Texas Instruments

Application Report
Mar 2016

Safety Features on VisionSDK

Automotive Processor Business Unit

ABSTRACT

This document describes the integration of various safety modules in TDAx family of SoCs in VisionSDK. This document is intended to highlight key points like boot-flow, memory layouts, etc. to be addressed during integration of these modules into any system.

Contents

Figures

No table of figures entries found.

Tables

Abbreviations

connID: Connection Identifier. Used in L3 Firewalls to control access permissions to different targets.

CPU: Refers to the processor rather than the subsystem. eg: CPU in ARP32 refers to the RISC core. ARP32 subsystem refers to the RISC Core, VCOP, internal MMUs, internal EDMAs, etc.

DCC: Dual Clock Comparator – HW module available on TDA3x

ECC: Error Correcting Code module. Supports Single Bit Correction and Double Bit Detection.

ESM: Error Signaling Modules – HW module available on TDA3x

IPC: Inter-processor communication

L3FW: Level 3 Interconnect Firewall – HW module available on TDAx SoCs to control accesses to slave modules from different masters based on multiple parameters.

MPU: Memory Protection Unit on C66x

FFI: Freedom From Interference. The scope of this document is limited only to methods of achieving FFI on memory regions and limited to only two levels of protection – QM and ASIL.

kB, **MB**: 1024 bytes and 1024*1024 bytes respectively.

RTI: Real Time Interrupt module available on TDA3x. This module implements the Windowed Watchdog Timer functionality. Any mention of RTI, implicitly refers to this functionality.

SBL: Secondary Boot Loader

WWDT: Windowed Watchdog Timer. This term is used interchangeably with RTI in this document.

XMC: Extended Memory Controller on C66x

1 Introduction

Table 1. Feature List

	TDA2X (SR 1.1)	TDA2X (SR 2.0)	TDA2EX (SR 1.0)	TDA2EX (SR 2.0)	TDA3X (SR 1.0)	TDA3X (SR 2.0)
EMIF ECC	NS	YES	NS	YES	NS	YES
FFI (DSP CPU) - XMC	YES	YES	YES	YES	YES	YES
FFI (DSP EDMA) - L3FW	NS	NS	NS	NS	YES	YES
FFI (EVE) - L3FW	NS	NS	NA	NA	YES	YES
ESM	NA	NA	NA	NA	YES	YES
DCC	NA	NA	NA	NA	NS	YES
RTI	NA	NA	NA	NA	YES	YES
IPU ECC	NA	NA	NA	NA	YES	YES
DSP Parity	YES	YES	YES	YES	YES	YES

NA: Feature is not available in hardware
NS: Feature is not available due to silicon errata

Following is a brief summary of these features

- EMIF ECC
  - On TDAx SoC, EMIF1 supports ECC features for DDR memories. This supports error handlers for one-bit and two-bit errors. This feature is available only SR 2.0 versions for TDA2x, TDA2Ex and TDA3x SoCs. It is not available on SR 1.1 versions due to silicon erratum i882.
- FFI (DSP CPU) - XMC
  - To implement FFI on C66x DSP CPU on TDA2x/TDA2Ex/TDA3x, we use the XMC module to control the read and write permissions of different tasks.
- FFI (DSP EDMA) – L3FW
  - To implement FFI on accesses made by EDMA within the C66x subsystem, we use the L3 firewalls since EDMA accesses do not go through XMC.
- FFI (EVE) – L3FW
  - To implement FFI on accesses made by EDMA and the CPU within the EVE subsystem, we use the L3 firewalls.
- ESM
  - This module allows the software to track multiple events in the SoC using a single interrupt handler and is available only on TDA3x.
- DCC
  - This module allows the software to track drifts between two clock sources and is available only on TDA3x SR 2.0. In TDA3x SR 1.0, DCC is unusable due to a silicon erratum.

- RTI
  - This module provides the WWDT functionality and is available only on TDA3x.
- IPU ECC
  - IPU Unicache on TDA3x support ECC functionality of IPU L2RAM and IPU Unicache.
- DSP Parity
  - This module is a part of C66x subsystem and does simple parity checks on L1 program memory, L1 program cache, L2 RAM and L2 cache.

2 Vision SDK updates

2.1 Build flow

For safety specific features, following variables are enabled in Rules.make

- ECC_FFI_INCLUDE
  - To search for all software changes relevant to these, user can search for following terms and files:
    ECC_FFI_INCLUDE
    BspSafetyOsal_setSafetyMode
    BspSafetyOsal_getSafetyMode
    vision_sdk/src/utils_common/utils_l3fw.c
    vision_sdk/src/utils_common/utils_xmc.c
    vision_sdk/src/utils_common/utils_emif_ecc.c
    vision_sdk/src/utils_common/utils_ecc_c66x.c
    vision_sdk/src/utils_common/safety_osal.c
    vision_sdk/src/utils_common/tda3xx/utils_ipu_ecc.c
  - This enables the consolidated memory map change to allow for ECC and FFI.
  - This enables error handlers for interrupts from XMC and L3FW.
  - This enables the ECC error handlers
  - This also enables IPU ECC error handlers
  - This also enables the DSP parity checks and corresponding error handlers
  - To ensure that ECC and DSP parity checks work correctly, SBL should be built with following variables defined correctly in following files based on platform in **sbl_lib_config_tda2xx.h / sbl_lib_config_tda2ex.h / sbl_lib_config_tda3xx.h**
    - Ensure **ECC_FFI_INCLUDE** is set to **yes** in **vision_sdk/Rules.make** while building SBL.
    - **SBL_LIB_CONFIG_DSP1_PARITY_CHECK = 1**
    - **SBL_LIB_CONFIG_DSP2_PARITY_CHECK = 1**
    - **SBL_LIB_CONFIG_ENABLE_IPU_RAM_ECC = 1**
    - **SBL_LIB_CONFIG_ENABLE_EMIF_ECC = 1**
    - **SBL_LIB_CONFIG_EMIF_ECC_START_ADDR1 (TDA2x)**
      SBL_LIB_CONFIG_EMIF_ECC_START_ADDR1_15X15 (TDA3x 15x15)
      SBL_LIB_CONFIG_EMIF_ECC_START_ADDR1_12X12 (TDA3x 12x12)
      SBL_LIB_CONFIG_EMIF_ECC_END_ADDR1 (TDA2x)
      SBL_LIB_CONFIG_EMIF_ECC_ END _ADDR1_15X15 (TDA3x 15x15)
      SBL_LIB_CONFIG_EMIF_ECC_END_ADDR1_12X12 (TDA3x 12x12)
      The value of these variables will be explained in 4.3
    - **SBL_LIB_CONFIG_EMIF_ECC_REG1_RANGE_TYPE**
      For VisionSDK implementation, this is set to **EMIF_ECC_ADDR_RANGE_WITHIN**. For

custom implementations, this can be changed to **EMIF_ECC_ADDR_RANGE_OUTSIDE**. These correspond to values 1 and 0 for **REG_ECC_ADDR_RGN_PROT** in **EMIF_ECC_CTRL_REG** register. Refer to TDAx TRM for further details.

- DCC_ESM_INCLUDE
  - DCC errors are tracked using ESM. This variable enables the example integration of these two modules in VisionSDK.
- RTI_INCLUDE
  - This enables the example integration of RTI WWDT functionality on TDA3x.
- Build commands for VisionSDK for different platforms
  - `make -s BUILD_DEPENDANCY_ALWAYS=yes VSDK_BOARD_TYPE=TDA2XX_EVM ECC_FFI_INCLUDE=yes`
  - `make -s BUILD_DEPENDANCY_ALWAYS=yes VSDK_BOARD_TYPE=TDA2EX_EVM ECC_FFI_INCLUDE=yes`
  - `make -s BUILD_DEPENDANCY_ALWAYS=yes VSDK_BOARD_TYPE=TDA3XX_EVM ECC_FFI_INCLUDE=yes DCC_ESM_INCLUDE=yes RTI_INCLUDE=yes`
- Build commands for SBL for different platforms
  - `make -s BUILD_DEPENDANCY_ALWAYS=yes VSDK_BOARD_TYPE=TDA2XX_EVM ECC_FFI_INCLUDE=yes sbl_sd`
  - `make -s BUILD_DEPENDANCY_ALWAYS=yes VSDK_BOARD_TYPE=TDA2EX_EVM ECC_FFI_INCLUDE=yes sbl_sd`
  - `make -s BUILD_DEPENDANCY_ALWAYS=yes VSDK_BOARD_TYPE=TDA3XX_EVM ECC_FFI_INCLUDE=yes DCC_ESM_INCLUDE=yes RTI_INCLUDE=yes sbl_qspi_sd`
- AppImage generation
  - In case of TDA3x, all safety features are enabled in SBL when ECC_FFI_INCLUDE is used. Therefore, ensure that **calculate_crc** is set to 1 in **MulticoreImageGen_tda3xx.sh** or **MulticoreImageGen_tda3xx.bat** when generating AppImages

2.2 New plugin and use-cases

A new plugin called “**safeframecopy**” has been added to demonstrate “Freedom from Interference (FFI)”. This is based on the older “**framecopy**” plugin in VisionSDK. This plugin is intended to be executed only on DSP and EVE. Example use-case based on this plugin on DSP and EVE is available through the standard VisionSDK use-case menu. This plugin switches between CPU based copy and EDMA based copy for alternate frames. EDMA based copy in forced for a special scenario explained in 6.2.

2.3 FFI mode for AlgorithmLink

A new API has been included for AlgorithmLink - AlgorithmLink_setPluginFFIMode(). This is used to set the FFI mode to ASIL or QM for Algorithm links. By default, all algorithms are provided ASIL (full) access. The “**safeframecopy**” plugin registers itself as QM using this API.

2.4 RTI/DCC/ESM

- This is enabled in the "**framecopy**" and "**safeframecopy**" based use-cases for TDA3x.
- RTI monitoring code is available in the folder **vision_sdk/examples/tda2xx/src/modules/rti/**. Other framework updates are under the macro **RTI_INCLUDE**.
- DCC and ESM related code is under the macro **DCC_ESM_INCLUDE** and in following files **vision_sdk/src/utils_common/src/tda3xx/utils_dcc.c** **src/utils_common/src/tda3xx/utils_esm.c.**

3 EMIF ECC, IPU ECC, DSP Parity

3.1 Hardware requirements

- EMIF ECC
  - Available only on SR 2.0 and higher revisions of TDA2x, TDA2Ex, TDA3x.
  - All write accesses to ECC protected region in EMIF must be 32bit aligned
  - ECC protected region should be “primed” by doing a write to complete region before performing any reads.
  - Since DSP L1 cache is not “write-allocate”, boot time stack pointer should be in L2SRAM to ensure no un-aligned writes to EMIF. L2 cache must be enabled before moving to a stack in EMIF region.
  - Priming can be done only by using EDMA or system DMA or by making non-cached “memset” from CPU. Cached “memset” will not work since cache will always do read before write which will cause uncorrectable errors.
- IPU ECC
  - Available only on TDA3x
  - IPU L2RAM and IPU Cache need to be “primed” by doing writes without any reads.
  - Unicache “priming” is done by doing a cache preload of a 64kB (size of Unicache) section using the Unicache maintenance registers. During this step, software must ensure following steps:
    - Ensure cache is enabled
    - “Priming” code and its stack is placed in non-cached section. This is to prevent errors due to code/data caching during the “priming” step.
    - Do full cache write-back and invalidate.
    - Enable ECC generation and ECC checks using **ECC_CFG** register in Unicache. Refer TRM for details.
    - Preload a 64kB (size of Unicache) section from a cacheable region using **CACHE_MAINT**, **CACHE_MTSTART** and **CACHE_MTEND** registers in IPU Unicache. This completes the “priming”.
    - Switch back to normal code execution from cached regions
- DSP Parity
  - DSP L2RAM needs to be “primed” using 128bit write to ensure valid parity.
  - Since EMIF ECC needs boot time to be in L2SRAM (refer bullet point above), L2SRAM “priming” must happen in SBL.

3.2 VisionSDK and SBL implementation

- "Priming"
  - SBL performs priming for EMIF, IPU and DSP based on macros defined in 2.1. All code can be found using these macros.
  - EMIF, IPU L2SRAM and DSP L2SRAM are "primed" using EDMA.
  - It should be ensured that EMIF ECC start address matches the start address of data memory (IPU1_1_DATA_MEM in the current implementation, this may change in future).
  - IPU Unicache is primed using maintenance register as explained in 3.1
- Ensure "32bit" aligned write access for EMIF ECC
  - A15/M4
    - A15 and M4 support "write-back, write-allocate" cache. This ensures all write accesses to EMIF are cache line aligned. This is ensured in the A15 and M4 SYS/BIOS configuration files.
  - DSP
    - DSP L1 cache is not "write-allocate". This is enabled at reset.
    - DSP L2 cache is "write-allocate". This is not enabled at reset.
    - If default ".stack" section is in ECC protected DDR region, this can generate errors.
      To avoid this, VisionSDK configuration for DSP ensures ".stack" section which is used as stack during initial boot of DSP is kept in L2RAM.
    - Using SYS/BIOS cache and reset hooks, L2 cache and DSP parity checks are enabled before any other code executes.
  - EVE
    - EVE does not have any data-cache. As a result, EVE code and data must not be kept in ECC protected regions. Although, theoretically it might be possible to ensure only 32bit write accesses from EVE, due to compiler optimizations, typical coding optimizations, it is not practical to do so.
- Simplifications to avoid adding complexity to VisionSDK
  - IPC, Remote Log, Link Stats, VIP/VPE descriptors are kept in non-cached section
    - Since, the region is non-cached, software changes are needed to IPC to ensure all write accesses are 32bit aligned. To avoid this, this section is not kept in ECC protected region.
    - These are kept in a single section to prevent memory fragmentation and avoid need for extra regions in L3 firewalls and DSP XMC.
  - Debugging
    - Breakpoints on IPU are 16bit instructions. Using this will cause ECC errors.
    - When combined with FFI features, the code section may not be always writable. This will prevent user from adding or removing a breakpoint.

- To simplify this and allow easy debugging, all code sections and EVE code and data sections are kept contiguous and separate from IPU/DSP data sections. This will allow users to move code sections out of ECC protected regions easily for debugging during the development phase

4 Freedom From Interference (FFI)

4.1 Introduction

The ISO 26262 “Road Vehicles – Functional Safety” standard for automotive products defines the Automotive Safety Integrity Level (ASIL) risk classification scheme – ASIL A through ASIL D in increasing order of safety criticality. A typical ECU contains a mix of software modules with different criticalities including non-critical software which is classified as QM (Quality Managed). ISO 26262 allows co-existence of software modules with different criticalities as long as the system demonstrates “Freedom from Interference (FFI)” across the different modules. FFI ensures that errors in one module do not propagate or trigger errors in another – potentially more critical – software module. The system should address interference on three fronts – Memory usage, Time usage and Communication channels.

In VisionSDK, we demonstrate FFI on memory usage by DSP, EVE and their respective EDMAs. We use XMC (Extended memory controller) for achieving FFI for DSP CPU accesses to memories outside the DSP sub-system like OCMC RAMs and DDR. For DSP EDMA, EVE CPU and EVE EDMA, we use L3 firewalls for the same purpose. In case of DSP internal memories, C66x MPU (Memory Protection Unit) can be used to achieve FFI on L1 and L2 memories. MPU is not used VisionSDK since L1 memories are used only as cache and L2 memory is used mainly as scratch. Example usage of MPU is available in starterware example at **starterware_/examples/xmc_mpu_app**.

4.2 Hardware requirements

- L3 Firewall
  - L3 Firewall features:
    - Control memory access using connID (master identification) and privilege mode (USER and SUPERVISOR) using N-regions.
    - Regions can overlap. Higher numbered region gets precedence over lower numbered regions.
    - The privilege mode setting for any region applies to all connID enabled in that region. Eg: If USER accesses are to be prevented for DSP for a region, USER accesses from M4 will also get blocked for that region.
  - EMIF L3 firewall supports 8 regions on TDA2x and TDA2Ex, 16 regions on TDA3x. VisionSDK uses only 8 regions for keeping code common across platforms.
  - L3 Firewall permissions cannot be changed at run-time on TDA2x (SR 1.1 and SR 2.0) and TDA2Ex (SR 1.1)
  - EVE supports only single privilege level which is marked as “USER” at L3 interconnect. As a result, to ensure EVE accesses always reach EMIF, background region (0th region) in L3 firewall should allow all USER and SUPERVISOR accesses. Alternately, we can use additional regions to be defined in L3 firewall which may or may not be possible due to other system constraints.
  - DSP1 CPU and DSP1 EDMA have same connID.

    - Any attempt to block EDMA writes using connID will block DSP CPU write as well as cache writes.
    - This is usually not a problem, since DSP EDMA will inherit USER and SUPERVISOR permissions from DSP CPU and privilege level based access control is sufficient.
  - DSP2 CPU and DSP2 EDMA have same connID.
    - Any attempt to block EDMA writes using connID will block DSP CPU write as well as cache writes.
    - This is usually not a problem, since DSP EDMA will inherit USER and SUPERVISOR permissions from DSP CPU and privilege level based access control is sufficient.
  - EVE1/2/3/4 CPU and corresponding EDMA have same connID
    - If FFI is attempted on multiple EVEs, firewall permissions should be changed atomically from a single core and software should implement mechanism to track firewall mode changes on all EVE cores.
  - If firewall marks a region as read-only, user cannot put breakpoints in this region. So software must ensure that code sections for different cores are kept together so that permissions for these can be easily turned ON/OFF using a single regions for easier debugging.
- XMC
  - Features
    - Support 16 regions of the format defined by “address” and “size” where “address” is “size” aligned, “size” is a power of 2 and “size” is greater than or equal to 4096.
    - Access to different regions can be controlled using USER or SUPERVISOR mode
    - EDMA accesses do not go through XMC and need to be controlled using L3FW
  - Guard-bands at the boundaries of EMIF and OCMC RAM need to be implemented to prevent prefetch accesses going into invalid region. These regions can be skipped if software ensures that no access occurs within 4kB from the start and 4kB from the end of the EMIF and OCMC RAMs.
  - XMC requirement of “address” being “size” aligned can put limitations on memory segments in software. If “address” is not “size” aligned for a memory segment, software would need to set up multiple XMC segments to protect a single contiguous memory region.
- EVE
  - EVE does not support privilege levels like USER and SUPERVISOR. As a result, you cannot differentiate between VisionSDK framework which is assumed ASIL and QM algorithms. The only way to achieve FFI from QM tasks is to mark ASIL memory sections as read-only in L3FW before starting QM tasks.
  - Interrupts should be disabled during QM algorithm execution. This ensures that scheduler does not execute in QM mode as it will not have access to all relevant data structures.
  - Since interrupts are disabled, functions like Task_sleep() will not execute for correct time.
    - This causes errors for some use-case like RTI. Refer to 6.2 for details.

4.3 VisionSDK implementation

- FFI is demonstrated only on DSP for TDA2x and TDA2Ex and for DSP and EVE on TDA3x.
- VisionSDK framework is assumed to be ASIL. All code on A15/M4 is assumed to be ASIL.
- QM memories are writable by all, ASIL memories will be protected in QM algorithms on DSP and EVE only.
- FFI is implemented only in “**safeframecopy**” based use-cases.
- EVE does not have a data cache and, therefore, its stack is kept in internal memory. FFI cannot be achieved using hardware mechanism for EVE internal memories. Software mechanisms can be employed for stack integrity checks, but VisionSDK examples do not implement these.
- All tasks on DSP and EVE in VisionSDK share a common stack as a framework simplification. As a result, algorithms’ stack (“DSP QM STACK”) has to be kept in QM regions.
- LINK STATS are accessed by QM algorithms. The corresponding data section needs to be mapped as QM.
- For EVE, .bss section cannot be kept far away in memory from .const and other data sections due to linker constraints. EDMA library on EVE does a .bss access from the QM safeframecopy plugin. To prevent changes in EDMA library, EVE data section is marked as QM.
- Memory map (based on ECC and FFI constraints)

Table 2. Vision SDK Memory Map

EVE CODE/DATA
IPU/DSP CODE
IPU/DSP DATA
A15 CODE/DATA
ECC + ASIL HEAP
ECC + QM HEAP
DSP QM STACK
NON ECC + ASIL HEAP
NON ECC + QM HEAP

  – SBL_LIB_CONFIG_EMIF_ECC_START/END macros for SBL will map to (start of “IPU/DSP CODE”) and (start of “NON ECC + ASIL HEAP” – 1) respectively. Start address must be 64kB aligned.

  – Memory map is defined in
  vision_sdk/build/tda2xx/mem_segment_definition_512mb_bios.xs
  vision_sdk/build/tda2ex/mem_segment_definition_512mb_bios.xs
  vision_sdk/build/tda3xx/mem_segment_definition_512mb.xs

  – Following data sections are added specifically for supporting ECC and FFI
  SR1_BUFF_ECC_ASIL_MEM
  SR1_BUFF_ECC_QM_MEM
  SR1_BUFF_NON_ECC_ASIL_MEM

- XMC segments

Table 3. XMC segments for FFI

ASIL 0x0000_0000 to 0x7FFF_FFFF
ASIL 0x8000_0000 to 0xFFFF_FFFF
4kB Guard Band at OCMC1 start
512kB Guard Band between OCMC1/2 (TDA2x only)
ECC + ASIL HEAP
ECC + QM HEAP + DSP QM STACK
NON ECC + ASIL HEAP
QM LINK STATS

  – XMC segments are set up in the function **Utils_xmcMpuInit()** in the file vision_sdk/src/utils_common/src/utils_xmc_mpu.c
  – It should be ensured that all XMC segments are 4 KB aligned otherwise this will result in assert error.

- L3 Firewall regions

Table 4. Regions for L3 firewall on EMIF

Full EMIF ASIL
ECC + ASIL HEAP
ECC + QM HEAP + DSP QM STACK
NON ECC + ASIL HEAP
IPC + LINK STATS + LOGS + VIP/VPE Descriptor QM
DSP1 DATA ASIL
DSP2 DATA ASIL
EVE DATA QM

  – These regions are set up in vision_sdk/src/utils_common/src/utils_l3fw.c. Refer to **L3FW_VSDK_REGION_xxx** macros.

- Algorithm Link
  – A new API has been included for AlgorithmLink - AlgorithmLink_setPluginFFIMode(). This is used to set the FFI mode to ASIL or QM for Algorithm links. By default, all algorithms are provided ASIL (full) access. The “safeframecopy” plugin registers itself as QM using this API.

- Safety OSAL
  – This layer provides two APIs – **BspSafetyOsal_setSafetyMode()** and **BspSafetyOsal_getSafetyMode()** – to allow users to switch the level of execution to QM or ASIL using appropriate arguments.
  – This interface is defined in **bsp_drivers_/include/safety_osal/bsp_safety_osal.h**
  – For VisionSDK, this layer is implemented in **vision_sdk/src/utils_common/safety_osal.c**
  – SYS/BIOS OS functions are assumed to be ASIL. Since these can be triggered even during QM tasks, we need to ensure that the BspOsal layer in **bsp_drivers_/src/osal/bsp_ osal.c** switches to ASIL mode before any OS function calls and restores back the QM mode at the end of OS function call.

- VisionSDK framework is assumed to be ASIL, certain framework commands are triggered from QM tasks. These function use the safety OSAL layer to temporarily move into ASIL mode and then go back to QM mode.

- Firewall register configuration
  - In case of TDA2x SR 1.1 and SR 2.0, TDA2Ex SR 1.1, EMIF firewall configuration is not allowed when there is activity on EMIF as per **Silicon errata i895**
    - To work around this, users should configure firewalls statically in boot-loader context which ensures no EMIF activity
    - VisionSDK does not follow this recommendation from point of view of software maintenance only. VisionSDK does firewall configuration only once during the system initialization – occurrence error is possible, but is rare and has not been observed in testing. There is no firewall reconfiguration in VisionSDK after the first initialization in accordance with the errata.
    - This constraint is not applicable to TDA3x
  - FFI on EVE requires run-time reconfiguration of firewall registers. Silicon Errata i895 prevents this on TDA2x SR 1.1 and SR 2.0, TDA2Ex SR 1.1. A work-around detailed below exists but not implemented in VisionSDK to simplify software
  - Brief details of work-around for i895 for FFI on EVE
    - Alias DDR memory space using DMM_LISA_MAP register. Eg: For a 512MB DDR, same memory can be accessed from 0x8000_0000 and 0xA000_0000.
    - All memory accessed at 0xA000_0000 should be marked as ASIL in firewall
    - By default, access to this aliased space is disabled through EVE MMU. 0x8000_0000 on EVE gets mapped to 0x8000_0000 by the EVE MMU.
    - When switching to an ASIL task, EVE MMU tables are re-mapped to access ASIL region. 0x8000_0000 should be mapped to 0xA000_0000 in EVE MMU.
  - In case of warm-reset, firewall configurations are not lost. Bootloader or application must ensure to reset firewall registers when booting after a warm reset.
    - VisionSDK resets the firewall before system initialization to ensure proper booting after warm reset. Refer to **vision_sdk/src/main_app/*/ipu1_0/src/main_ipu1_0.c** – Search for ECC_FFI_INCLUDE.

5 DCC/ESM (TDA3x only)

5.1 Hardware requirements

- DCC (Dual clock comparator)
  - DCC tracks the drift between two clock sources and generates an interrupt if the drift exceeds a specified threshold
  - Reference clock for DCC can be SYSCLK or external reference clock. Refer to TRM for additional details.
  - If the clock under test gets "gated", DCC will detect this as an error. Software must ensure DCC is turned off if the clock under test is expected to turn off. One such example is CPU clocks. If DSP goes to a low power state, corresponding DPLL clocks are turned off. If DCC is tracking drifts in DSP clock, it should be turned off before DSP enters low power mode.
- ESM
  - ESM muxes multiple events in the SoC to a single interrupt lines
  - DCC error interrupt is one of the events supported by ESM

5.2 Vision SDK integration

- DCC
  - Implemented **in vision_sdk/src/utils_common/tda3xx/utils_dcc.c**
  - VisionSDK example for DCC tracks DDR DPLL since it is never turned off in VisionSDK framework.
  - DCC is configured to track drifts more than 1% from 532MHz.
  - Reference clock source for DCC can be SYSCLK1, SYSCLK2 or XREF_CLK and is set using the enumeration **dccClkSrc0_t** from **starterware_/include/dcc.h**.
  - Different DCC modules support tracking of different clocks in the system. These are listed in the enumeration **dccClkSrc1_t** in the file **starterware_/include/tda3xx/soc_defines.h**. Application must ensure that correct enumeration is used when setting the test clocks.
- ESM
  - Implemented in **vision_sdk/src/utils_common/tda3xx/utils_esm.c**
  - This provides interface to register different callback function for different ESM events
  - Enumeration **esmGroup1IntrSrc_t** for ESM events is defined in **starterware_/include/tda3xx/soc_defines.h**
  - VisionSDK example track DCC error interrupt using ESM
  - ESM and DCC usage is triggered only in "**framecopy**" and "**safeframecopy**" based use-cases.

6 RTI/WWDT (TDA3x only)

6.1 Hardware requirements

- RTI – WWDT
  - RTI module implements the WWDT (Windowed Watchdog Timer) functionality.
  - If a WWDT is serviced outside its specified window or not serviced at all, the RTI module can generate an interrupt signal which can be routed to all CPUs in the system using the IRQ_CROSSBAR. Alternately, the expiry of an RTI can also generate a WARM reset on the SoC.
  - RTI1 is used by ROM bootloader and is set up to a time-out of 3 minutes. Application can re-use RTI1 if this time-out value is acceptable.
  - RTI2/3/4/5 can be used by software without any limitations. Software can set up timeout value as required. This timeout value cannot be changed once configured. Please refer to TRM for further details.

6.2 Vision SDK integration

- RTI task
  - Implemented in the folder **vision_sdk/examples/tda2xx/src/modules/rti**
  - Other changes are present under **RTI_INCLUDE** macro.
  - This task runs on all cores and registers for WWDT expiry interrupts from all RTI modules
  - IPU1_0/DSP1/DSP2/EVE1 setup and service RTI2/3/4/5 respectively in a periodic manner.
  - If any core other than IPU1_0 is unable to service the WWDT in the configured service-window, all cores receive the RTI interrupt.
    - On receiving this interrupt, IPU1_0 resets the corresponding core.
    - Other cores track this WWDT expiry and stop sending any further message to the expired core. This allows other frame-work to not hang-up.
  - If IPU1_0 is unable to service its RTI correctly, the entire SoC is reset.
  - To allow debugging with RTI, the emulation suspend lines from the CPU can be connected to the associated RTI modules to prevent RTI WWDT from continuing the timer when cores are in a debug-halt state.
  - A single file **rtiLink_tsk.c** is used to implement the task on all cores, the execution is changed on basis of **System_getSelfProcId()** which identifies the current CPU.
  - Each RTI is configured with a time-out of 4 seconds and a window size of 50% (2 seconds)
- Integration into a use-case. This section is specific to VisionSDK – any integration of RTI will not require this method in the final system but can be useful during development phase for debugging.
  - RTI WWDT servicing is enable only in “framecopy” and “safeframecopy” usecases.

- At the end of the use-case, a special programming sequence is to ensure following:
    - RTI tasks stop servicing the WWDT
    - SoC reset generation is not generated by RTI associated with IPU1_0
    - Change service window to 100% to allow reconfiguration if needed later
    - This is implemented in the function rti_service()
    - If any re-configuration of RTI register is needed, a different programming sequence is needed. This is implemented in **rti_setup()** under the condition **(RTIDwwdIsCounterEnabled() == TRUE)**

- Special constraints: RTI with FFI on EVE
    - FFI on EVE needs interrupts to be disabled. This causes Task_sleep() command to work incorrectly.
    - RTI implementation uses Task_sleep() to wait till WWDT service window is open. Since, Task_sleep() works incorrectly the WWDT associated with EVE can expire under some scenarios.
    - In the **safeframecopy** plugin, if RTI is enabled, we force the copy mode to always use EDMA instead of CPU. This ensures that the errors in sleep times are not large enough to cause WWDT expiry.

- IPC consideration in case of using RTI
    - When a core expires, the WWDT expiry interrupt handler on IPU1-0 resets the CPU corresponding to the WWDT.
    - In this case, the core which is in reset will not respond to any new messages.
    - There is a small window between core failure and WWDT expiry, where messages sent will not be acknowledged. This can result in IPC queue getting stuck and prevent software recovery.
    - Therefore, IPC waits must use time-outs and check for core status when time-out occurs to avoid indefinite waits. This is implemented in **vision_sdk/src/links_common/system/system_ipc_msgq.c**. Search for RTI_INCLUDE for relevant code.

- Warm reset recovery considerations in case of using RTI
    - If the WWDT corresponding to master core expires, the system is configured to undergo a warm-reset.
    - During a warm-reset, all register configurations are not lost. Significant among these are
        - Control modules registers
        - Interrupt crossbar registers
        - Firewall configurations
    - Software must ensure that to reset such register configurations to ensure that system boots up correctly after a warm-reset.

7 BSP and Starterware additions for FFI

- Safety OSAL
  - This layer provides two APIs – **BspSafetyOsal_setSafetyMode()** and **BspSafetyOsal_getSafetyMode()** – to allow users to switch the level of execution to QM or ASIL using appropriate arguments.
  - This interface is defined in **bsp_drivers_/include/safety_osal/bsp_safety_osal.h**
  - This is implemented in **bsp_drivers_/src/safety_osal/bsp_safety_osal.c**
  - BSP examples do not implement FFI. Therefore, the safety OSAL implementation in BSP uses only empty functions.
- USER and SUPERVISOR switch in DSP
  - Relevant code is available in
    starterware_/include/c66x/dsp_usrSpvSupport.h
    starterware_/system_config/c66x/dsp_usrSpvSupport.c
    starterware_/system_config/c66x/swenr.asm
  - Using the C66x Memory Protection Unit (MPU) and Extended memory controller (XMC), SW can set up differential access permissions to L1/L2/L3 and DDR memories based on DSP CPU mode.
  - DSP CPU supports two modes – USER and SUPERVISOR. At reset, the CPU is in SUPERVISOR mode. The active mode is available in the CXM bits of the TSR register.
  - Current mode can be queried using the **DSP_getCpuMode()** API or changed using the **DSP_setCpuMode()** API.
  - Implementation details:
    - To switch the CPU mode from SUPERVISOR to USER or vice-versa, we use the SWENR instruction and the corresponding handler are used.
    - The SWENR handler is setup in the when **DSP_setCpuMode()** is called for the first time. The handler address is set up in the **REP** register.
    - To change the CPU mode, we execute the **SWENR** instruction with argument of 0 or 1. When the CPU jumps to the handler, the current **TSR** is copied to **NTSR**. The handler changes the value of **CXM** bit in **NTSR** to 0 or 1 to switch to SUPERVISOR or USER mode respectively based on the argument.
  - The handler then jumps back to the function which executed the **SWENR** instruction using the **NRP** pointer. This causes the **NTSR** (with new operating mode) to be copied to the **TSR** register. This completes the switch of CPU mode. Normal software can resume at this point.
- Summary of features added to SBL
  - "Priming" of EMIF ECC protected regions
    - ECC regions must be 64 kB aligned and have length in multiples of 64 kB
    - Start and End address are considered inclusive. eg: To define a 64kB region at address 0x8000_0000, start address must be 0x8000_0000 and end address must be 0x8000_FFFF.

- "Priming" of IPU L2RAM and Unicache in TDA3x for ECC.
- "Priming" of DSP L2RAM for parity checking.
  - SBL assumes all start of all code/data sections in L2SRAM to be 16 byte aligned and length to be a multiple of 16 bytes.
- IPU cache is set to write-back, write-allocate mode in TDA3x to allow ECC to work correctly.