCONTROL := 4008_H (direction up, ROW first, EN_DESCR = 0, START = 0). MSTATUS.DONE will be reset now.
8. Wait for the end of the test - wait for MTU_DONE interrupt to be triggered, or poll MSTATUS.DONE bit to be set, via MTU_MEMDONE register.
9. Check the result of the test
   - verify *ERR bits in ECCD register
10. If the test failed check the error tracking register and ETRR overflow bit.
11. Clear the flags and the error tracking registers to enable further error tracking after the test.

To find failures during this test, software has to read the ECCD register for the CERR/UCERR and flags for any errors during the test. ETRR(0) will contain the first failed address, and the ERRINFO(0) register will contain the corresponding error type.

SSH Ganging

- Testing all SRAMs at the same time results in a huge current jump.
- At the same time, testing them sequentially results in a long test time.
- Therefore, as a trade-off, the SSHs are grouped into gangs.
- The SSHs in a gang are tested together.
- The Gangs are executed sequentially.
- In TC3XX, the gangs are specified in the Appendix (Device specific).
- Some example gangs from TC39xB:

Table 80 GANG-0

MCx(x=)	Module / SRAM
44	EMEM0
62	M_CAN10

Table 81 GANG-1

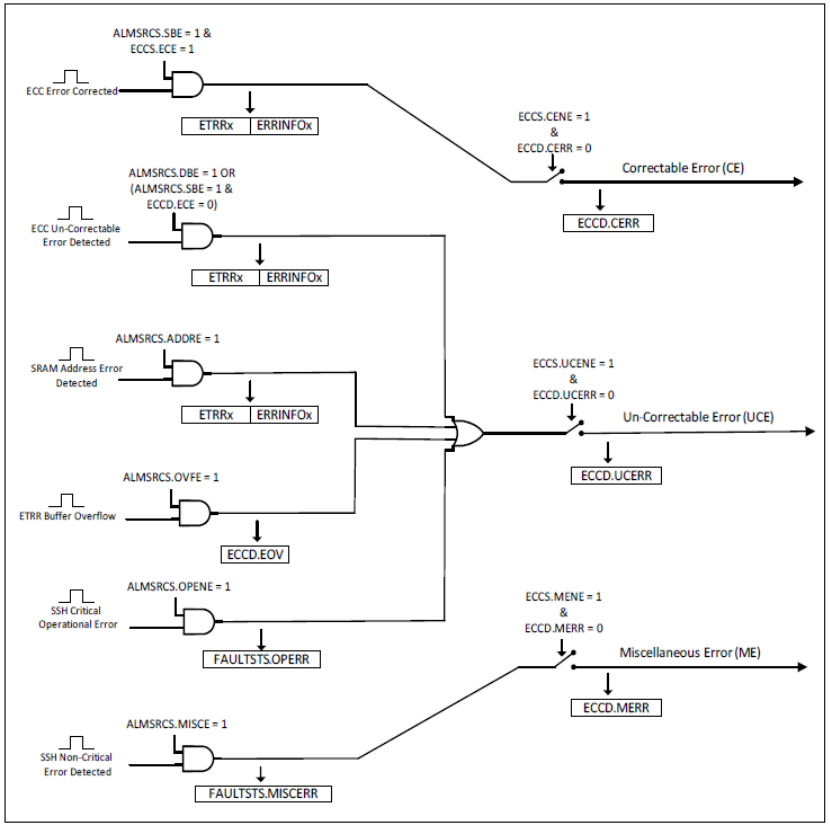
MCx(x=)	Module / SRAM
45	EMEM1
63	M_CAN20
64	M_CAN21
70	ERAY_MBF0

Table 82 GANG-2

MCx(x=)	Module / SRAM
46	EMEM2
71	ERAY_MBF1

Safety Mechanisms & Alarms

- From each SSH, 3 alarms are sent out to the SMU.
  - Multiple errors are muxed to these alarms.
  - SRAM errors: Single Bit Error (Correctable error), Double-bit un-correctable error and Address error, as well as ETRR overflow are implemented as in TC2XX.



New in TC3XX: OPERR sources

- New errors, OPERR sources are mapped to the Uncorrectable Error (UCE) alarm.
- Some of these errors are triggered on a startup, due to firmware handling of SRAMs.
  - Refer ESM:SYS:MCU_FW_CHECK

Table 572 Mapping of Errors to FAULTSTS.OPERR and UCE alarm

Error Status Bit	Error Description during normal application run	Hints for software reaction for recovery
FAULTSTS.OPERR[0]	SSH enabled. Functional access to SRAM is disabled.	Perform an application reset[1].
FAULTSTS.OPERR[1]	Auto-data-init or Partial erase (caches) was triggered, resulting in overwriting of SRAM data.	Perform an application reset[2].
FAULTSTS.OPERR[2]	Safety Flip-Flop error detected in one of the registers. (After a self test triggered by SFFD, this error has to be taken into account together with MISCERR[0]).	Perform a system reset. - Initialize the SRAM & SSH registers - Trigger a safety flip-flop self test via ECCS.SFFD. - Check that the test does not report a fail (MISCERR[0]). If the error still persists and cannot be cleared, a warm PORST has to be issued before performing the initialization and safety flip-flop self test.
FAULTSTS.OPERR[3]	MBIST FSM got triggered, test muxes got enabled in the data path, or some other random hardware failure triggered caused data in the SRAM to be overwritten or corrupted by a part of SSH logic.	Perform an application reset[1].

1) For FSI, perform a system reset. For SCR RAMs, a cold PORST is required.
2) For FSI, a system reset is required.

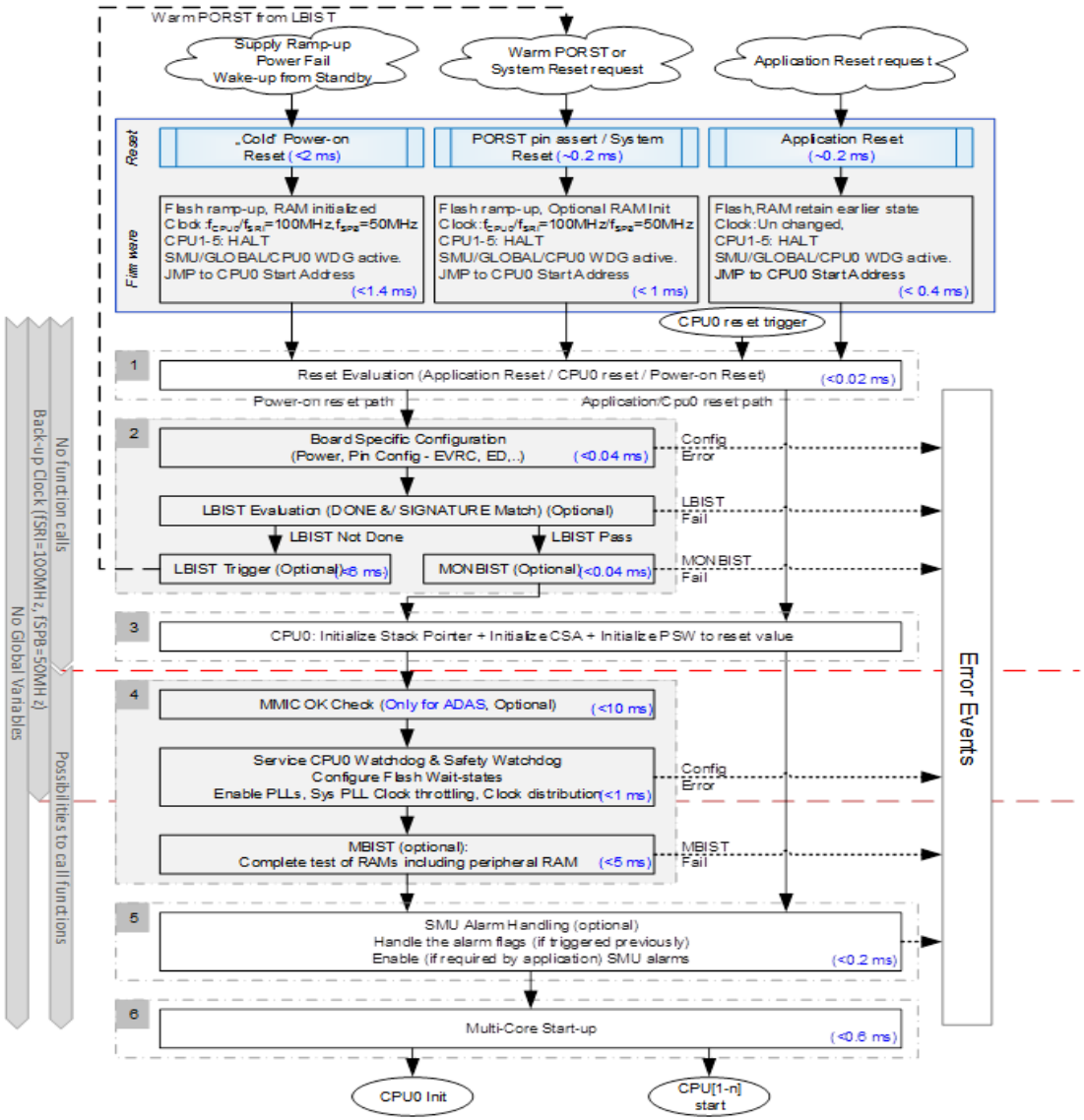
New in TC3XX: MISCERR sources

- New errors, MISCERR sources are mapped to the Miscellaneous Error (ME) alarm.
- Some of these errors are triggered on a startup, due to firmware handling of SRAMs.
  - Refer ESM:SYS:MCU_FW_CHECK

Table 573 Mapping of Errors to FAULTSTS.MISCERR and ME alarm

Error Status Bit	Error Description during normal application run	Hints for software reaction for recovery
FAULTSTS.MISCERR[0]	This bit can be set only after the software has triggered a Safety Flip-Flop self test via the ECCS.SFFD bit. If the test detected an error and the test failed, this bit is set. (The ME alarm is not triggered in this case).	Perform a system reset. - Initialize the SRAM & SSH registers - Trigger a safety flip-flop self test via ECCS.SFFD. - Check that the test does not report a fail (MISCERR[0]). If the error still persists and cannot be cleared, a warm PORST has to be issued before performing the initialization and safety flip-flop self test.
FAULTSTS.MISCERR[1]	Any of the alarm notifications of UCENE or CENE got disabled (i.e. one or more of the bits CENE or UCENE in the ECCS register went from 1 to 0). Note: ECCS.MENE is protected by Safety Flip Flop.	Re-initialize SSH registers (ECCS).
FAULTSTS.MISCERR[2]	Any of the safety mechanisms got disabled (i.e. ECE, TRE, SBE, DBE, ADDRE, OVFE, OPENE) was set from 1 to 0. Note: ALMSRCS.MISCE is protected by a Safety Flip Flop.	Re-initialize SSH registers (ECCD, ALMSRCS).

TC3XX- detailed startup



Part of your life. Part of tomorrow.

Infineon