TDA3xx Error Signaling Module (ESM)

Vivek Dhande, Rajesh Veettil, and Sivaraj R

ABSTRACT

The Error Signaling Module (ESM) in TDA3xx is used to report certain activity on the monitored signals at an external error pin or to the device CPUs through interrupt. The external error pin is normally used as a second indication path to switch off (or reset) the device by an external device. Therefore, the external controller is able to reset the device or keep the system in a fail-safe state by disabling the peripherals outside of the ECU. This application report looks at different signals that can cause system failure and can be monitored using ESM. This document also aims at efficient use of ESM using the Starterware device driver APIs.

Contents

List of Figures

List of Tables

1   Hardware Introduction

The ESM is a safety MODULE in TDA3xx capable of monitoring activity on signals that are safety critical. Upon detecting any such activities, the ESM can alert the device through an interrupt or an external device (like the microcontroller or field programmable gate array (FPGA)) by means of an external error signal generation. The device, internal or external, can then take appropriate action to keep the system in fail-safe state.

The ESM is capable of monitoring following type of signals:

- Error interrupt signals
- Alert interrupt signals
- Memory or logic power loss events
- Reset signals
- PLL lock-loss events
- Idle request signals
- Standby signals

All trademarks are the property of their respective owners.

Submit Documentation Feedback

- Any interrupt in the system that can be mapped through the ESM IRQ crossbar
- Any DMA event in the system that can be mapped through the ESM DMA crossbar

The ESM hardware is built with support for monitoring 128 channels (also called as Signals or Events). After detection of certain activity on monitored channels, ESM can report an error through interrupt to the IRQ_CROSSBAR module – through which it can go to any CPU (IPU/DSP/EVE) or to an external pin, named *esm_error.*

- Feature list:
  - 128 channels in *esm_group1* [1] with configurable interrupt and error pin behavior
  - Generation of two group interrupts on detection of certain activity on monitored channels
  - *esm_error* pin to signal certain activity on monitored channels
  - Configurable low time period for error signal pin
  - Error forcing capability for debug and diagnostic purpose

[1] All the input signals monitored by ESM are referred to as *esm_group1* throughout this document.

1.1 ESM Functional Description

Figure 1 shows the ESM functional blocks. Along with error generation on external pin, ESM can also generate two group interrupts: namely ESM_IRQ_LOW and ESM_IRQ_HIGH interrupts to the device. Each channel monitored can be set to generate high or low level interrupts through configurable registers.
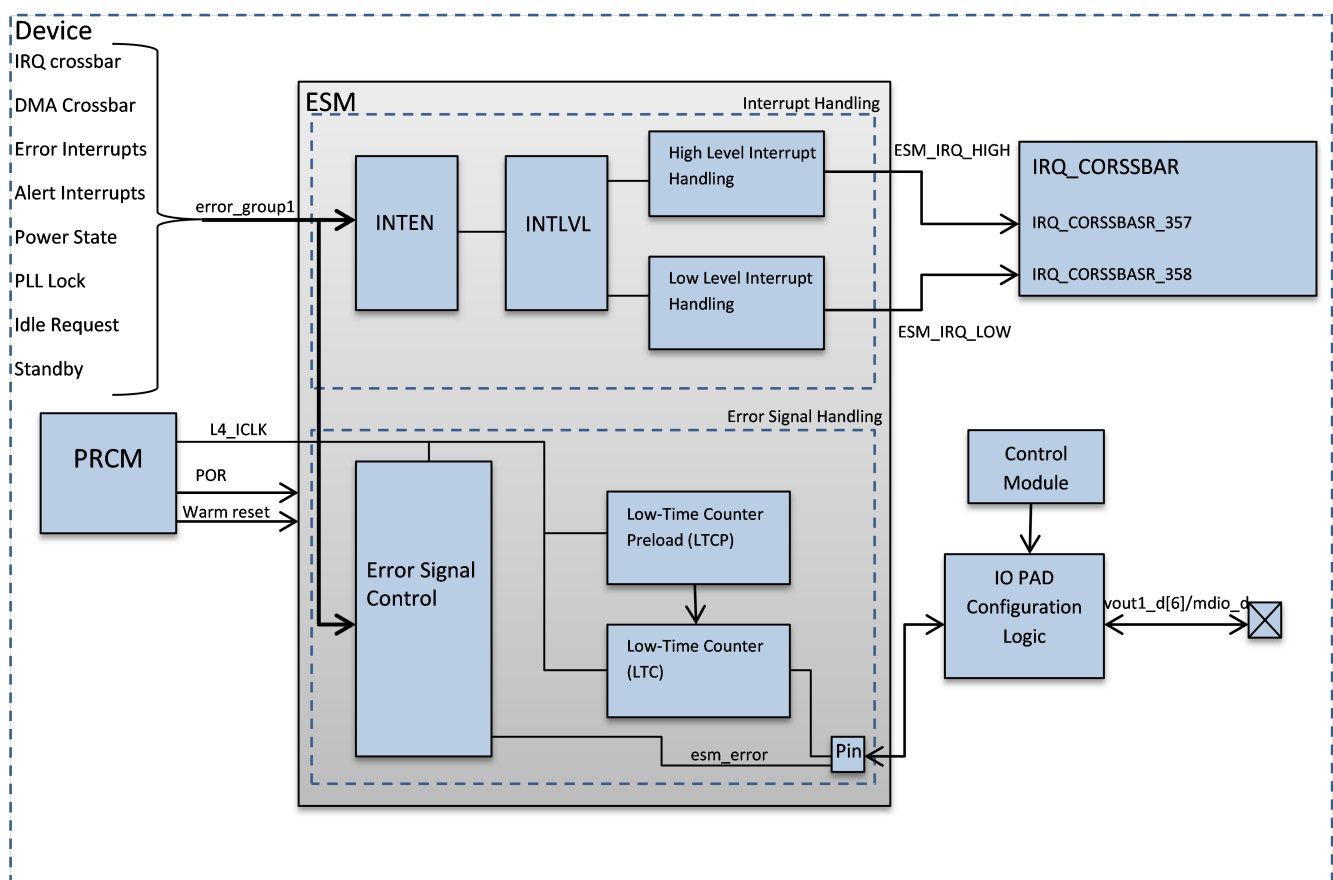

Figure 1. ESM Overview

1.1.1 Interrupt Handling

Input signals monitored under *esm_group1* by ESM, are maskable and configurable. They can be enabled to generate interrupt through the configurable ESM Interrupt Enable Set/Status Registers (ESMIESRx) or disabled through the configurable ESM Interrupt Enable Clear/Status Registers (ESMIECRx). On detection of certain activity on any input channel from *esm_group1*, the corresponding interrupt status flag is set in the ESM Status Registers (ESMSRx). This status is kept until power-on-reset.

On detection of certain activity on any input channel from *esm_group1*, ESM can generate one of the two interrupts to the device: *ESM_IRQ_HIGH* or *ESM_IRQ_LOW*. Particular input channel can be configured to generate low level or high level interrupt upon certain activity on it. Particular input channels can be configured to generate high level interrupt by configuring the corresponding bit in the ESM Interrupt Level Set/Status Registers (ESMILSRx). The same can be done to generate low level interrupt for a given input channel by configuring corresponding bit in the ESM Interrupt Level Clear/Status Registers (ESMILCRx).

1.1.2 Error Signal Handling

The *esm_error* pin is an active low signal. The ESM drives the *esm_error* pin high as long as no error is detected and drives it low on detection of activity on monitored input channels. The state of the *esm_error* pin can be read through the ESM Error Pin Status Register (ESMEPSR). ESM can report an error on the *esm_error* pin on detection of certain activity on any channel from *esm_group1*. Influence of any input channel from *esm_group1* on *esm_error* can be enabled by configuring the corresponding bit in the ESM Influence Error Pin Set/Status Registers (ESMIEPSRx). Influence of any input channel from *esm_group1* on *esm_error* can be disabled by configuring the corresponding bit in the ESM Influence Error Pin Clear/Status Registers (ESMIEPCRx).

The error generated on the external pin is used by the external device as an indication of the system state change. Therefore, it is necessary for this pin to be low for sufficient time so that the external device should not sense this transition as a glitch. The low time period for the external error pin is configurable using the ESM Low-Time Counter Preload Register (ESMLTCPR), which makes sure that this is the minimum low time period for the error pin. The ESM Low-Time Counter Register (ESMLTCR) shows the current value of the down counter for controlling low-time of the error pin.

The *esm_error* pin is low active when the error is generated and it remains low until the Low Time Counter reaches the final value, for instance, when it expires and a valid 4-bit key value is written to the ESM Error Key Register (ESMEKR) by the software (the *esm_error* pin reset request). This assures that the certain activity on the monitored channels is recognized by the external device.

1.2 ESM Operational Modes

ESM can operate in two modes: Normal Mode and Error Force Mode. ESM mode of operation can be configured through ESMEKR register.

1.2.1 Normal Mode

This mode of operation of ESM can be activated by writing 0x0 to the ESM Error Key Register (ESMEKR[3:0]). In this mode, on detection of certain activity on monitored signals, ESM can report an error on the external pin or can generate an interrupt to device depending on the configuration. If ESM is configured to generate an interrupt as an indication of certain activity on the monitored signals, the ESM Interrupt Offset High (ESMIOFFHR[8:0]) and ESM Interrupt Offset Low (ESMIOFFLR[8:0]) Registers give the channel number of the highest pending interrupt request for the high level and low level interrupt line, respectively. If ESM is configured to indicate certain activity on the monitored channels of an external pin, reset of the *esm_error* pin (driving it high/de-assertion of the *esm_error* pin) is done as soon as the Low Time Counter expires and software initiates reset through register write, depending on which one is later.
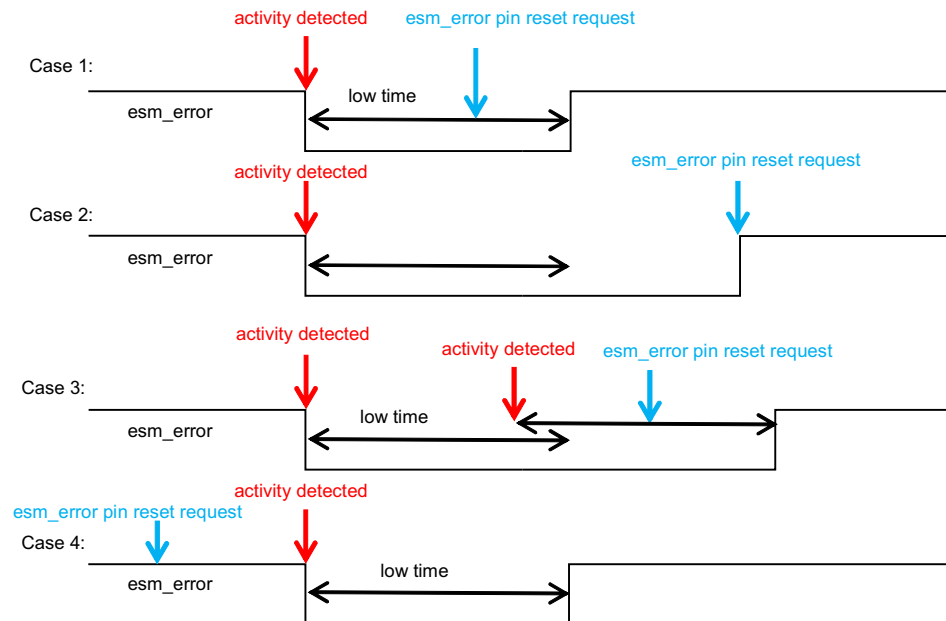
Figure 2. ESM esm_error Pin Timings

Case 1 and 2: As shown in Figure 2, for reset of the *esm_error* pin to happen, both the low time counter expiration and the *esm_error* pin reset request conditions are necessary. Reset of the *esm_error* pin takes place as soon as the condition happens (which is later).

Case 3: If another activity is detected when the *esm_error* pin is already low (within the low time period of the *esm_error* pin), the *esm_error* low time counter is reloaded with a low time preload value configured by the application and the counter is restarted as shown in Figure 2.

Case 4: Where reset of the *esm_error* pin is requested, even before certain activity on the monitored signals occurs, the *esm_error* pin goes low for the Low Time Period configured through the ESM Low-Time Counter Preload Register (ESMLTCPR) as soon as certain activity is detected on monitored channels. After the Low Time Counter expires, the *esm_error* pin is reset back to high. This scenario can be used to test the *esm_error* pin functionality. This case is not recommended and should be avoided by software.

1.2.2 Error Force Mode

In the above mentioned functionality (case 4), reset of the *esm_error* pin is requested even before detection of certain activities on the monitored signal, which should be testable by the software in a way that failure can be forced. This can be done by configuring the ESM in Error Force Mode, which can be done by writing a dedicated key 0xA to ESMEKR[3:0]. This sets the *esm_error* pin to go low for the specified time. This allows testing not only the *esm_error* pin functionality but also the signal reaching the error pin activation unit. While ESM is in Error Force Mode, it cannot handle normal error functionality; any activity detected gets latched and software should be aware of this as it has requested the test.

2 ESM – Integration in TDA3xx

Figure 1 shows integration of the ESM module in the device, including information about clocks, resets, monitored signals in the system and hardware requests.

2.1 Input Channel Mapping

Table 1 lists the mapping of the *esm_group1* channels.

Table 1. ESM Input Interrupt Channel Mapping

ESM Channel No	Channel Request Tag/Signal Name	Channel/Signal Type	Description/Activity Detected
ESM_GROUP1_0	ESM_GROUP1_0_IRQ_0 [1]	ESM IRQ Crossbar	
[2]ESM_GROUP1_1	ESM_GROUP1_1_IRQ_1[1]	ESM IRQ Crossbar	[2]
ESM_GROUP1_2	ESM_GROUP1_2_IRQ_2[1]	ESM IRQ Crossbar	[2]
ESM_GROUP1_3	ESM_GROUP1_3_DMA_3[1]	ESM DMA Crossbar	[2]
ESM_GROUP1_4	CTRL_MODULE_CORE_IRQ_SEC_EVTS	Error Interrupt	Firewall error [2]
ESM_GROUP1_5	L3_MAIN_IRQ_DBG_ERR	Error Interrupt	Debug error on L3[2]
ESM_GROUP1_6	L3_MAIN_IRQ_APP_ERR	Error Interrupt	Application error on L3 [2]
ESM_GROUP1_7	L3_MAIN_IRQ_STAT_ALARM	Alert Interrupt	L3 NoC Statistic Collector Alarm [2]
ESM_GROUP1_8	CTRL_MODULE_CORE_IRQ_THERMAL_ALERT	Alert Interrupt	Thermal Alert [2]
ESM_GROUP1_9	DCC1_IRQ_ERROR	Error Interrupt	DCC 1 error [2]
ESM_GROUP1_10	DSP1_IRQ_TPCC_ERR	Error Interrupt	DSP 1 EDMA TPCC error [2]
ESM_GROUP1_11	DSP2_IRQ_TPCC_ERR	Error Interrupt	DSP 2 EDMA TPCC error[2]
ESM_GROUP1_12	PRCM_PD_DSS_AGOOD	Power State	DSS memory power loss
ESM_GROUP1_13	PRCM_PD_EVE_AGOOD	Power State	EVE memory power loss
ESM_GROUP1_14	EDMA_TPCC_IRQ_ERR	PRCM Resets	System EDMA TPCC error [2]
ESM_GROUP1_15	EDMA_TC0_IRQ_ERR	PRCM Resets	System EDMA TC0 error [2]
ESM_GROUP1_16	EDMA_TC1_IRQ_ERR	PRCM Resets	System EDMA TC1 error [2]
ESM_GROUP1_17	DCC2_IRQ_ERROR	PRCM Resets	DCC 2 error [2]
ESM_GROUP1_18	DCC3_IRQ_ERROR	PRCM Resets	DCC 3 error [2]
ESM_GROUP1_19	DCC4_IRQ_ERROR	PRCM Resets	DCC 4 error [2]
ESM_GROUP1_20	DCC5_IRQ_ERROR	PRCM Resets	DCC 5 error [2]
ESM_GROUP1_21	DCC6_IRQ_ERROR	PRCM Resets	DCC 6 error [2]
ESM_GROUP1_22	DCC7_IRQ_ERROR	PRCM Resets	DCC 7 error [2]
ESM_GROUP1_23	CAM_RST	PRCM Resets	VIP reset asserted[2]
ESM_GROUP1_24	CM_CORE_PWRON_RET_RST	PRCM Resets	Power On Retention reset asserted [2]
ESM_GROUP1_25	CM_CORE_RET_RST	PRCM Resets	Retention reset asserted [2]
ESM_GROUP1_26	CUSTEFUSE_RST	PRCM Resets	E-Fuse reset asserted [2]
ESM_GROUP1_27	DSP1_PWRON_RST	PRCM Resets	DSP1 Power On Reset asserted [2]
ESM_GROUP1_28	DSP1_RET_RST	PRCM Resets	DSP1 Retention reset asserted [2]
ESM_GROUP1_29	DSP1_RST	PRCM Resets	DSP1 reset asserted [2]
ESM_GROUP1_30	DSP1_SYS_RST	PRCM Resets	DSP1 system reset asserted [2]
ESM_GROUP1_31	DSP2_PWRON_RST	PRCM Resets	DSP2 Power On Reset asserted [2]
ESM_GROUP1_32	DSP2_RET_RST	PRCM Resets	DSP2 Retention reset asserted [2]
ESM_GROUP1_33	DSP2_RST	PRCM Resets	DSP1 reset asserted [2]
ESM_GROUP1_34	DSP2_SYS_RST	PRCM Resets	DSP1 system reset asserted [2]
ESM_GROUP1_35	DSS_RET_RST	PRCM Resets	DSS Retention reset asserted [2]

[1] Channels mapped to *ESM_GROUP1_0* to *ESM_GROUP1_3* are IRQ/DMA Crossbar outputs; any signals coming to these crossbars can be routed to ESM for monitoring purpose. For more information, see the *TDA3x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 1.0* (SPRUHQ7).

[2] For more information, see the *TDA3x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 1.0* (SPRUHQ7).

Table 1. ESM Input Interrupt Channel Mapping (continued)

ESM Channel No	Channel Request Tag/Signal Name	Channel/Signal Type	Description/Activity Detected
ESM_GROUP1_36	DSS_RST	PRCM Resets	DSS reset asserted [2]
ESM_GROUP1_37	EMU_EARLY_PWRON_RST	PRCM Resets	Emulation early Power On reset asserted.[2]
ESM_GROUP1_38	EMU_PWRON_RST	PRCM Resets	Emulation Power On reset asserted [2]
ESM_GROUP1_39	EMU_RST	PRCM Resets	Emulation reset asserted [2]
ESM_GROUP1_40	EVE_CPU_RST	PRCM Resets	VIP reset asserted [2]
ESM_GROUP1_41	EVE_PWRON_RST	PRCM Resets	EVE Power On reset asserted [2]
ESM_GROUP1_42	EVE_RST	PRCM Resets	EVE reset asserted [2]
ESM_GROUP1_43	IPU_CPU0_RST	PRCM Resets	IPU CPU0 reset asserted [2]
ESM_GROUP1_44	IPU_CPU1_RST	PRCM Resets	IPU CPU1 reset asserted [2]
ESM_GROUP1_45	IPU_PWRON_RST	PRCM Resets	IPU system Power On reset asserted [2]
ESM_GROUP1_46	IPU_RET_RST	PRCM Resets	IPU system retention reset asserted [2]
ESM_GROUP1_47	IPU_RST	PRCM Resets	IPU system reset asserted [2]
ESM_GROUP1_48	L3INIT_PWRON_RST	PRCM Resets	L3 Power On reset asserted [2]
ESM_GROUP1_49	L3INIT_RST	PRCM Resets	L3 reset asserted [2]
ESM_GROUP1_50	L4PER_PWRON_RET_RST	PRCM Resets	L4PER Power On Retention reset asserted [2]
ESM_GROUP1_51	L4PER_RET_RST	PRCM Resets	L4PER Retention reset asserted [2]
ESM_GROUP1_52	L4PER_RST	PRCM Resets	L4PER reset asserted.[2]
ESM_GROUP1_53	DEBUGSS_STANDBY	Standby Signal	Debug Sub System is in Standby
ESM_GROUP1_54	RESRVED	RESRVED	RESRVED
ESM_GROUP1_55	RESRVED	RESRVED	RESRVED
ESM_GROUP1_56	LPRM_PWRON_RST	PRCM Resets	LPRM Power On reset asserted [2]
ESM_GROUP1_57	LPRM_RST	PRCM Resets	LPRM reset asserted [2]
ESM_GROUP1_58	LPRM_SECURE_RST	PRCM Resets	LPRM Secure reset asserted [2]
ESM_GROUP1_59	WKUPAON_PWRON_RST	PRCM Resets	WKUPAON Power On reset asserted [2]
ESM_GROUP1_60	WKUPAON_RST	PRCM Resets	WKUPAON reset asserted [3]
ESM_GROUP1_61	WKUPAON_SYS_PWRON_RST	PRCM Resets	WKUPAON System Power On reset asserted [3]
ESM_GROUP1_62	CM_CORE_AON_PWRON_RST	PRCM Resets	CM_CORE_AON Power On reset asserted [3]
ESM_GROUP1_63	CM_CORE_AON_RST	PRCM Resets	CM_CORE_AON reset asserted [3]
ESM_GROUP1_64	COREAON_PWRON_RST	PRCM Resets	COREAON_PWRON reset asserted [3]
ESM_GROUP1_65	COREAON_RST	PRCM Resets	COREAON reset asserted [3]
ESM_GROUP1_66	DPLL_CORE_LOCK	PLL Lock signal	DPLL CORE state changed from Locked to Unlocked
ESM_GROUP1_67	DPLL_DDR_LOCK	PLL Lock signal	DPLL DDR state changed from Locked to Unlocked
ESM_GROUP1_68	DPLL_EVE_VID_DSP_LOCK	PLL Lock signal	DPLL EVE_VID_DSP state changed from Locked to Unlocked
ESM_GROUP1_69	DPLL_DSP_GMAC_LOCK	PLL Lock signal	DPLL DSP_GMAC state changed from Locked to Unlocked
ESM_GROUP1_70	DPLL_PER_LOCK	PLL Lock signal	DPLL PER state changed from Locked to Unlocked
ESM_GROUP1_71	PD_CUSTFUSE_PGODD	Power State	CUSTFUSE Logic power loss
ESM_GROUP1_72	PD_DSP1_PGOOD	Power State	DSP1 Logic power loss
ESM_GROUP1_73	PD_DSP2_PGOOD	Power State	DSP2 logic power loss
ESM_GROUP1_74	PD_DSS_PGOOD	Power State	DSS logic power loss
ESM_GROUP1_75	PD_EMU_PGOOD	Power State	EMU logic power loss
ESM_GROUP1_76	PD_EVE_PGOOD	Power State	Eve Logic power loss
ESM_GROUP1_77	PD_IPU_PGOOD	Power State	IPU logic power loss

[3] For more information, see the *TDA3x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 1.0* (SPRUHQ7).

Table 1. ESM Input Interrupt Channel Mapping (continued)

ESM Channel No	Channel Request Tag/Signal Name	Channel/Signal Type	Description/Activity Detected
ESM_GROUP1_78	PD_L3INIT_PGOOD	Power State	L3INIT logic power loss
ESM_GROUP1_79	PD_L4PER_PGOOD	Power State	L4PER logic power loss
ESM_GROUP1_80	PD_DSP1_L1_AGOOD	Power State	DSP1 L1 memory power loss
ESM_GROUP1_81	PD_DSP1_L2_AGOOD	Power State	DSP1 L2 memory power loss
ESM_GROUP1_82	PD_DSP2_L1_AGOOD	Power State	DSP2 L1 memory power loss
ESM_GROUP1_83	PD_DSP2_L2_AGOOD	Power State	DSP2 L2 memory power loss
ESM_GROUP1_84	PD_CORE_IPU_L2_AGOOD	Power State	IPU L2 memory power loss
ESM_GROUP1_85	PD_CORE_IPU_UNICACHE_AGOOD	Power State	IPU Unicache power loss
ESM_GROUP1_86	PD_CORE_OCMC_RAM_AGOOD	Power State	OCMC Ram power loss
ESM_GROUP1_87	PD_CAM_VIP_AGOOD	Power State	VIP memory power loss
ESM_GROUP1_88	DSP1_IDLEREQ	IDLE Request	DSP1 IDLE Request
ESM_GROUP1_89	DSP2_IDLEREQ	IDLE Request	DSP2 IDLE Request
ESM_GROUP1_90	DSS_IDLEREQ	IDLE Request	DSS IDLE Request
ESM_GROUP1_91	EMIF_IDLEREQ	IDLE Request	EMIF IDLE Request
ESM_GROUP1_92	EVE_IDLEREQ	IDLE Request	EVE IDLE Request
ESM_GROUP1_93	GPMC_IDLEREQ	IDLE Request	GPMC IDLE Request
ESM_GROUP1_94	IPU_IDLEREQ	IDLE Request	IPU IDLE Request
ESM_GROUP1_95	L4_CFG_IDLEREQ	IDLE Request	L4_CFG IDLE Request
ESM_GROUP1_96	L4_PER1_IDLEREQ	IDLE Request	L4_PER1 IDLE Request
ESM_GROUP1_97	L4_PER2_IDLEREQ	IDLE Request	L4_PER2 IDLE Request
ESM_GROUP1_98	L4_PER3_IDLEREQ	IDLE Request	L4_PER3 IDLE Request
ESM_GROUP1_99	L4_WKUP_IDLEREQ	IDLE Request	L4_WKUP IDLE Request
ESM_GROUP1_100	MCASP_IDLEREQ	IDLE Request	MCASP IDLE Request
ESM_GROUP1_101	MMU_IDLEREQ	IDLE Request	MMU IDLE Request
ESM_GROUP1_102	OCMC_RAM_IDLEREQ	IDLE Request	OCMC RAM IDLE Request
ESM_GROUP1_103	GMAC_SW_STANDBY	Standby Signal	GMAC IP is in Standby
ESM_GROUP1_104	DSP1_STANDBY	Standby Signal	DSP1 is in Standby
ESM_GROUP1_105	DSP2_STANDBY	Standby Signal	DSP1 is in Standby
ESM_GROUP1_106	DSS_STANDBY	Standby Signal	DSS IP is in Standby
ESM_GROUP1_107	EVE_STANDBY	Standby Signal	EVE is in Standby
ESM_GROUP1_108	IEEE1500_STANDBY	Standby Signal	IEEE1500 IP is in Standby
ESM_GROUP1_109	IPU_STANDBY	Standby Signal	IPU is in Standby
ESM_GROUP1_128	EDMA_TC0_STANDBY	Standby Signal	EDMA TC0 is in Standby
ESM_GROUP1_112	VIP_STANDBY	Standby Signal	VIP IP is in Standby
ESM_GROUP1_113 – ESM_GROUP1_127	RESERVED	RESERVED	RESERVED

2.2 ESM Register Summary

Table 2 describes register summary of ESM instance.

Table 2. ESM Registers Mapping Summary

Register Name	Description	Address Offset
ESMIEPSR1	Enable influence of channels from 0 to 31 on the *esm_error* pin	0x0000 0000
ESMIEPCR1	Disable influence of channels from 0 to 31 on the *esm_error* pin	0x0000 0004
ESMIESR1	Enable interrupt generation for channels from 0 to 31	0x0000 0008
ESMIECR1	Disable interrupt generation for channels from 0 to 31	0x0000 000C
ESMILSR1	Set channels from 0 to 31 to generate high level interrupt	0x0000 0010
ESMILCR1	Set channels from 0 to 31 to generate low level interrupt	0x0000 0014
ESMSR1	Status of channels from 0 to 31 pending interrupt	0x0000 0018
ESMEPSR	*esm_error* pin status register	0x0000 0024
ESMIOFFHR	Highest pending interrupt channel number for channels mapped to generate high level interrupt	0x0000 0028
ESMIOFFLR	Highest pending interrupt channel number for channels mapped to generate low level interrupt	0x0000 002c
ESMLTCR	Current Low Time Counter value	0x0000 0030
ESMLTCPR	Low Time Preload Value for the *esm_error* pin	0x0000 0034
ESMEKR	ESM Error Key Register	0x0000 0038
ESMIEPSR4	Enable influence of channels from 32 to 63 on *esm_error* pin	0x0000 0040
ESMIEPCR4	Disable influence of channels from 32 to 63 on *esm_error* pin	0x0000 0044
ESMIESR4	Enable interrupt generation for channels from 32 to 63	0x0000 0048
ESMIECR4	Disable interrupt generation for channels from 32 to 63	0x0000 004C
ESMILSR4	Set channels from 32 to 63 to generate high level interrupt	0x0000 0050
ESMILCR4	Set channels from 32 to 63 to generate low level interrupt	0x0000 0054
ESMSR4	Status of channels from 32 to 63 pending interrupt	0x0000 0058
ESMIEPSR7	Enable influence of channels from 64 to 95 on the *esm_error* pin	0x0000 0080
ESMIEPCR7	Disable influence of channels from 64 to 95 on the *esm_error* pin	0x0000 0084
ESMIESR7	Enable interrupt generation for channels from 64 to 95	0x0000 0088
ESMIECR7	Disable interrupt generation for channels from 64 to 95	0x0000 008C
ESMILSR7	Set channels from 64 to 95 to generate high level interrupt	0x0000 0090
ESMILCR7	Set channels from 64 to 95 to generate low level interrupt	0x0000 0094
ESMSR7	Status of channels from 64 to 95 pending interrupt	0x0000 0098
ESMIEPSR10	Enable influence of channels from 96 to 127 on the *esm_error* pin	0x0000 00C0
ESMIEPCR10	Disable influence of channels from 96 to 127 on the *esm_error* pin	0x0000 00C4
ESMIESR10	Enable interrupt generation for channels from 96 to 127	0x0000 00C8
ESMIECR10	Disable interrupt generation for channels from 96 to 127	0x0000 00CC
ESMILSR10	Set channels from 96 to 127 to generate high level interrupt	0x0000 00D0
ESMILCR10	Set channels from 96 to 127 to generate low level interrupt	0x0000 00D0
ESMSR10	Status of channels from 96 to 127 pending interrupt	0x0000 00D8

ESMIOFFHR and ESMIOFFLR gives the channel number with the highest pending interrupt from the respective interrupt level pool. Within *esm_group1*, channel 0 has highest priority and channel 127 has lowest priority. Channel number from register value is decoded as shown in Table 3.

Table 3. ESMIOFFHR and ESMIOFFLR Registers

ESMIOFFHR and ESMIOFFLR Register Value	Decoded Channel Number
0x0000_0000	No pending interrupt
0x0000_0001 to 0x0010_0000	Channel 0 to 31
0x0100_0001 to 0x0110_0000	Channel 32 to 63
0x1000_0001 to 0x1010_0000	Channel 64 to 95
0x1100_0001 to 0x1110_0000	Channel 96 to 127

3   ESM – Usage

ESM is used to report certain activity on monitored signals on the external pin to the external device or through interrupt to the device. If ESM is configured to generate interrupt on certain activity detection, 128 lines monitored under *esm_group1* can be divided into two groups: high level and low level interrupt lines. Each of which can generate one interrupt to the device. The highest pending interrupt from each group can be obtained from registers *ESMIOFFHR* and *ESMIOFFLR*, respectively. Inside *esm_group1*, channel 0 has the highest priority and channel 127 has the lowest priority. If multiple errors happened at the same time, these offset registers will contain a channel number of the highest pending interrupt from that group. Status of other interrupts and errors gets latched into the respective ESM Status Register 4 (ESMSRx). Irrespective of whether the channel is enabled or not, its status gets latched to the respective ESMSRx status register on detection of certain transitions on that channel. The software should be aware of this condition before doing further analysis and logging of the error condition.

Any activity detected on the monitored signals and reported by ESM does not necessarily mean error. For example, ESM is monitoring the *DSP1_RST* signal, so the following two scenarios may happen:

- DSP1 is running an application and DSP1 reset is asserted. This may result in system failure since DSP1 was functional and monitoring the device (external or internal); take appropriate action to keep system in fail-safe state.
- DSP1 has finished running the application and an application running on IPU has requested for DSP1 boot. This triggers transition on the *DSP1_RST* signal; this activity is detected and reported by ESM. As this is not an erroneous condition, the system does not to take any action.

To avoid running into such conditions (explained in the second bullet above), before performing any operation that triggers transition on the monitored signals, the application should first disable those monitored signals and then perform the operation. After performing the required operation, the application can then enable these signals, if needed.

If interrupts are used to report an activity, software needs to do following configuration before enabling interrupts for monitored signals:

- Input channels mapped from *ESM_GROUP1_0* to *ESM_GROUP1_3* under *esm_group1* are IRQ/DMA crossbars. Any system interrupt/DMA request coming to these crossbars can be routed to ESM for monitoring purpose. This can be done by configuring the respective control module registers (CTRL_CORE_ESM_GROUP1_0, CTRL_CORE_ESM_GROUP1_1, CTRL_CORE_ESM_GROUP1_2 and CTRL_CORE_ESM_GROUP1_3). For more information, see the *TDA3x SoC for Advanced Driver Assistance Systems (ADAS) Silicon Revision 1.0* (SPRUHQ7).

If the external pin is used to report an activity, the software needs to do the following configurations before starting to use the external error pin:

1. Use the *esm_error* pin for indication of certain activity on the monitored channels to the external device from *vout1_d[6]* or *mdio_d* by configuring the appropriate selection for the *MUXMODE* bits in the pin configuration register.

2. Select weak pull-up by configuring the *PULLTYPESELECT* bits in the pin configuration register.

3. Enable weak pull-up selected by configuring the *PULLUDENABLE* bits in the pin configuration register.

4. Enable Receive mode by configuring the *INPUTENABLE* bits in the pin configuration register. This is necessary for reading the state of the *esm_error* pin through the ESMEPSR register.

Since interrupt handling logic and error handling logic are independent, you can choose to generate interrupt or generate error on the *esm_error* pin or both on detection of certain activity for that particular channel. The application can generate interrupt for a wider variety of signals for internal monitoring purposes within the device and may be generating errors on the external *esm_error* pin sent out to external device for few critical signals. If the *esm_error* pin is used to report an activity to an external device, external devices should report back to the application to request reset of the *esm_error* pin after reception of the detected activity on monitored signals.

Since ESM can generate two groups of interrupts, *ESM_IRQ_HIGH* (high level interrupt) or *ESM_IRQ_LOW* (low level interrupt), you can divide the monitored signals into two groups and have one interrupt generated to the device per group. In this way, you can divide monitored signals of equal importance and criticality into one group.

If warm reset occurs when the *esm_error* pin is low, the ESM Low-Time Counter Register (ESMLTCR) will reset to its default value to 0x3FFF. You need to write a sequence of key values to the ESMEKR[3:0] register as follows:

- Write key values '0xA' and then '0x5' or '0x0' to the ESMEKR[3:0] register, which disables ESM. Then, enable ESM through the Power, Reset, and Clock Management Registers (PRCM).

- Enable ESM through the PRCM registers. Then write key values '0xA' and '0x5' or '0x0' to the ESMEKR[3:0] register. This will enable ESM.

In both cases mentioned above, the status of all the errors and interrupts that have happened are kept in their respective ESMSR status registers for later use and analysis.

Any activity detected gets latched irrespective of its enablement for error or interrupt generation. Before enabling any signal for monitoring purpose, check whether that signal is already in error state or not. If the monitored signal is already indicating an error condition, it will generate an error or interrupt as soon as it is enabled for monitoring. To avoid running into such conditions, clear the signal at the source and then its status at ESM module.

You should not enable channels from *esm_group1* that are marked as RESERVED in Table 1 for the interrupt generation or do not enable their influence on the *esm_error* pin.

3.1 Programming Guide

This section illustrates several of the ways in which the ESM can be utilized to perform signal monitoring and error reporting using the Starterware device driver APIs. Available Starterware APIs are:

```
1.  void ESMInitialize(uint32_t baseAddr);
•   This API is used to initialize of ESM module and this API will clear status of all
interrupts.
2.  void ESMSetMode(uint32_t baseAddr, uint32_t mode);
•   This API is used to configure operation mode of ESM module.
3.  void ESMSetInfluenceOnErrPin(uint32_t baseAddr, uint32_t intrSrc, uint32_t influence);
•   This API is used to set the influence of interrupt on esm_error pin.
4.  int32_t ESMSetErrPinLowTimePreload(uint32_t baseAddr, uint32_t lowTime);
•   This API is used to configure the low time counter pre-load value.
5.  uint32_t ESMGetCurrErrPinLowTimeCnt(uint32_t baseAddr);
•   This API is used to get the current value of low time counter.
6.  uint32_t ESMGetErrPinStatus(uint32_t baseAddr);
•   This API is used to get the current status of esm_error pin.
7.  void ESMResetErrPin(uint32_t baseAddr);
•   This API is used to reset the esm_error pin.
8.  void ESMEnableIntr(uint32_t baseAddr, uint32_t intrSrc);
•   This API is used to enable interrupt.
9.  void ESMDisableIntr(uint32_t baseAddr, uint32_t intrSrc);
•   This API is used to disable interrupt.
10. void ESMSetIntrPriorityLvl(uint32_t baseAddr, uint32_t intrSrc, uint32_t intrPriorityLvl);
•   This API is used to set interrupt level.
11. uint32_t ESMGetIntrStatus(uint32_t baseAddr, uint32_t intrSrc);
•   This API is used to get the interrupt status.
12. int32_t ESMGetGroupIntrStatus(uint32_t baseAddr, uint32_t grpNum, esmGroupIntrStatus_t
*intrstatus);
•   This API is used to get the interrupt/error status for a group.
13. void ESMClearIntrStatus(uint32_t baseAddr, uint32_t intrSrc);
•   This API is used to clear the interrupt status.
14. uint32_t ESMGetHighPriorityLvlIntrStatus(uint32_t baseAddr);
•   This API is used to get the highest level pending interrupt in high level interrupts.
15. uint32_t ESMGetLowPriorityLvlIntrStatus(uint32_t baseAddr);
•   This API is used to get the highest level pending interrupt in low level interrupts.
```

For more details about Starterware APIs, see the Starterware user's guide [3].

3.1.1 Example

This section illustrates the way in which ESM can be used to monitor signals and report on detection of certain activity on them. The Starterware device driver APIs are used to configure ESM.

A group of signals is monitored by ESM. On detection of certain activity on these signals, ESM then reports it to the device through interrupts (known as low and high level interrupts) and to the external device through the external *esm_error* pin. For illustration purpose, the following signals are monitored: *DPLL_DDR_LOCK*, *DCC1_IRQ_ERROR*, *ESM_GROUP1_0_IRQ_0*, *ESM_GROUP1_3_DMA_3*, *PD_DSP1_L1_AGOOD* and *PD_DSP1_PGOOD*. The application running on DSP1 is configuring ESM for monitoring the above signals. In this case, *DPLL_DDR_LOCK* and *DCC1_IRQ_ERROR* are mapped to generate low level interrupts and *ESM_GROUP1_0_IRQ_0*, *ESM_GROUP1_3_DMA_3*, *PD_DSP1_L1_AGOOD* are mapped to generate high level interrupts as they are error interrupts, DMA requests and memory logic power loss signals, respectively. Configuring ESM for this purpose is explained in following steps:

1. Initialize the ESM module.

   As any activity detected on monitored signals gets latched to status register ESMSRx irrespective of whether it is enabled or not. Therefore, it is necessary to clear these statuses before using ESM. This is done by the following Starterware API:

   ```
   /* Initialize ESM module */
   ESMInitialize(SOC_ESM_BASE);
   ```

2. Configure ESM mode of operation to Normal mode.

   This is done by the following Starterware API:

   ```
   /* Configure ESM mode of operation to Normal mode*/
   ESMSetMode(SOC_ESM_BASE, ESM_OPERATION_MODE_NORMAL);
   ```

3. Configure ESM interrupt levels and the *esm_error* pin.

   As explained, *DPLL_DDR_LOCK* and *DCC1_IRQ_ERROR* are mapped to generate *ESM_IRQ_LOW* interrupt. *ESM_GROUP1_0_IRQ_0*, *ESM_GROUP1_3_DMA_3*, *PD_DSP1_L1_AGOOD* are mapped to generate the *ESM_IRQ_HIGH* interrupt. This is done as:

   ```
   /* Configure Low Level interrupts */
   ESMSetIntrPriorityLvl(SOC_ESM_BASE, ESM_GROUP1_INTR_SRC_DPLL_DDR_LOCK,
                         ESM_INTR_PRIORITY_LEVEL_LOW);
   ESMSetIntrPriorityLvl(SOC_ESM_BASE, ESM_GROUP1_INTR_SRC_DCC1_IRQ_ERROR,
                         ESM_INTR_PRIORITY_LEVEL_LOW);
   ESMSetIntrPriorityLvl(SOC_ESM_BASE, ESM_GROUP1_INTR_SRC_ESM_GROUP1_0_IRQ_0,
                         ESM_INTR_PRIORITY_LEVEL_HIGH);
   ESMSetIntrPriorityLvl(SOC_ESM_BASE, ESM_GROUP1_INTR_SRC_ESM_GROUP1_3_DMA_3,
                         ESM_INTR_PRIORITY_LEVEL_HIGH);
   ESMSetIntrPriorityLvl(SOC_ESM_BASE, ESM_GROUP1_INTR_SRC_PD_DSP1_L1_AGOOD,
                         ESM_INTR_PRIORITY_LEVEL_HIGH);
   ```

   The *PD_DSP1_PGOOD* signal is configured to report an activity on the *esm_error* pin. On detection of DSP1 logic power loss, DSP1 will not be able to service any interrupts. Also configure Low Time Counter preload to the appropriate value so that an external device would not see this transition on the *esm_error* pin as a glitch. This is done by the following Starterware API:

   ```
   /* Configure Low Time Period for esm_error pin */
   ESMSetErrPinLowTimePreload(SOC_ESM_BASE, 0x3U);
   ```

4. Interrupt/DMA Crossbar and external pad configuration.

   For illustrative purpose, *TESOC_IRQ_DONE* for the ESM IRQ crossbar and *CT_TBR_DREQ* for the ESM DMA crossbar are being used. Configure *ESM_GROUP1_0 IRQ* crossbar for *TESOC_IRQ_DONE* interrupts and configure *ESM_GROUP1_3* DMA crossbar for the *CT_TBR_DREQ* DMA event.

   Before using the *esm_error* pin as an indication to the activity detected on monitored channels, configure the I/O pad first. In this scenario, the mdio_d pad for error reporting to the external device is used. This is done as:

   ```
   /* Set PAD configuration parameters */
   pad_config_t esm_pad_config = {CTRL_CORE_PAD_IO_MDIO_D,
                                  CTRL_CORE_PAD_IO_MDIO_D_MUXMODE_ESM_ERROR_5,
                                  CTRL_CORE_PAD_IO_MDIO_D_PULLUDENABLE_ENABLE,
                                  CTRL_CORE_PAD_IO_MDIO_D_PULLTYPESELECT_PULL_UP,
                                  CTRL_CORE_PAD_IO_MDIO_D_INPUTENABLE_ENABLE,
                                  CTRL_CORE_PAD_IO_MDIO_D_SLEWCONTROL_FAST,
   ```

```
                                              0xff};
/* PAD configuration */
cntrl_core_pad_configuration(SOC_CORE_PAD_IO_REGISTERS_BASE, &esm_pad_config);
```

Any activity getting latched prior to enabling that channel for influence on the *esm_error* pin or for interrupt generation, can generate interrupt or report it on the external pin as soon as that channel is enabled. Therefore, configure the IRQ/DMA crossbar and the *esm_error* pad only after initializing ESM to prevent the system from receiving this report of activity detection.

5. IRQ crossbar configuration for *ESM_IRQ_HIGH* and *ESM_IRQ_LOW* group interrupts.

The *ESM_IRQ_HIGH* interrupt to interrupt no. 44 of DSP1 and *ESM_IRQ_LOW* interrupt to interrupt no. 45 of DSP1 are being mapped, respectively. This is done as:

```
/* Following code will configure IRQ crossbar for ESM_IRQ_HIGH */
if (irq_xbar_success == IRQXBARConnect(SOC_IRQ_DMARQ_CROSSBAR_REGISTERS_BASE,
                                       CPU_DSP1,
                                       XBAR_INST_DSP1_IRQ_44,
                                       ESM_IRQ_HIGH))
{
    /* IRQ crossbar configuration successful */
    Intc_IntEnable(44U);
    Intc_Init();
    Intc_IntRegister(44U,
                    (IntrFuncPtr) AppESMHighGrpIntrISR, 0);
    Intc_IntPrioritySet(44U, 1, 0);
    Intc_SystemEnable(44U);
}
else
{
    /* IRQ crossbar configuration unsuccessful */
}

/* Following code will configure IRQ crossbar for ESM_IRQ_LOW */
if (irq_xbar_success == IRQXBARConnect(SOC_IRQ_DMARQ_CROSSBAR_REGISTERS_BASE,
                                       CPU_DSP1,
                                       XBAR_INST_DSP1_IRQ_45,
                                       ESM_IRQ_LOW))
{
    /* IRQ crossbar configuration successful */
    Intc_IntEnable(45U);
    Intc_Init();
    Intc_IntRegister(45U,
                    (IntrFuncPtr) AppESMLowGrpIntrISR, 0);
    Intc_IntPrioritySet(45U, 1, 0);
    Intc_SystemEnable(45U);
}
else
{
    /* IRQ crossbar configuration unsuccessful */
}
```

6. Enable ESM channels and signals for monitoring.

This step enables channels for monitoring. Before enabling channels for monitoring purpose, check to see if that channel is already in error condition. If channel is in indicating error condition, then clear it at the source as well as at the ESM module. This can be done as follows:

```
/* check for error condition on DCC1_IRQ_ERROR channel */
if (1U == ESMGetIntrStatus(SOC_ESM_BASE,   ESM_GROUP1_INTR_SRC_DCC1_IRQ_ERROR))
{
    /* Activity detected on DCC1_IRQ_ERROR signal */
    /* clear interrupt at DCC 1 module*/
    /* clear interrupt status at ESM module */
    ESMClearIntrStatus(SOC_ESM_BASE, ESM_GROUP1_INTR_SRC_DCC1_IRQ_ERROR);
}
/* Enable interrupt for monitoring */
ESMEnableIntr(SOC_ESM_BASE, ESM_GROUP1_INTR_SRC_DCC1_IRQ_ERROR);
/* check for error condition on PD_DSP1_PGOOD channel */
if (1U == ESMGetIntrStatus(SOC_ESM_BASE,   ESM_GROUP1_INTR_SRC_PD_DSP1_PGOOD))
{
    /* Activity detected on PD_DSP1_PGOOD signal */
    /* clear error condition at DSP 1 */
    /* clear interrupt status at ESM module */
    ESMClearIntrStatus(SOC_ESM_BASE, ESM_GROUP1_INTR_SRC_PD_DSP1_PGOOD);
}
/* Enable interrupt for monitoring */
/* Configure influence on esm_error pin */
ESMSetInfluenceOnErrPin(SOC_ESM_BASE, ESM_GROUP1_INTR_SRC_PD_DSP1_PGOOD, TRUE);
```

The code above only shows the error condition check for the *DCC1_IRQ_ERROR* and *PD_DSP1_PGOOD* channels. Repeat this step for each channel that needs to be monitored.

7. After step 6, all configurations needed for the reporting of detection of an activity on monitored channels by ESM are done. An activity detected on any of the monitored channels will be reported to the device through interrupts or to an external device through the *esm_error* pin. Consider following scenarios:

(a) DSP1 got *ESM_IRQ_LOW* interrupt mapped to interrupt no. 45.

This means an activity is detected on the *DPLL_DDR_LOCK* or *DCC1_IRQ_ERROR* signal.

```
/* ESM_IRQ_LOW isr */
void AppESMLowGrpIntrISR (void *handle)
{
    uint32_t lowGrpPendIntr;
    /* Get channel number of highest pending interrupt in ESM_IRQ_LOW group */[
    lowGrpPendIntr = ESMGetLowPriorityLvlIntrStatus (SOC_ESM_BASE);
    /* Check for individual status of monitored signals */
    if (1U == ESMGetIntrStatus(SOC_ESM_BASE, ESM_GROUP1_INTR_SRC_DPLL_DDR_LOCK))
    {
        /* Activity detected on DPLL_DDR_LOCK signal */
        /* Clear Interrupt Status */
        ESMClearIntrStatus(SOC_ESM_BASE, ESM_GROUP1_INTR_SRC_DPLL_DDR_LOCK);
    }
    if (1U == ESMGetIntrStatus(SOC_ESM_BASE,
                                  ))
    {
        /* Activity detected on DCC1_IRQ_ERROR signal */
        /* Clear Interrupt Status */
        ESMClearIntrStatus(SOC_ESM_BASE, ESM_GROUP1_INTR_SRC_DCC1_IRQ_ERROR);

    }
}
```

If getting individual status for the monitored signals is not needed by the application, an application can then get the highest pending interrupt channel number for a particular group by using *ESMGetLowPriorityLvlIntrStatus()* or *ESMGetHighPriorityLvlIntrStatus()*, respectively.

(b) DSP1 got the *ESM_IRQ_HIGH* interrupt mapped to interrupt no. 44.

This means an activity is detected on the *ESM_GROUP1_0_IRQ_0*, *ESM_GROUP1_3_DMA_3* or *PD_DSP1_L1_AGOOD* signal.

```
/* ESM_IRQ_HIGH isr */
void AppESMHighGrpIntrISR (void *handle)
{
    uint32_t highGrpPendIntr;
    /* Get channel number of highest pending interrupt in ESM_IRQ_LOW group */
    highGrpPendIntr = ESMGetHighPriorityLvlIntrStatus (SOC_ESM_BASE);
    /* Check for individual status of monitored signals */
    if (1U == ESMGetIntrStatus(SOC_ESM_BASE,   ESM_GROUP1_INTR_SRC_ESM_GROUP1_0_IRQ_0))
    {
        /* Activity detected on DPLL_DDR_LOCK signal */
        /* Clear Interrupt Status */
        ESMClearIntrStatus(SOC_ESM_BASE, ESM_GROUP1_INTR_SRC_ESM_GROUP1_0_IRQ_0);
    }
    if (1U == ESMGetIntrStatus(SOC_ESM_BASE,
                               ESM_GROUP1_INTR_SRC_ESM_GROUP1_3_DMA_3))
    {
        /* Activity detected on DCC1_IRQ_ERROR signal */
        /* Clear Interrupt Status */
        ESMClearIntrStatus(SOC_ESM_BASE, ESM_GROUP1_INTR_SRC_ESM_GROUP1_3_DMA_3);
    }
    if (1U == ESMGetIntrStatus(SOC_ESM_BASE,
                               ESM_GROUP1_INTR_SRC_PD_DSP1_L1_AGOOD))
    {
        /* Activity detected on DCC1_IRQ_ERROR signal */
        /* Clear Interrupt Status */
        ESMClearIntrStatus(SOC_ESM_BASE, ESM_GROUP1_INTR_SRC_PD_DSP1_L1_AGOOD);
    }
}
```

(c) External device received report or alert on the *esm_error* pin.

After receiving a report on the *esm_error* pin, an external device should send an acknowledgment back to the DSP1 running application requesting reset of the *esm_error* pin. The external device should then take the appropriate actions to keep the system in fail-safe state. Reset of the *esm_error* pin is done by the following Starterware API:

```
/* Reset esm_error pin */
ESMResetErrPin(SOC_ESM_BASE);
```

If the external device needs further details about the error reported by ESM, such as the channel that caused this transition on the *esm_error* pin, then it can be reported by the application running DSP1. The application running on DSP can then check the status set for each individual signal for which influence on the *esm_error* pin is enabled. For example:

```
/* Check for individual status of monitored signals */
if (1U == ESMGetIntrStatus(SOC_ESM_BASE,   ESM_GROUP1_INTR_SRC_PD_DSP1_PGOOD))
{
    /* Activity detected on PD_DSP1_PGOOD signal */
}
```

4 References

1. *TDA3xx Starterware User's Guide*: This is a part of the starterware release that can be found under the 'docs' folder in the starterware package, which is available for download at https://www.ti.com/securesoftware/docs/autopagepreview.tsp?opnId=12132.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have *not* been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		Applications	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		