Application Report
SPRAC26A–May 2016–Revised October 2018

TDA3xx Tester On Chip (TESOC)

Prasad Jondhale, Rajesh Veettil, and Sivaraj R

ABSTRACT

This application report explains how to use the Tester On Chip (hereafter mentioned as TESOC) module of the TDA3xx System-on-Chip (SoC). It also provides detailed design information for using TESOC to run field tests on targeted modules in the SoC. Flow charts and other programming details are presented for writing code for TESOC. This document is intended for designers and programmers who want to program the TESOC in TDA3xx SOC using PDK CSL APIs.

Contents

List of Figures

List of Tables

Trademarks

1 Introduction

Electrical and electronic systems in automotive road vehicles should meet the ISO 26262 international standard for road vehicles functional safety. Automotive Safety Integrity Level (ASIL) is a key component for ISO 26262 compliance. There are four levels in ASIL compliance (“A” to “D”). As a part of ASIL-B compliance the SOC must have software initiated structural (Logic and Memory testing) field-test capability on critical processors and memories to meet desired diagnostic coverage as required by ISO26262. The TDA3xx SOC, which is targeted for Advanced Driver Assistance System (ADAS) applications supports software initiated structural (Logic and Memory testing) field-test capability using TESOC module. The TESOC provides the capability of software initiated structural (Logic and Memory) testing on processing cores and safety critical memories, using factory programmed test vectors residing in TESOC ROM.

2 Tester On Chip (TESOC)

The TESOC is a safety module in TDA3xx capable of running software initiated structural tests in the field after deployment of SOC into the production module.

TESOC tests (also called field tests) can be initiated on any supported domain that can be powered-down by Power Reset Control Module (PRCM) without affecting TESOC. For example, Embedded Vision Engine (EVE) can be powered-down and it can be tested during run-time. The field test can be initiated by any SOC master having access to TESOC registers.

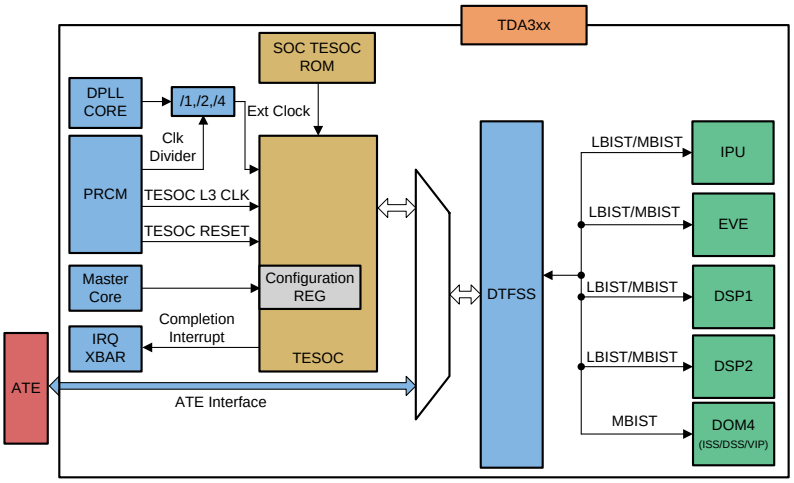

Figure 1. TDA3xx SoC With TESOC

Figure 1 shows the top level features for TESOC. For more details about TESOC hardware and integration into TDA3xzx, see the *TDA3x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0A, 2.0, 1.0A, 1.0 Technical Reference Manual*.

2.1 Logical Domains

To start the TESOC test gracefully, without affecting the functional operation of the rest of SoC, logical power domains are created so that TESOC can run test on one domain while rest of domains can be in functional mode. On each domain, tests are divided into multiple test vectors (compressed TDLs, also called slices) and there is flexibility to run tests for limited time in multiples of test slices. For instance, if DSP is idle for only say 30 µs, software can run 10 slices of DSP test, assuming each test slice takes 3 µs.

Logical domains are powered down by software before start of TESOC tests through PRCM to ensure that all pending transactions are terminated gracefully. Before entering field test mode, TESOC checks all dependencies in power domain and start test only if all modules in power domain are powered down. This is even though only single module is undergoing TESOC test. All outputs from these domains are bounded during test to ensure rest of the SOC is not affected.

TESOC waits for power-down of the module before starting the test. There is no way to indicate failure to start TESOC if there was no power-down. It is the responsibility of the software to ensure it.

TDA3xx supports structural logic tests (referred in this document as LBIST) on EVE, DSP, IPU and memory tests (referred to in this document as MBIST) on memories of IPU, EVE, DSP and DSS/ISS/VIP.

Table 1. TDA3xx TESOC Domains

Sr. No.	TESOC Domain	IP targeted	Voltage Domain	Power Domain	Modules in this domain
1	DOMAIN0 (1)	IPU	VD_CORE	PD_IPU	MCASP1, TIMER5, TIMER6, TIMER7, TIMER8
2	DOMAIN1	EVE	VD_ DSPEVE	PD_EVE	EVE
3	DOMAIN2	DSP1	VD_ DSPEVE	PD_DSP1	DSP1
4	DOMAIN3	DSP2	VD_ DSPEVE	PD_DSP2	DSP2
5	DOMAIN4 (2)	(ISS,DSS,VIP)	VD_CORE	PD_ISS PD_DSS PD_CAM	ISS, DSS, VIP

(1) In case of IPU self-test, all modules in PD_IPU should be brought into power-down state.

(2) While MBIST is being performed on domain 4, for example, any of VIP, ISS or DSS, all three must be powered down even if a single one has been selected for MBIST.

2.2 Modes of Operation

TESOC field-test can be initiated at three different stages of execution:

- Start-up Field Test – Start-up field test indicates field tests that are run during device boot/startup. Software can decide when to start TESOC test on device boot. TI recommends the start up field test due to less complexity in configuring TESOC in start up compared to runtime and shutdown.
- Shutdown Field Test – Software can trigger field-test when it sees that system is shutting down. It can be powered by battery during this time. The status can be stored and will be referred-to when the system is powered-up again. Appropriate action can be taken depending on the pass/fail status.
- Runtime Field Test – During the application runtime, software can trigger field-test when it sees any of supported module is free for minimum duration to at-least finish one slice. As both of LBIST and MBIST operations are destructive in nature, memory contents and logic states of flops will be lost after these operations. Runtime BIST operations are supported by hardware, but the SW complexity of system context save/restore will be enormously high, hence not recommended in general. The runtime field tests are not supported in the TI VSDK software.

NOTE: The TI SBL and Vision SDK support only the startup field tests

2.3 Fast Boot After TESOC IPU Field Test

During device boot, being device boot master, only IPU is up so it needs to do self-field test before proceeding with boot and starting tests on other domains like EVE/DSPs. This test initiated by IPU on itself is described as IPU self-test.

In case of the TESOC IPU self-test, IPU is reset after the test completion (through PRCM by TESOC completion interrupt), which results into device reboot. TDA3xx has the control module register TESOC_LAST_RESET_INDICATOR, which indicates that the reset is due to the TESOC test. Software writes into this register before triggering the self test. This register is not set by hardware and software should write the correct value into this register. This control module register is read by the SBL to distinguish between PORz vs TESOC initiated reset. SBL can do partial boot if reset is due to TESOC. Figure 2 shows the TESOC self test with partial boot. Path 1-2-3-4 shows the flow before TESOC IPU field test where ROM Bootloader (RBL) boots fully, whereas, path 1-2-3-5 is after TESOC IPU field test is completed, where full boot is bypassed as reset is due to TESOC test.

Device RBL reads this reset indicator register after booting up from reset and understands that the reset was due to TESOC. Software can choose to do partial boot using this indicator and control module registers ROM_CPU0_BRANCH for CPU0 and ROM_CPU1_BRANCH for CPU1. These registers store jump addresses for respective CPUs which is used by ROM to resume execution after field test.

The fast boot after TESOC IPU self-test is useful to optimize reboot time, avoiding double initialization. As TESOC IPU field test does not affect system memories like OCMC, DDR, peripheral initialization like QSPI can be avoided after reboot and RBL can jump directly to application. This is particularly helpful in cases with boot time restrictions like CAN initialization, and so forth.

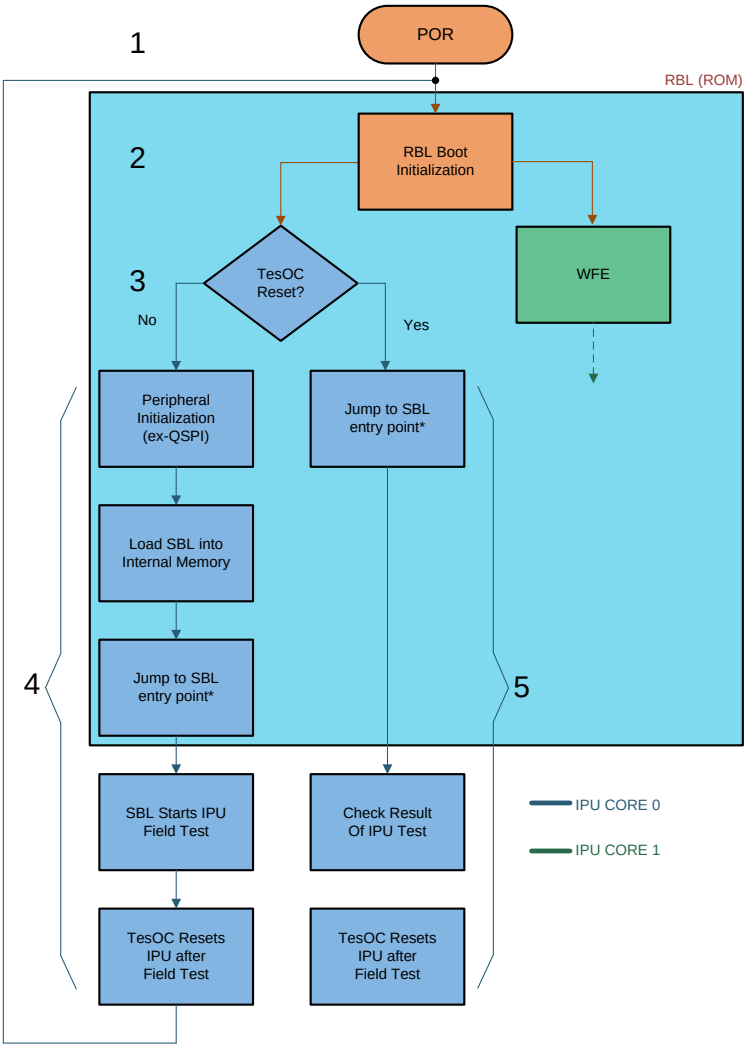

Figure 2. TDA3xx SoC IPU TESOC Boot Sequence

2.4 TESOC External Clock Configuration

For normal operation of TESOC, the external clock (TESOC_EXT_CLK) should be running at the rated speed. For more details, see the *TDA3x SoC for Advanced Driver Assistance Systems (ADAS) 15mm Package (ABF) Silicon Revision 2.0 Data Sheet*. The TESOC operation is not guaranteed if TESOC_EXT_CLK is not configured correctly. TESOC_EXT_CLK is configured through control module registers as shown in Figure 1. PDK PM APIs can be used for this configuration.

2.5 MBIST Clocking Requirements

For correct operation of TESOC MBIST on supported modules, modules under test should be running at the rated speed. If targeted module is not running at rated speed, MBIST operations are not assured. For rated frequencies on these modules, see the *TDA3x SoC for Advanced Driver Assistance Systems (ADAS) 15mm Package (ABF) Silicon Revision 2.0 Data Sheet*.

2.6 Default Slice Configurations

The TESOC ROM contains default and basic test vectors to achieve the desired diagnostic coverage on the targeted IPs and memories. These test vectors are compressed TDLs (also called slices) to be used by TESOC during the field test. TESOC ROM contains slices for each domain. Table 2 details these default slice configurations for LBIST and MBIST tests in the TDA3xx TESOC ROM.

Table 2. TESOC Domain Default Slice Configuration (1)

Sr. Number	TESOC Domain	IP Targeted	Test Type (1)	Slice Configuration
1	DOMAIN0	IPU	LBIST	0x003FFFF8, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
			MBIST	0x00000003, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
			DIAG	0x00400000, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
2	DOMAIN1	EVE	LBIST	0x000FF800, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
			MBIST	0x00000BFF, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
			DIAG	0x00100000, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
3	DOMAIN2	DSP1	MBIST	0xFFFFFFFF, 0x01FFFFFF, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
			LBIST	0x0, 0xFC000000, 0x3F, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
			DIAG	0x0, 0x0, 0x40, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
4	DOMAIN3	DSP2	MBIST	0xFFFFFFFF, 0x05FFFFFF, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
			LBIST	0x0, 0xFC000000, 0x3F, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
			DIAG	0x0, 0x0, 0x40, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
5	DOMAIN4(2)	ISS	MBIST	0xFFFFFFFF, 0x3FFF, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
		DSS	MBIST	0x0, 0x018000, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
		VIP	MBIST	0x0, 0x1C0000, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0

(1) Only LBIST diagnostic slices are provided.

(2) Domain4 does not have diagnostic slice.

2.7 TESOC Test Coverage and Test Duration

Table 3 shows TESOC test coverage in the TDA3xx SoC.

Table 3. TESOC Test Coverage and Test Duration

TDA3x Core	Stuck At Logic Coverage	Memory Coverage (1)	Time Taken (ms)
IPU	85.00%	100%	3.95
DSP1	90.10%	100%	13.9
DSP2	90.10%	100%	13.9
EVE	92.80%	100%	3.5
VIP	N/A	100%	0.8
DSS	N/A	100%	0.4
ISS	N/A	100%	3.1

(1) Memory coverage using TI proprietary algorithms.

Table 4. TESOC Slice Wise Performance

Sr. Number	TDA3x TESOC Domain	Test Type	Number of Slices	Time (µs)
1.	IPU	LBIST	19	3781.25
		MBIST	3	169.00
		Diagnostic	1	1125.00
2.	EVE	LBIST	9	1750.00
		MBIST	10	2125.00
		Diagnostic	1	375.00
3.	DSP1/2	LBIST	12	2812.50
		MBIST	57	11500.00
		Diagnostic	1	718.75

2.8 TESOC Interrupt Handling

The TESOC has one interrupt line, TESOC_IRQ_DONE, mapped to the device IRQ Crossbar IRQ_CROSSBAR_404. Before starting the field test, the software can enable this TESOC interrupt through the TESOC configuration registers. Also, the associated device control module registers need to be configured to route the interrupt from the crossbar to one of the device interrupt controller.

In case of IPU self-test, TESOC interrupt also acts as wakeup event to local IPU PRCM. In normal flow, the master starting TESOC test on the core will power down core before TESOC field test and power on after TESOC test is completed. Here interrupt processing is optional as polling on TESOC BUSY register can be used to check for test completion. But in case of IPU self-test as IPU itself triggers TESOC test, mechanism to power on IPU after field test completion is needed. Also fact that IPU is SOC master in TDA3xx and hence other core can not power it on. For this reason we need to map TESOC interrupt as Wakeup event in internal wakeup generator module of IPU. Once TESOC test is completed TESOC completion interrupt will wake up IPU and IPU will start rebooting. If TESOC completion interrupt is not enabled in case of TESOC IPU self-test there is no way to wake up IPU after test is completed.

Special software consideration for interrupt handling is required when TESOC field test on IPU (LBIST or MBIST) is triggered by core other than IPU (DSPs or EVE). In this case as DSPs/EVE are running faster than IPU, interrupt will be cleared by other core before IPU is woken up and will keep IPU in non-functional state. It is recommended that interrupt should not be cleared by DSPs/EVE till IPU reboot is completed. Register TESOC_LAST_RESET_INDICATOR (bit field [27:24]) can be used for this purpose as RBL after rebooting (due to TESOC test) changes it 0xAh to 0x5h.

2.9 Miscellaneous Features

- **Aborting Ongoing TESOC Field Test**

  If targeted module under test needs to be used by software when TESOC field test is ongoing, the master who has initiated the TESOC test can write to ABORT register to abort field test. This will abort field-test immediately and generate completion interrupt.

- **Support for Diagnostic Slice for TESOC Functionality Verification**

  To validate TESOC in functioning properly diagnostic slice is added for each LBIST domains. This slice will check if TESOC is able to detect errors during LBIST. A false compare is made during test and TESOC detects this, then the slice fails. Note that this slice is expected to fail and TESOC result registers should show it as fail.

  MBIST diagnostic slice are not supported.

- **Error Diagnostic Logs in Case of TESOC Test Failure**

  TESOC supports error diagnostic feature to give software access to the failure data. TESOC dumps error log to diagnostic log registers in case of field test failure. TESOC stores 16 failures logs which can be accessed through these diagnostic registers.

2.10 Usage Model

As described in Section 2.2, TESOC supports startup, runtime and shutdown field tests. Though these field tests can be triggered at any time during system runtime, below are recommended usage model for startup tests. Run time tests are run when Master sees core idle for minimum time required to execute field test.

- **Delayed Startup Field Tests**

  Considering boot time optimization needs for different use cases, TESOC start up field tests can be delayed till the point core is needed for use. For example system where CAN subsystem should be responsive within desired time limit running start up tests for EVE, DSP1/2 and ISS/DSS/VIP would add unnecessary delay as these cores are not needed during initial boot. In this case only IPU LBIST and MBIST can be run and field tests on EVE, DSP1/DSP2 and ISS/DSS/VIP can be done just before actual use. This approach is used in TI SBL of VISION SDK. Here upon boot only IPU LBIST and MBIST is done after system initialization and before loading core images for DSP1/DSP2 and EVE, TESOC field tests are run. The VISION SDK app triggers test for ISS/DSS/VIP before enabling them. Note that this is for reference purpose only and depending on customer use case, tests can be triggered in any order intended by user.

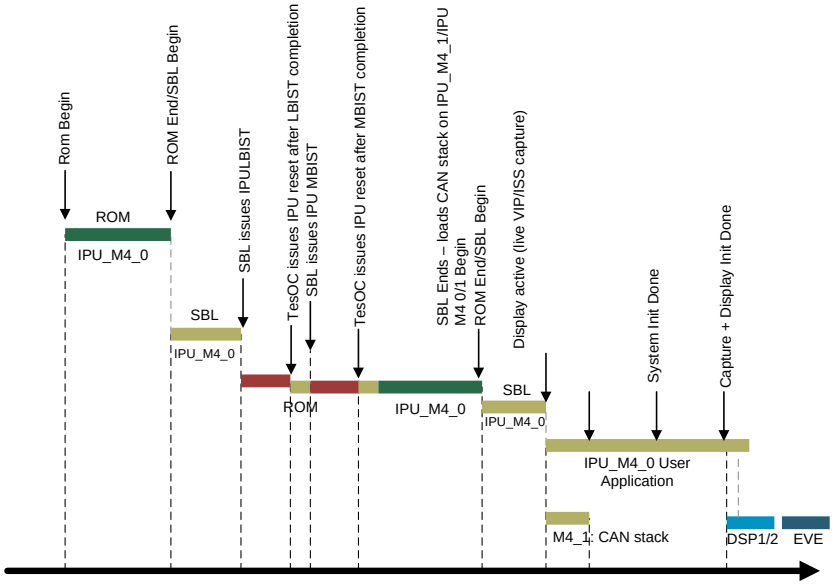

Figure 3. TESOC Startup Tests for Boot Optimization

3 TESOC – Usage

3.1 TESOC Field Test Execution Sequence

As discussed in Section 2.1, TESOC field tests can be started on one domain without affecting rest of SOC. Tests can be run for total available slices or subset of slices on particular domain.

This section describes generic programming sequence for starting TESOC field test. Also as an example IPU self-test sequence is explained in Section 3.3.

For the flow chart, see Figure 4.

1. Before starting the field test, the software should enable clocks to TESOC through PRCM.
2. Software has to poll TESOC_BUSY to make sure TESOC is not busy. Note that if TESOC is busy running field test, TESOC configuration is not allowed. In this case wait for TESOC to complete field test and once it is done proceed with next configuration.
3. Unlock the TESOC by writing b1010 (0xA) to the Lock register TESOC_LOCK [3:0] LOCK. Software has to clear Abort, Error Diagnostic, Status and Result Registers.
4. Configure register TESOC_DOMAIN_EN_DOM0 to TESOC_DOMAIN_EN_DOM4 to enable a domain.
5. Software must configure the slices for field test. Slices are configured from TESOC_SLICE_CONFIG_DOM0_n to TESOC_SLICE_CONFIG_DOM4_n (n = 0 to 7) registers.
6. Lock the TESOC by writing b0101 (0x5) to the TESOC Lock register TESOC_LOCK [3:0] LOCK.
7. Configure PRCM to put domain in IDLE mode.
8. If the specific target domain is in powered down state and the field test enable bit is enabled for the corresponding domain, TESOC will start the LBIST/MBIST operations and generate an interrupt when completed.
9. Software has to first read TESOC_SLICE_STATUS_DOM0_n - TESOC_SLICE_STATUS_DOM4_n (n = 0 to 7) to check if slice was completely run and then read the registers (TESOC_SLICE_RESULT_DOM0_n TESOC_SLICE_RESULT_DOM4_n) to check pass/fail status. This is to ensure that there are no false positive due to early exit due to test fail or abort. For more details, see 4 in Section 3.3.

3.2 Example : TESOC Field Test Execution Sequence for IPU Self-Test (Startup)

1. After device boot is complete, load the application to program the IPU TESOC test.
2. Write to the control module register TESOC_LAST_RESET_INDICATOR, which indicates to RBL that reset is due to the TESOC test, but not PORz or Warm reset.

```
HW_WR_FIELD32(TESOC_LAST_RESET_INDICATOR[27:24]) = 0xA
```

3. Set ROM_CPU0_BRANCH and ROM_CPU1_BRANCH register values so that RBL can jump to these addresses when it detects reboot after TESOC test. RBL once it detects reboot due to TESOC jump to these *jump_addrs*.

```
HW_WR_REG32 (ROM_CPU0_BRANCH, <cpu0_jump_addr>);
HW_WR_REG32 (ROM_CPU1_BRANCH, <cpu1_jump_addr>);
```

4. Clear TESOC configuration registers as these register are not hardware cleared. If TESOC has run test earlier configuration registers will reflect result of that test.

```
HW_WR_FIELD32 (TESOC_ABORT, 0x0);
HW_WR_FIELD32 (TESOC_MISC_CONFIG, 0x0);
HW_WR_FIELD32 (TESOC_INTR_STATUS_ENABLED_CLEAR, 0x0);
HW_WR_REG32 (TESOC_DIAG_INFO [0-16], 0x0);
HW_WR_REG32 (TESOC_SLICE_CONFIG_DOM [0-8], 0x0);
HW_WR_REG32 (TESOC_SLICE_STATUS_DOM [0-8], 0x0);
HW_WR_REG32 (TESOC_SLICE_RESULT_DOM [0-8], 0x0);
```

5. Configure slices to be run in the TESOC config registers for the IPU field test (LBIST/MBIST or all).

```
HW_WR_REG32 (TESOC_SLICE_CONFIG_DOM0[0-
8] = { 0x003FFFF8, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0 };
```

6. Configure IRQ XBAR for mapping the TESOC completion interrupt to one of the IPU interrupts. For the following example, irq line 67 is used.

```
HW_WR_FIELD32 (SOC_IRQ_DMARQ_CROSSBAR_REGISTERS_BASE +
             CTRL_CORE_INTR_DMA_IPU_IRQ_67_68,
             CTRL_CORE_INTR_DMA_IPU_IRQ_67_68,
             INTH_INT_ID_TESOC_IRQ_DONE);
```

7. Enable domain in domain enable register for domain 0.

```
HW_WR_REG32 (TESOC_DOMAIN_EN_DOM0) = 0x5;
```

8. Enable TESOC interrupt generation.

```
HW_WR_REG32 (TESOC_INTR_ENABLE_SET, 0x1);
```

9. Set source of slice execution.

```
HW_WR_FIELD32 (TESOC_MISC_CONFIG, TESOC_SOURCE_CONTROL, 0x0);
```

10. Configure the TESOC interrupt as wake up event in the IPU wakeup generator module. This will reset IPU once test is completed. As TESOC interrupt is mapped to 67th IRQ line of IPU, enable 51st bit.

```
HW_WR_REG32 (LOCAL_PRCM + IPU_WUGEN_MEVT0, 0x00000000);
 HW_WR_REG32 (LOCAL_PRCM + IPU_WUGEN_MEVT1, 0x00080000);
```

11. Configure PRCM to put IPU into power down mode. Disable add static dependencies and all modules that are in same power domain as of IPU.

NOTE: The sequence below is for reference purpose only and not complete.

```
HW_WR_REG32 (IPU_WAKEUP, 0x0);
/* Enable DEEPSLEEP of CORTEX-M3-1  */
HW_WR_FIELD32 ( IPU_CM3_NVIC_SYSTEM_CONTROL,
IPU_CM3_NVIC_SYSTEM_CONTROL_SLEEPDEEP, 1);
/* Put L2MMU in smart idle mode */
HW_WR_FIELD32 (IPU_MMU_SYSCONFIG, IPU_MMU_SYSCONFIG_IDLEMODE,
IPU_MMU_SYSCONFIG_IDLEMODE_SSIDLE);
/* Put L2MMU in smart idle mode */
HW_WR_FIELD32 ( IPU_MMU_SYSCONFIG,
IPU_MMU_SYSCONFIG_IDLEMODE, IPU_MMU_SYSCONFIG_IDLEMODE_SSIDLE);
/* Put local PRCM in smart standby wakeup mode */
HW_WR_FIELD32(IPU_WUGEN_STANDBY_CORE_SYSCONFIG,
IPU_WUGEN_STANDBY_CORE_SYSCONFIG_STANDBYMODE, 0x3

HW_WR_FIELD32 (IPU_PRM + IPU_PRM__PM_IPU_PWRSTCTRL,
               IPU_PRM__PM_IPU_PWRSTCTRL__POWERSTATE,
               IPU_PRM__PM_IPU_PWRSTCTRL__POWERSTATE__OFF);

HW_WR_FIELD32 (IPU_PRM + IPU_PRM__RM_IPU1_RSTCTRL,
               IPU_PRM__RM_IPU1_RSTCTRL__RST_IPU,
               IPU_PRM__RM_IPU1_RSTCTRL__RST_IPU__CLEAR);

HW_WR_FIELD32 (IPU_PRM + IPU_PRM__RM_IPU1_RSTCTRL,
               IPU_PRM__RM_IPU1_RSTCTRL__RST_CPU0,
               IPU_PRM__RM_IPU1_RSTCTRL__RST_CPU0__CLEAR);

HW_WR_FIELD32 (IPU_PRM + IPU_PRM__RM_IPU1_RSTCTRL,
               IPU_PRM__RM_IPU1_RSTCTRL__RST_CPU1,
               IPU_PRM__RM_IPU1_RSTCTRL__RST_CPU1__ASSERT);

HW_WR_FIELD32(IPU_CM_CORE_AON+ PU_CM_CORE_AON__CM_IPU1_CLKSTCTRL,
             IPU_CM_CORE_AON__CM_IPU1_CLKSTCTRL__CLKTRCTRL,
IPU_CM_CORE_AON__CM_IPU1_CLKSTCTRL__CLKTRCTRL__HW_AUTO);

HW_WR_FIELD32 (IPU_CM_CORE_AON +
               IPU_CM_CORE_AON__CM_IPU1_IPU1_CLKCTRL,
               IPU_CM_CORE_AON__CM_IPU1_IPU1_CLKCTRL__MODULEMODE,
IPU_CM_CORE_AON__CM_IPU1_IPU1_CLKCTRL__MODULEMODE__AUTO);

HW_WR_FIELD32 (IPU_CM_CORE_AON+ IPU_CM_CORE_AON__CM_IPU_CLKSTCTRL,
IPU_CM_CORE_AON__CM_IPU_CLKSTCTRL__CLKTRCTRL,

IPU_CM_CORE_AON__CM_IPU_CLKSTCTRL__CLKTRCTRL__HW_AUTO);
```

12. TESOC monitors power the state of IPU to start the field test. Once it is powered down, TESOC starts the field test on the configured domain.
13. After TESOC completes, the field test generates the completion interrupt that will wake up IPU. Then, IPU will start rebooting. Upon detection that reset is due to TESOC, RBL will jump to <jump_addr>.
14. Check for TESOC_LAST_RESET_INDICATOR [27:24] to become 0x5. If the TESOC test is already completed, check the results of the test and clear the interrupt.

```
HW_WR_REG32 (TESOC_INTR_STATUS_ENABLED_CLEAR) = 0x1;
```

Check the results of the configured slices only and slices that are completed; default value of the result registers is 1. For more details, see 4 in Section 3.3.

```
for(block=0;block<8;block++) //8 blocks each containing 32 slice
{
 sliceResult = HW_RD_REG32( TESOC_SLICE_RESULT_DOM0[block] );
 for(slice=0;slice<32;slice++)//32 slices in each block
 {
   /*check if current slice was configured for execution*/
   if(sliceConfig[block] & (1<<slice))
   {
     sliceResultPassed = sliceResult & (1<<slice);
     /* current slice failed to completed  execution or  current slice is failed */
     if(sliceStatus[currentSlice]==sliceNotCompleted)
     ||  sliceResultPassed)
     {
      sliceResult[currentSlice]=sliceFailed;
      checkReturn = FAIL;
      }
      else
      {
       tesocResultStruct->sliceResult[currentSlice]=slicePassed;
      }
   }
 }
}
```

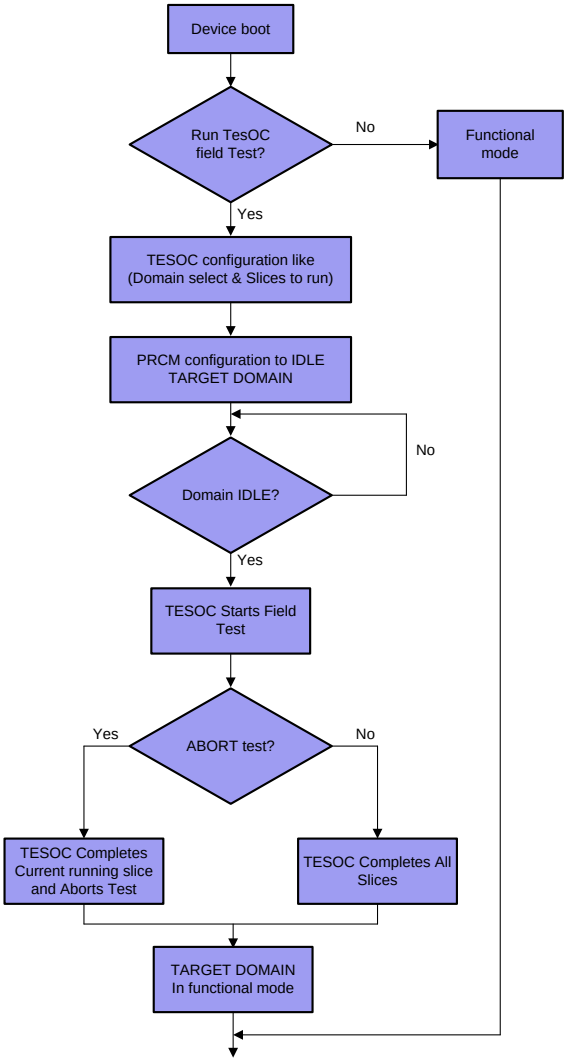

Figure 4. TESOC Field Test Execution Sequence (Generic)

3.3 Few Care-Abouts

1. Logic and Memory testing is destructive in nature and there is no hardware-assisted Context Save/Restore mechanisms. The TESOC solution is best suited for Startup and Shutdown tests.
2. Before starting the field test, the software should enable clocks to TESOC through PRCM.
3. At any given time, only one domain can be tested by TESOC. The Domain Enable (DE) register should be programmed for one domain at a time. Programming multiple domains in DE MMR corrupts the TESOC operation.
   a. TESOC should be configured to test only one domain at a time.
   b. Software has to configure TESOC again to test next domain.
4. Software should only read pass/fail of the configured slices. The default value of the result register is “1”, which indicates a Pass. The software should only read pass/fail of the configured slices from LSB to MSB. TESOC executes all configured slices from LSB to MSB. In case of Abort or Failures, TESOC exits the field test executing current slice. If software reads from MSB, it can interpret it as a pass on that slice that was never run. Software should read the pass/fail register from LSB to MSB and if it sees a Failing slice, it should re-run all slices after that.
5. After configuring TESOC and enabling a domain by writing to the domain enable[0-4] register followed by Lock register, it is assumed that only the following OCP transfers will be performed until an interrupt from TESOC is seen:
   a. Write to abort register.
   b. Read to configuration registers (NOT memories).

3.4 TESOC TDA3xx Software Special Considerations

While the TESOC-based BIST operations (described in the device-specific TRM) are fully operational in all TDA3x revisions, there are certain additional software considerations to be taken into account while performing the BIST operations. The software considerations are meant for avoiding certain incorrect sequences of events during the BIST operations that can affect the test results or cause the domain under test to be in an undefined state.

Special software considerations are needed for the following issues:

- On device boot DSPs are uninitialized; these uninitialized signals cause false TESOC MBIST test failures. To avoid this DSPs are powered down before start of TESOC test. For more details, see the *TDA3x SoC Errata for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0, 1.0A, 1.0*.

  The PDK bootloader (SBL) takes care of powering down the DSP modules before starting TESOC tests. This can be used as a reference.
- To ensure that TESOC correctly determines field pass/fail results and avoids an unintended sequence of events, TESOC MBISTs are recommended to be run at OPP_NOM frequencies. An MBIST result at other frequencies is not guaranteed.
- Before starting the MBIST field test, the software should make sure the module is running at OPP_NOM mode. If the device needs to run in another mode than NOM during the startup field test, it can do a field test in OPP_NOM mode and then switch to the desired mode. For example, for OPP usage, see Section 3.6.

3.5 Programming Guide

This section illustrates software sequence for programming TESOC to perform field tests using PDK CSL APIs. Available PDK CSL APIs are:

For more details about TESOC interface APIs, see the *PDK/PDK TDA Software Developer Guide* wiki.

Table 5. TESOC Starterware APIs

Sr. Number	PDK CSL API Name	Description
1	int32_t TESOCGetDefaultSliceInfo (uint32_t baseAddr, tesocTestCfg_t *testCfg)	This API gets default slice configuration information for a particular domain
2	int32_t TESOCClearPrevState(uint32_t baseAddr, uint32_t domainLabel)	This API clears register state of previous TESOC test run (if any)
3	int32_t TESOCConfigAndStart(uint32_t baseAddr, const tesocTestCfg_t *testCfg)	This API configures TESOC for test and starts test on selected domain
4	uint32_t TESOCGetTestExecutionStatus(uint32_t baseAddr)	Get execution state of TESOC (busy/completed)
5	int32_t TESOCWaitUntilBusy(uint32_t baseAddr, uint32_t timeout)	This API is used to wait till TESOC starts test
6	int32_t TESOCWaitUntilNotBusy(uint32_t baseAddr, uint32_t timeout)	This API is used to wait till TESOC completes running test
7	int32_t TESOCCheckTestResult(uint32_t baseAddr, const tesocTestCfg_t *testCfg);	This API Checks result of TESOC test
8	int32_t TESOCAbortTest(uint32_t baseAddr)	This API aborts ongoing TESOC test
9	void TESOCClearAbort(uint32_t baseAddr)	This API clears TESOC abort register
10	int32_t TESOCSetSliceSrc(uint32_t baseAddr, uint32_t testSliceSrc)	This API will set slice source for running TESOC test
11	int32_t TESOCRunTesoCDiagnostic(uint32_t baseAddr, uint32_t testId)	This API will run TESOC diagnostic slice for targeted domain
12	int32_t TESOCGetAdvanceResult(uint32_t baseAddr, const tesocTestCfg_t *testCfg, tesocAdvanceResult_t *advanceResult)	This API reads detailed result of TESOC test. Result contains details like slices configured for test, slice status like completed/not run and result (pass/fail) of each of these slice
13	int32_t TESOCReadDiagnosticLog(uint32_t baseAddr, uint32_t *const *diagnosticLog)	This API reads error diagnostic log for TESOC test failure
14	void TESOCIntrEnable(uint32_t baseAddr, uint32_t intrMask)	This function enables specified TESOC interrupts
15	void TESOCIntrDisable(uint32_t baseAddr, uint32_t intrMask)	This function disables specified TESOC interrupts
16	uint32_t TESOCGetIntrEnable(uint32_t baseAddr);	This function gets the status of enabled interrupts
17	uint32_t TESOCGetIntrStatus(uint32_t baseAddr);	This function returns the status of interrupts
18	void TESOCIntrClear(uint32_t baseAddr, uint32_t intrMask)	This Function clears the status of specified interrupts
19	uint32_t TESOCGetIntrRawStatus(uint32_t baseAddr)	This function returns the status of interrupts
20	void TESOCUnlockMMR(uint32_t baseAddr)	Unlock access to TESOC registers by writing unlock pattern to TESOC lock register
21	void TESOCLockMMR(uint32_t baseAddr)	Lock access to TESOC registers by writing lock pattern to TESOC lock register

3.5.1 Example

This section illustrates the way in which the TESOC can be used to run field tests on selected modules. PDK CSL APIs are used to configure TESOC. Figure 5 shows the flow chart.

The TESOC field test can be started at any domain without affecting the rest of SoC. TESOC will start the logic and memory test after the module is powered down by the software. Configuring TESOC for running the test is explained in the following steps:

NOTE: Power down DSPs with PM APIs before starting any TESOC test to avoid random TESOC errors as they are uninitialized.

1. Configure TESOC_EXT_CLK

   To ensure proper operation of TESOC TESOC_EXT_CLK needs to be configured through control module registers. It can be configured at approximately 12/24/48Mhz. It is recommended to configure TESOC_EXT_CLK to 48 MHz. PDK PM APIs can be used to configure this clock.

2. Clear the TESOC registers status from the previous run before starting the field test.

   Except for the Abort and Domain enable registers that are hardware cleared, software has to clear Abort, Error Diagnostic, Status and Result Registers.

   The abort register must be software cleared as a precaution to the corner case of the “Field test ended normally and Master wrote abort at the same time”, now the interrupt is due to the end of field test and the abort is not cleared. Hence, the next field test will be aborted”.

   Clearing of the previous state is done by following the PDK CSL API:

```
/* Clear register state of previous TESOC run */
TESOCClearPrevState (SOC_TESOC_BASE, DOMAIN_LABLE);
```

3. Configure the TESOC interrupt.

```
/* configure TESOC interrupts */
TESOCIntrEnable (SOC_TESOC_BASE, INT_MASK);
```

   Interrupt configuration is not mandatory for TESOC tests except for IPU. In case of IPU, TESOC completion interrupt acts as wakeup signal in local IPU PRCM (WUGEN), which resets IPU after TESOC test and starts rebooting. For this reason, TESOC interrupt needs to be enabled and configured as a wakeup signal for TESOC IPU self-test.

4. Interrupt Crossbar configuration.

   Interrupt *TESOC_IRQ_DONE* for TESOC needs to be mapped in the IRQ crossbar. The code below shows one possible configuration. TESOC_IRQ_DONE interrupt is mapped to interrupt no.44 of IPU.

   TESOC_IRQ_DONE is mapped interrupt to interrupt number 44 of IPU.

```
/*
 * Configure TESOC interrupt for IPU. TESOC interrupt acts
 * as wake up event for IPU to reboot.
 */
IRQXBARConnect(SOC_IRQ_DMARQ_CROSSBAR_REGISTERS_BASE,
               CPU_IPU1,
               (SBL_IPU_TESOC_TEST_INTR_LINE_NUMBER - 22U),
               TESOC_IRQ_DONE);
Intc_IntPrioritySet(SBL_IPU_TESOC_TEST_INTR_LINE_NUMBER,
                    (uint16_t) 1U,
                    (uint8_t) 0U);
Intc_SystemEnable(SBL_IPU_TESOC_TEST_INTR_LINE_NUMBER);

/* Register IPU Wake-up event instead of ISR */
IPU_WUGEN_Enable(SBL_IPU_TESOC_TEST_INTR_LINE_NUMBER);
```

5. Power down the module on which the TESOC test is to be run using PM APIs.

6. Configure TESOC for the field test.

   Configure TESOC for target domain and slices to be enabled. You can use the PDK CSL API TESOCConfigAndStart for this purpose.

```
int32_t TESOCConfigAndStart (SOC_TESOC_BASE, tesocTestCfg_t *testCfg);
```

   Structure tesocConfigStruct contains the domain on which the field test is to be run (slices to be run for selected domain).

7. Get the TESOC test running status.

   If the module is powered down properly and all TESOC configurations are correct, TESOC will start executing the field test on the selected module. TESOC execution status register can be used to read the test status. The following PDK CSL API can be used:

   `int32_t TESOCGetTestExecutionStatus (SOC_TESOC_BASE);`

8. Wait for the test completion (interrupt based or polling).

   TESOC issues TESOC_TEST_DONE interrupt after the test completes, which can be used to check test completion. TESOC also has the BUSY register that indicates whether TESOC is busy running the field test. The following PDK CSL API can be used while waiting until the test completes.

```
int32_t TESOCWaitUntilNotBusy (SOC_TESOC_BASE);
```

9. Check the test results.

   Upon completion of the test TESOC issues, TESOC_TEST_DONE interrupts and updates status and the result registers. These registers are used to detect test pass and fail status. The following PDK CSL API does this task:

```
int32_t TESOCCheckTestResult (SOC_TESOC_BASE);
```

10. Clear the interrupt.

```
/*Clear TESOC interrupt status */
TESOCIntrClear (SOC_TESOC_BASE, INT_MASK);
```

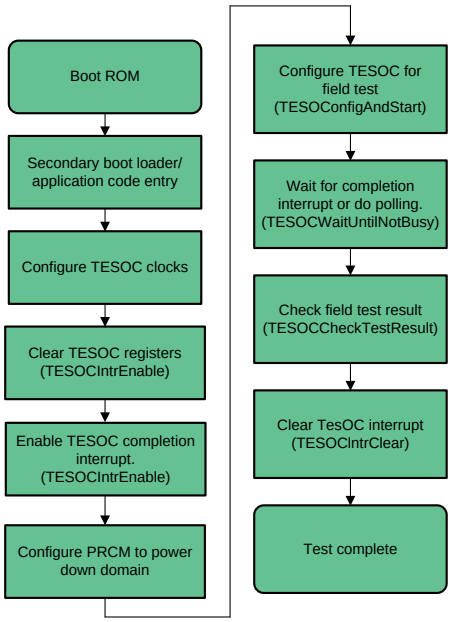

Figure 5. TESOC Field Test Flow Example With PDK CSL APIs

3.6 TDA3x Secondary Bootloader (SBL) TESOC Flow

TDA3xx secondary bootloader (SBL) enables TESOC field tests during device boot before application is loaded. These are start up field tests and SBL enables all slices for all supported domains.

When TESOC is enabled, SBL divides the boot sequence between pre-TESOC and post-TESOC; this is to avoid reinitializing SOC after the TESOC IPU self-test. Pre-TESOC initialization only configures a minimum part of the SoC required for IPU field test.

Below is brief SBL execution flow with TESOC field test.

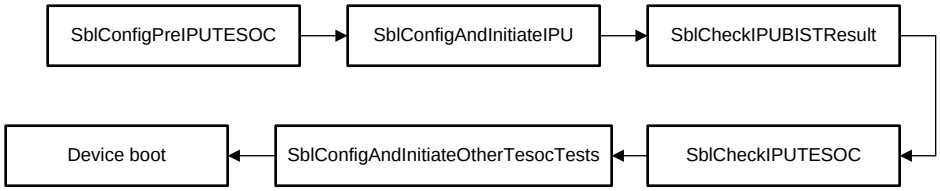

Figure 6. SBL Execution Flow With TESOC Field Test

SBL app uses SBL library to abstract TESOC field test configuration specified in section 2.4. Below are SBL library functions for TESOC. These functions can be used by applications as well by including SBL library.

Table 6. SBL Library Functions for TESOC

Sr. Number	Function Name	Brief
A	int32_t SBLLibRunTesocTest (uint32_t testId, const tesocTestCfg_t *tesocTestConfig)	Run TESOC on selected domain with config like domain, slice configuration in tesocTestConfig.
B	int32_t SBLLibCheckTesocTestResult (const tesocTestCfg_t *tesocTestConfig)	Checks the result of previously run TESOC test.

- TESOC field tests are enabled in default configuration and run unless SBL_CONFIG_DISABLE_SAFETY_FEATURES is set through build flags. In other configurations, TESOC can be enabled or disabled explicitly using the flags in sbl_lib_config_tda3xx.h (pdk\packages\ti\boot\sbl_auto\sbl_lib\src\tda3xx).

```
#define SBL_LIB_CONFIG_ENABLE_TESOC
```

- To make sure TESOC ROM is not altered due to EMI or cosmic effects, before running TESOC, a TESOC ROM CRC check and compare is done with the precomputed fixed CRC value. If the CRC check fails, the boot is aborted. The CRC check can be disabled by using the flag below, though it is not recommended.

```
#define SBL_LIB_CONFIG_ENABLE_IPU_TESOC_ROM_CRC
```

- Values for the TDA3xx PG2.0A ROM CRC are:

```
#define TESOC_ROM_IPU_CRC_SIGNATURE_REGL    ((uint32_t)0x6F10A976)
#define TESOC_ROM_IPU_CRC_SIGNATURE_REGH    ((uint32_t)0x731BA4C1)
```

TDA3xx SBL flow with TESOC field tests enabled:

1. Configure CORE and PER DPLLs.
2. Run TESOC IPU field test.
3. Configure DSPEVE OPP_x AVS voltage values.
4. Configure DSPEVE clocks at OPP_NOM frequencies.
5. Execute DSP1/2, EVE & DSS/ISS/VIP TESOC tests.
6. Reconfigure DSPEVE clocks to OPP_x frequency configuration.
7. Continue application boot

3.7 Failure to Complete TESOC Field Test

Failure to complete TESOC field test can be due to an issue in the domain-under-test or in the TESOC module. User can take below steps to root cause failure.

- Confirm TESOC functionality by negative testing. TESOC has diagnostic slices that injects an artificial error and verifies that TESOC indeed fails. Diagnostic slices are described in the section 1.9 (b).
- Confirm the domain-under-test functionality
  - Re-run the field test and if the error occurs multiple times then report a permanent failure and abort the application.
  - Log the error diagnostic information as described in the section 1.9 (c). This diagnostic data can be logged into Flash or any other NVM.
  - On multiple failures please contact your TI representative with error diagnostic logs for further analysis of failure symptoms.

4 References

- *TDA3x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 2.0A, 2.0, 1.0A, 1.0 Technical Reference Manual*
- *TDA3x SoC for Advanced Driver Assistance Systems (ADAS) 15mm Package (ABF) Silicon Revision 2.0 Data Sheet*
- *TDA3x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision Silicon Revision 2.0, 1.0A, 1.0*
- *PDK/PDK TDA Software Developer Guide* wiki

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Original (May 2016) to A Revision	Page
• Replaced StarterWare PM APIs with PDK PM APIs throughout the document.	1
• Update was made in Section 1.	2
• Updates were made in Section 2.1.	2
• Updates were made in Section 2.2.	3
• Update was made in Section 2.3.	3
• Update was made in Section 3.2.	8
• Update was made in Section 3.5.1.	15

IMPORTANT NOTICE AND DISCLAIMER

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

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

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

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