Application Report
SPRAC41–October 2017

Safety Features on VisionSDK

Chaitanya Ghone, Rishabh Garg

ABSTRACT

This application report describes the integration of various safety modules in TDAx family of System-on-Chips (SoCs) in VisionSDK. This document is intended to highlight key points like boot-flow, memory layouts, and so forth to be addressed during integration of these modules into any system.

Contents

List of Figures

List of Tables

Trademarks

1 Introduction

Table 1. Features 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	YES	YES	YES
FFI (EVE) - L3FW	NS	NS	NA	NS	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

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

The following is a brief summary of these features:

- EMIF ECC
  - On TDAx SoC, EMIF1 supports ECC for DDR memories. It supports single bit error correction and double bit error detection. This feature is available only SR 2.0 versions for TDA2x, TDA2Ex and TDA3x SoCs. It is not available on SR 1.1 or SR1.0 versions due to silicon erratum i882.
- FFI (DSP CPU) - XMC
  - The XMC sub-module in the C66x DSP subsystem can be used to achieve FFI for DSP CPU tasks. The XMC provides ability to set different read-write permissions for different tasks based on the CPU mode.
- FFI (DSP EDMA) – L3FW
  - L3 firewalls must be used to achieve FFI for DSP EDMA access since these accesses cannot be controlled via XMC as in the case of DSP CPU.
- FFI (EVE) – L3FW
  - L3 firewalls must be used to achieve FFI for EVE CPU and EVE EDMA accesses.
- 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 supports ECC for IPU L2RAM and IPU Unicache.
- DSP EDC
  - This module is a part of C66x subsystem and supports parity checks for L1 Program memory/cache and supports single-bit error correction/double-bit error detection on L2 memory/cache.

2 Vision SDK Updates

The below changes are described in reference to Vision SDK 3.0 release. The file paths, API names, variable names, and so forth might change in future releases.

2.1 Build Flow

For safety specific features, the following variables are enabled in the particular cfg.mk (depending on the platform, OS, use cases, and so forth). For example, for enabling safety features for BIOS use cases on TDA3xx device, changes are made in vision_sdk/apps/configs/tda3xx_evm_bios_all/cfg.mk.

- ECC_FFI_INCLUDE
  - To search for all software changes relevant to these, search for following terms and files:
    ECC_FFI_INCLUDE
    BspSafetyOsal_setSafetyMode
    BspSafetyOsal_getSafetyMode
    vision_sdk/links_fw/src/rtos/utils_common/src/utils_l3fw.c
    vision_sdk/links_fw/src/rtos/utils_common/src/utils_xmc_mpu.c
    vision_sdk/links_fw/src/rtos/utils_common/src/utils_emif_ecc.c
    vision_sdk/links_fw/src/rtos/utils_common/src/utils_ecc_c66x.c
    vision_sdk/links_fw/src/rtos/utils_common/src/safety_osal.c
    vision_sdk/links_fw/src/rtos/utils_common/src/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 the particular cfg.mk 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 only)
    - SBL_LIB_CONFIG_EMIF_ECC_START_ADDR1_15X15 (TDA3x 15x15 only)
    - SBL_LIB_CONFIG_EMIF_ECC_START_ADDR1_12X12 (TDA3x 12x12 only)
    - SBL_LIB_CONFIG_EMIF_ECC_END_ADDR1 (TDA2x only)
    - SBL_LIB_CONFIG_EMIF_ECC_ END _ADDR1_15X15 (TDA3x 15x15 only)
    - SBL_LIB_CONFIG_EMIF_ECC_END_ADDR1_12X12 (TDA3x 12x12 only)
  - The value of these variables will be explained in Section 4.4.
  - 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. For further details, see the device-specific TRM.
- 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 all MAKEAPPNAME=apps MAKECONFIG=tda2xx_evm_bios_all ECC_FFI_INCLUDE=yes BUILD_DEPENDENCY_ALWAYS=yes
  - make -s all MAKEAPPNAME=apps MAKECONFIG=tda2ex_evm_bios_all ECC_FFI_INCLUDE=yes BUILD_DEPENDENCY_ALWAYS=yes
  - make -s all MAKEAPPNAME=apps MAKECONFIG=tda3xx_evm_bios_all ECC_FFI_INCLUDE=yes BUILD_DEPENDENCY_ALWAYS=yes DCC_ESM_INCLUDE=yes RTI_INCLUDE=yes
- Build commands for SBL for different platforms
  - make -s sbl MAKEAPPNAME=apps MAKECONFIG=tda2xx_evm_bios_all ECC_FFI_INCLUDE=yes
  - make -s sbl MAKEAPPNAME=apps MAKECONFIG=tda2ex_evm_bios_all ECC_FFI_INCLUDE=yes
  - make -s sbl MAKEAPPNAME=apps MAKECONFIG=tda3xx_evm_bios_all ECC_FFI_INCLUDE=yes DCC_ESM_INCLUDE=yes RTI_INCLUDE=yes"
- AppImage generation
  - AppImages can be generated by using the below commands. Please note that CRC check needs to be enabled for TDA3xx device:
    - make -s appimage MAKEAPPNAME=apps MAKECONFIG=tda2xx_evm_bios_all
    - make -s appimage MAKEAPPNAME=apps MAKECONFIG=tda2ex_evm_bios_all
    - make -s appimage MAKEAPPNAME=apps MAKECONFIG=tda3xx_evm_bios_all CRC_CALCULATE=yes

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 execute-based copy and EDMA-based copy for alternate frames. For a special scenario explained in Section 6.2, only EDMA based copy is used.

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

- These are enabled in the “framecopy” and “safeframecopy” based use-cases for TDA3x.
- RTI monitoring code is available in the folder vision_sdk/apps/src/rtos/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/links_fw/src/rtos/utils_common/src/tda3xx/utils_dcc.c, vision_sdk/links_fw/src/rtos/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 32-bit 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 causes uncorrectable errors.
- IPU ECC
  - Available only on TDA3x
  - IPU L2RAM and IPU Cache needs 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 the following steps:
    - Ensure cache is enabled
    - “Priming” code and its stack is placed in a non-cached section. This is to prevent errors due to code and 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. For further details, see the device-specific TRM.
    - 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 (see the 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
  - IPU Unicache is primed using maintenance register as explained in Section 3.1
- Ensure the “32-bit” 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 the ECC protected DDR region, this can generate errors. To avoid this, the 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 32-bit write accesses from EVE, due to compiler optimizations, typical coding optimizations, it may not be 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 32-bit 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 16-bit instructions. Using this causes 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 allows you to easily move code sections out of ECC protected regions 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 define the Automotive Safety Integrity Level (ASIL) risk classification scheme – ASIL A through ASIL D in increasing the order of safety criticality. A typical ECU contains a mix of software modules with different criticalities including non-critical software, which is classified as quality managed (QM). 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. The application report discusses how one can achieve FFI for memory accesses on TDAx SoCs.

4.2 FFI on TDAx SoCs

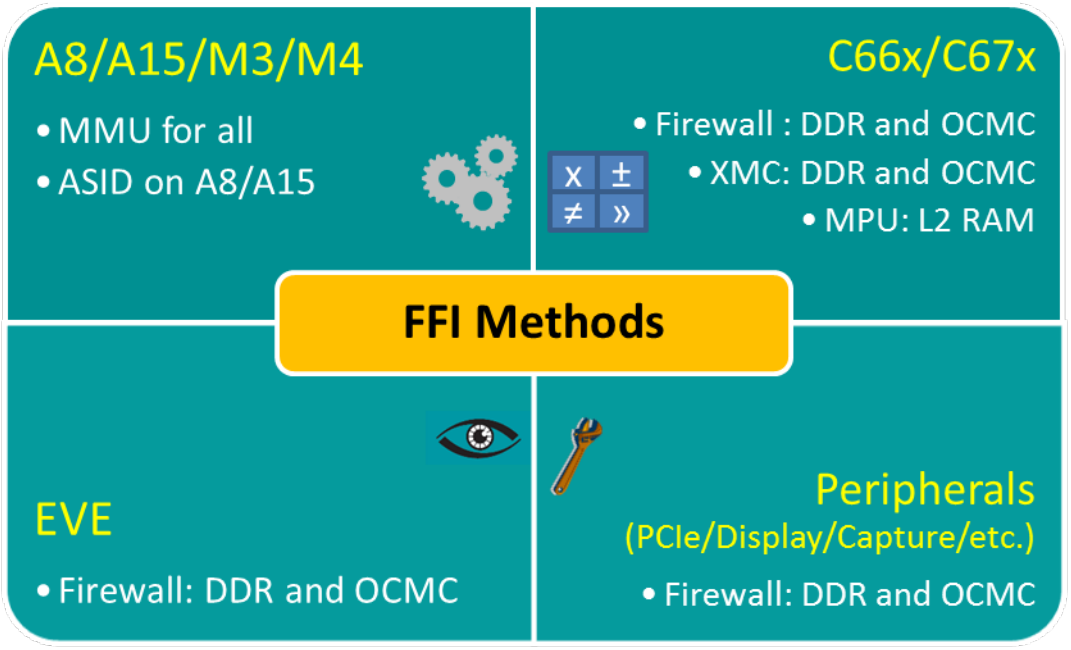

Figure 1. Hardware Support for FFI on TDAx SoC

To achieve FFI on the heterogeneous compute system, like TDA2x/TDA2Ex/TDA3x, a hybrid approach is needed. Figure 1 shows how to achieve FFI on different IPs in the SoCs.

- A15
  - A15 MMU allows switching of MMU tables for different tasks based on "Address-Space IDentifiers" (ASIDs). These allows application to achieve FFI on memory accesses made by different tasks by setting different access permissions for each task.
- M4
  - In case of M4, FFI between ASIL/QM can only be achieved by switching MMU tables at task-switch boundaries.
- DSP
  - DSP provides XMC to control memory accesses to EMIF and OCMC RAM
  - For internal memories, Memory Protection Unit (MPU) serves the same purpose
  - XMC and MPU can differentiate access based on the CPU mode (USER vs SUPERVISOR) to allow FFI between tasks at two different ASIL levels
- EVE, EDMA, and other peripherals like VIP, DSS
  - L3 Firewalls can control accesses permissions for different master in the system based on ConnID. By switching firewall permissions a task boundaries, users can achieve FFI between for accesses made by these IPs

In VisionSDK, FFI is demonstrated on memory usage by DSP, EVE and their respective EDMAs only. 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 PDK CSL example at ti_components/drivers/pdk/packages/ti/csl/example/xmc_mpu/xmc_mpu_test_app..

4.3 Hardware Requirements

- L3 Firewall
  - L3 Firewall uses a 4-bit ConnID as defined in the *L3_MAIN Interconnect Functional Description* sub-section in the *Interconnect* chapter in the TDAx TRMs
  - 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. For example, 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 1.0)
  - 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, additional regions can be used 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
    - Supports 16 regions 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 the 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.

4.4 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.

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

- Memory map (based on ECC and FFI constraints)
  - 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/apps/build/tda2ex/mem_segment_definition_bios.xs
    vision_sdk/apps/build/tda2xx/mem_segment_definition_bios.xs
    vision_sdk/apps/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

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
  - XMC segments are set up in the function Utils_xmcMpuInit() in the file vision_sdk/links_fw/src/rtos/utils_common/src/utils_xmc_mpu.c

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

- L3 Firewall regions
  - These regions are set up in vision_sdk/links_fw/src/rtos/utils_common/src/utils_l3fw.c. See the 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.
  - For VisionSDK, this layer is implemented in vision_sdk/links_fw/src/rtos/utils_common/src/safety_osal.c
  - SYS/BIOS OS functions are assumed to be ASIL. Since these can be triggered even during QM tasks, ensure that the BspOsal layer in drivers/pdk/packages/ti/drv/vps/src/osal/tirtos/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 1.0 and SR 2.0, 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 – Under such conditions, error may occur 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 SR1.0. 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. For example, 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 in the following files:
    - vision_sdk/links_fw/src/rtos/bios_app_common/tda3xx/ipu1_0/src/main_common_ipu1_0.c
    - vision_sdk/links_fw/src/rtos/bios_app_common/tda2xx/ipu1_0/src/ipu_primary.c
    - vision_sdk/links_fw/src/rtos/bios_app_common/tda2ex/ipu1_0/src/ipu_primary.c

4.5 EMIF Interleaving

TDA2x supports two EMIF interfaces – EMIF1 and EMIF2. The SoC supports interleaving across the two EMIF interfaces using the Dynamic Memory Manager (DMM) module. If EMIF1 and EMIF2 are used in an interleaved manner, interleaving is supported at 128-bytes/256-byte/512-byte boundaries as defined in DMM_LISA_MAP_x and MA_MPU_LISA_MAP_x registers. Consider 128-byte interleaving – even numbered (0-indexed) sets of 128 bytes will use EMIF1 interface and odd numbered sets of 128 bytes will use EMIF2 interface.

VisionSDK reference implementation of FFI does not support firewall configuration when EMIF is set up interleaved mode.

Section 4.5.1 gives an example for firewall configuration in a system with EMIF interleaving.

4.5.1 512 MB DDR on EMIF1, 512 MB DDR on EMIF2, 128-Byte Interleaving Enabled

- Set up 128-byte EMIF interleaving in DMM and MA_MPU
  – LISA_MAP_0 = 0x8064_0300
    - This will map the address range 0x8000_0000 to 0xBFFF_FFFF to map to both EMIFs in an interleaved fashion
  – All other LISA_MAP_x are assumed to not override the LISA_MAP_0/1 configuration
- Assume that you want to setup a firewall region from 0x9000_0000 to 0x97FF_FFFF. In this case, the start and end address should be calculated as per the physical address in the memories connected to EMIF interface as follows:
  – Start address: (0x9000_0000 – 0x8000_0000)/2 = 0x0800_0000
  – End address: (0x97FF_FFFF – 0x8000_0000)/2 = 0x03FF_FFFF
  – The value configured in the START_REGION_x and END_REGION_x registers in the EMIF_OCP_FW or MA_MPU_NTTP_FW firewalls will need an additional shift by 2 as per the value in REGUPDATE_CONTROL.FW_ADDR_SPACE_MSB field in these firewalls
- Software should also ensure that both END_REGION_i_ENABLE_CORE0 and END_REGION_i_ENABLE_CORE1 bits are set in the END_REGION_i register to ensure that firewall configuration applies to both EMIF1 and EMIF2 interface.
- Behavior of all other firewalls is not affected by EMIF interleaving

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

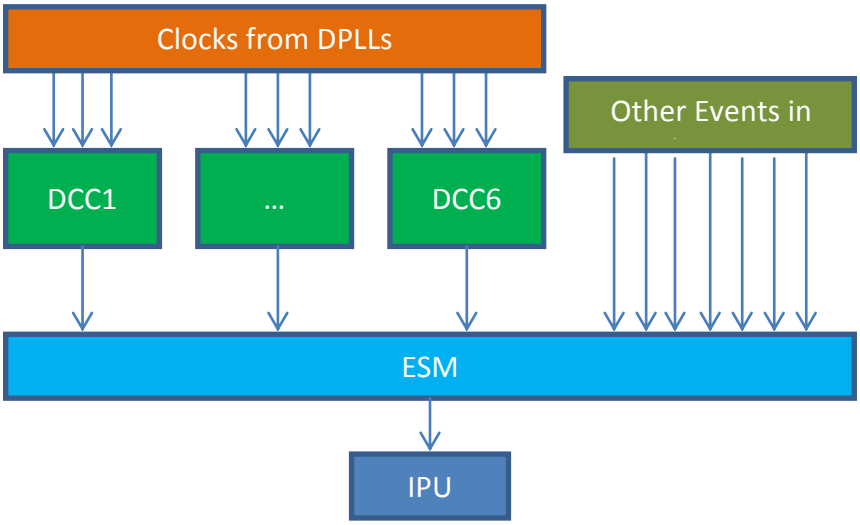

Figure 2. DCC and RTI Integration in VisionSDK

- DCC
  - Implemented in vision_sdk/links_fw/src/rtos/utils_common/src/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 532 MHz.
  - Reference clock source for DCC can be SYSCLK1, SYSCLK2 or XREF_CLK and is set using the enumeration dccClkSrc0_t from ti_components/drivers/pdk/packages/ti/csl/src/ip/dcc/V0/dcc.h.
  - Different DCC modules support tracking of different clocks in the system. These are listed in the enumeration dccClkSrc1_t in the file ti_components/drivers/pdk/packages/ti/csl/soc/tda3xx/cslr_soc_defines.h. Application must ensure that correct enumeration is used when setting the test clocks.
- ESM
  - Implemented in vision_sdk/links_fw/src/rtos/utils_common/src/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 ti_components/drivers/pdk/packages/ti/csl/soc/tda3xx/cslr_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 Windowed Watchdog Timer (WWDT) 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. For further details, see the device-specific TRM.

6.2 Vision SDK Integration

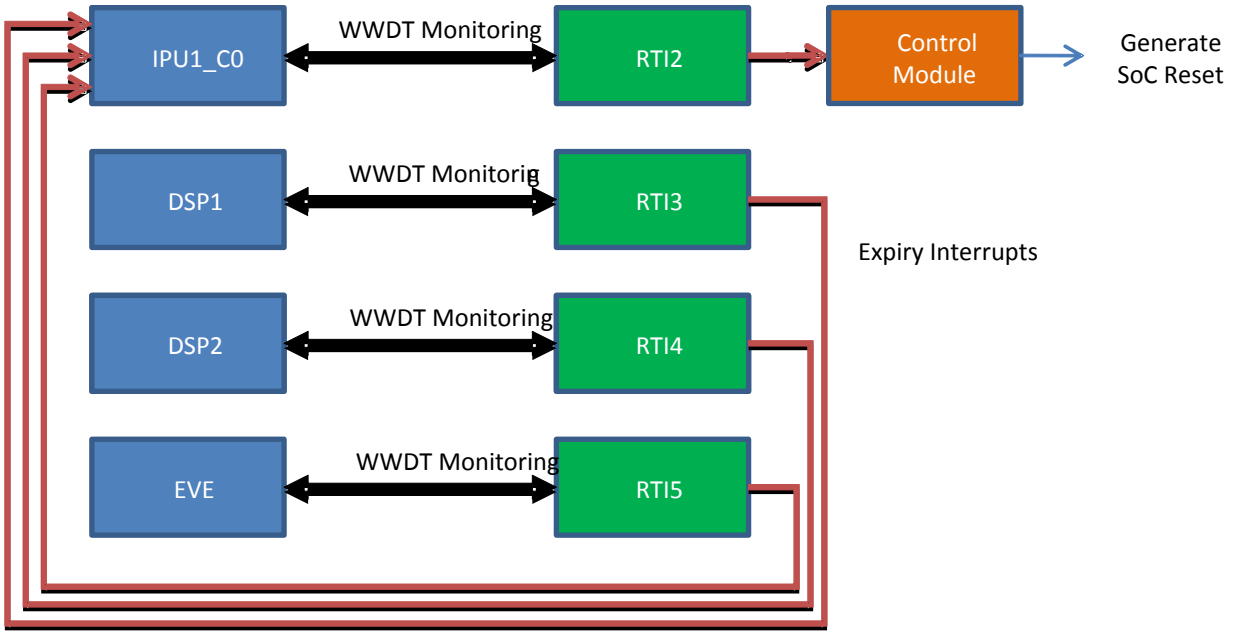

Figure 3. RTI integration in VisionSDK

- RTI task
  - Implemented in the folder vision_sdk/apps/src/rtos/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 does 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, 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/links_fw/src/rtos/links_common/system/system_ipc_msgq.c". For relevant code, search for RTI_INCLUDE.
- Warm reset recovery considerations in case of using RTI
  - If the WWDT corresponding to master core expires, the system is configured to undergo a warmreset.
  - 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 Driver Changes 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 ti_components/drivers/pdk/packages/ti/drv/vps/include/osal/bsp_safety_osal.h

  - This is implemented in ti_components/drivers/pdk/packages/ti/drv/vps/src/osal/tirtos/bsp_safety_osal.c
  - Driver examples do not implement FFI. Therefore, the safety OSAL implementation uses only empty functions.
- USER and SUPERVISOR switch in DSP
  - Relevant code is available in:
    - ti_components\drivers\pdk\packages\ti\csl\arch\c66x\dsp_usrSpvSupport.h
    - ti_components\drivers\pdk\packages\ti\csl\arch\c66x\src\dsp_usrSpvSupport.c
    - ti_components\drivers\pdk\packages\ti\csl\arch\c66x\src\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, use the SWENR instruction and the corresponding handler isused.
    - 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, 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.

8 References

ECC/EDC on TDAxx

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), 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, regulatory 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 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.

TI objects to and rejects any additional or different terms you may have proposed.

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