Application Report
SPRAC42B–July 2016–Revised June 2018

ECC/EDC on TDAxx

Chaitanya Ghone, Rishabh Garg, Piyali Goswami, and Yashwant Dutt *ADAS Software, Processor BU*

ABSTRACT

TDA2x and TDA3x series of automotive processor are designed to be used in automotive safety systems. To enable safety, these processors come with error detection and correction (EDC) support for various memories. This application report provides an overview and usage description of EDC.

Contents

List of Tables

Trademarks

1 Introduction

Table 1. EDC Support on various TI TDAx System-on-Chips (SoCs)

	TDA2X	TDA2EX	TDA2PX	TDA3X
EMIF1	EDC/ECC	EDC/ECC	EDC/ECC	EDC/ECC
EMIF2	NO	NA	NO	NA
OCMC RAM (All instances)	EDC/ECC	EDC/ECC	EDC/ECC	EDC/ECC
A15: L1P Cache	NO	NO	NO	NA
A15: L1D Cache	NO	NO	NO	NA
A15: L2 Cache	NO	EDC/ECC	NO	NA
M4: L1 Unicache	NO	NO	NO	EDC/ECC
M4: L2 RAM	NO	NO	NO	EDC/ECC
DSP: L1P Cache/RAM	PARITY	PARITY	PARITY	PARITY
DSP: L1P Tag	NO	NO	PARITY	NO
DSP: L1D Cache/RAM	NO	NO	EDC/ECC	NO
DSP: L1D Tag	NO	NO	EDC/ECC	NO
DSP: L2 RAM/Cache	EDC/ECC	EDC/ECC	EDC/ECC	EDC/ECC
DSP: L2 Tag	NO	NO	EDC/ECC	NO
EVE: DMEM/WBUF/IBUFxx	PARITY	NA	PARITY	PARITY

- EDC/ECC: Support for 1-bit error correction and 2-bit error detection.
- PARITY: Support for parity checks only
- NO: Memory checks are not available
- NA: Feature is not available in hardware

Due to silicon erratum i882, EMIF ECC is usable only in TDA2x-SR2.0, TDA2Ex-SR2.0, TDA3x-SR2.0 and higher.

Table 2. Acronyms

Acronym	Description
connID	Connection identifier. TDAx SoCs use different connID definitions in different parts of the SoC. In case of L3 Interconnect uses internally a 6-bit ConnID, L3 firewalls uses a 4-bit ConnID, Statistics collectors use a 8-bit ConnID. To identify which definition to use, see the context within the device-specific documentation.
CPU	Refers to the processor rather than the subsystem. For example, CPU in EVE refers to the ARP32 RISC core. EVE subsystem refers to the ARP32 RISC Core, VCOP, internal MMUs, internal EDMAs, an so forth.
ECC	Error correcting code module. Supports single bit correction and double bit detection
KB, MB	1024 bytes and 1024*1024 bytes, respectively
SBL	Secondary boot loader

2 EMIF EDC/ECC

2.1 Overview

EMIF supports ECC on the data written or read from the SDRAM. Enable the ECC feature by writing to the appropriate registers inside the EMIF subsystem. ECC accesses are allowed for both SYS and the MPU ports. 7-bit ECC is calculated over 32-bit data when in 32-bit DDR mode. 6-bit ECC is calculated over 16-bit data when in 16-bit DDR mode. The ECC is calculated for all accesses that are within the address ranges protected by ECC. These address ranges are software configurable. The ECC must be enabled and only aligned writes with byte count in multiple of 4 bytes (2 bytes for narrow mode) should be used. This is true for all TDA devices other than TDA2PX where you can have sub quanta writes as well.

The features of the EMIF ECC are as shown below:

- ECC on SDRAM data bus
- 7-bit ECC over 32-bit quanta or 6-bit ECC over 16-bit quanta in narrow mode
- 1-bit correction and 2-bit detection
- Programmable address ranges to define ECC protected region
- ECC calculated and stored on all writes to ECC protected address region
- ECC verified on all reads from ECC protected address region
- Statistics for 1-bit ECC and 2-bit ECC errors
- All DDR's must have the same data bus width. The ECC DDR must support the same data bus width as the normal DDR IC's for data. The total width of the ECC DDR data bus is 8 bits.

Any access to ECC protected region must be ECC quanta-aligned (32-bit or 16-bit based on narrow mode usage other than TDA2PX where you can have sub quanta writes as well.). For details on how to ensure this, see Section 2.5.

2.2 Programming Model

Step 1. Enable ECC in Control Module.

Table 3. CTRL_WKUP_EMIF1_SDRAM_CONFIG_EXT

Address offset	0x0000 0144		
Physical Address		**Instance**	CTRL_MODULE_WKUP
Description	SLICE register for emif1 and emif2		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED														EMIF1_NARROW_ONLY	EMIF1_EN_ECC	EMIF1_REG_PHY_NUM_OF_SAMPLES		EMIF1_REG_PHY_SEL_LOGIC	EMIF1_REG_PHY_ALL_DQ_MPR_RD_RESP	EMIF1_REG_PHY_OUTPUT_STATUS_SELECT		RESERVED	EMIF1_SDRAM_DISABLE_RESET	EMIF1_PHY_RD_LOCAL_ODT		RESERVED	EMIF1_DFI_CLOCK_PHASE_CTRL	EMIF1_EN_SLICE_2	EMIF1_EN_SLICE_1	EMIF1_EN_SLICE_0	

Step 2. Set Address ranges for ECC protection in EMIF registers.

Table 4. EMIF_ECC_ADDRESS_RANGE_1

Address offset	0x4C00 0114		
Physical Address		**Instance**	EMIF1
Description			
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REG_ECC_END_ADDR_1																REG_ECC_STRT_ADDR_1															

Bits	Field Name	Description	Type	Reset
31:16	REG_ECC_END_ADDR_1	End address[31:16] for ECC address range 1. If this bit field is set to 0x1000, this indicates that the SDRAM physical end address on which the ECC applies is 0x1000 FFFF. If this bit field is set to 0x0FFF the physical end address on which the ECC applies is 0x0FFF FFFF. This bit field controls only the 16 MSbs of the physical end address of the ECC protected range. The other 16 LSbs are always 0xFFFF.	RW	0x0
15:0	REG_ECC_STRT_ADDR_1	Start address[31:16] for ECC address range 1. If this bit field is set to 0x0000, this indicates that the SDRAM physical start address on which the ECC applies is 0x0000 0000. This bit field controls only the 16 MSbs of the physical start address of the ECC protected range. The other 16 LSbs are always 0x0000.	RW	0x0

Table 5. EMIF_ECC_ADDRESS_RANGE_2

Address offset	0x4C00 0118		
Physical Address		**Instance**	EMIF1
Description			
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REG_ECC_END_ADDR_2																REG_ECC_STRT_ADDR_2															

Bits	Field Name	Description	Type	Reset
31:16	REG_ECC_END_ADDR_2	End address[31:16] for ECC address range 2. If this bit field is set to 0x1000, this indicates that the SDRAM physical end address on which the ECC applies is 0x1000 FFFF. If this bit field is set to 0x0FFF the physical end address on which the ECC applies is 0x0FFF FFFF. This bit field controls only the 16 MSbs of the physical end address of the ECC protected range. The other 16 LSbs are always 0xFFFF.	RW	0x0
15:0	REG_ECC_STRT_ADDR_2	Start address[31:16] for ECC address range 2. If this bit field is set to 0x0000, this indicates that the SDRAM physical start address on which the ECC applies is 0x0000 0000. This bit field controls only the 16 MSbs of the physical start address of the ECC protected range. The other 16 LSbs are always 0x0000.	RW	0x0

Table 6. EMIF_ECC_CTRL_REG

Address offset	0x4C00 0110		
Physical Address		**Instance**	EMIF1
Description			
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REG_ECC_EN	REG_ECC_ADDR_RGN_PROT	RESERVED																												REG_ECC_ADDR_RGN_2_EN	REG_ECC_ADDR_RGN_1_EN

Bits	Field Name	Description	Type	Reset
31	REG_ECC_EN	Set 1 to enable ECC. Set 0 to disable ECC.	RW	0x0
30	REG_ECC_ADDR_RGN_PROT	Setting this field to 1 and reg_ecc_en to a 1 will enable ECC calculation for accesses within the address ranges and disable ECC calculation for accesses outside the address ranges. Setting this field to 0 and reg_ecc_en to a 1 will disable ECC calculation for accesses within the address ranges and enable ECC calculation for accesses outside the address ranges. The address ranges can be specified using the ECC Address Range 1 and 2 registers.	RW	0x0
29:2	RESERVED		R	0x0
1	REG_ECC_ADDR_RGN_2_EN	Set 1 to enable ECC address range 2. Set 0 to disable ECC address range 2.	RW	0x0
0	REG_ECC_ADDR_RGN_1_EN	Set 1 to enable ECC address range 1. Set 0 to disable ECC address range 1.	RW	0x0

- The EMIF_ECC_ADDRESS_RANGE_x registers must be configured before enabling ECC using the REG_ECC_EN bit in the EMIF_ECC_CTRL_REG register.

CAUTION

The value configured in this register is physical address of the memory. Thus, to define the address range of 0x8000_0000 to 0xAFFF_FFFF, the register should be configured to value 0x2FFF_0000.

- You can choose to use only one range, if needed, by using REG_ECC_ADD_EGB_x_EN bits in the EMIF_ECC_CTRL_REG register.
- By using both the address range registers and setting REG_ECC_ADD_RGN_PROT bit to 0, up to three non-contiguous regions can be defined over which ECC protection can be enabled.

CAUTION

EMIF_ECC_ADDRESS_RANGE_x must not overlap. Hardware behavior is undefined under such cases.

- As explained in the EMIF_ECC_ADDRESS_RANGE_x registers description, the start address and end addresses are provided using only the top 16 MSBits and the other 16 LSBits are assumed to be zero. For example, in order to protect address range from 0x8000_0000 to 0x8020_FFFF, the start address value will be 0x0000 and end address will be 0x0020. Hence, the register must be configured to value 0x0020_0000. One corner case here is to protect address range 0x8000_0000 to 0x8000_FFFF where the register must be configured to value 0x0000_0000.

CAUTION

In case of DDR3, hardware leveling must occur even for ECC memories. To ensure this, the EMIF_ECC_ADDRESS_RANGE_1 to 0x0 and set EMIF_ECC_CTRL_REG to 0xC000_0001 to use only one range before triggering hardware leveling. Once the leveling is complete, disable ECC by setting EMIF_ECC_CTRL_REG to 0. ECC can be enabled at any later point by software for appropriate memory ranges.

- It should be noted that EMIF_ECC_CTRL_REG register has an additional field REG_RMW_EN on TDA2PX device as shown in Table 7. This field should be set to 1 in order to enable read-modify-write functionality. This will disable the ECC error generation on sub quanta writes.

Table 7. EMIF_ECC_CTRL_REG

Address offset	0x4C00 0110		
Physical Address		**Instance**	EMIF1
Description			
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REG_ECC_EN	REG_ECC_ADDR_RGN_PROT	REG_ECC_VERIFY_DIS	REG_RMW_EN	RESERVED																										REG_ECC_ADDR_RGN_2_EN	REG_ECC_ADDR_RGN_1_EN

Bits	Field Name	Description	Type	Reset
28	REG_RMW_EN	[New TDA2Px feature versus TDA2x][New DRA7xxP feature versus DRA75x/DRA74x] When REG_ECC_EN = 0x1: 0x0: Disable RMW functionality for sub-quanta accesses 0x1: Enable RMW functionality for sub-quanta accesses The value of this bit is ignored when REG_ECC_EN = 0x0.	RW	0x0

Step 3. (Optional) Set the threshold for 1-bit error interrupt in EMIF registers.

Table 8. EMIF_1B_ECC_ERR_THRSH

Address offset	0x134		
Physical Address		**Instance**	EMIF1
Description			
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REG_1B_ECC_ERR_THRSH								RESERVED								REG_1B_ECC_ERR_WIN															

Bits	Field Name	Description	Type	Reset
31:24	REG_1B_ECC_ERR_THRSH	1-bit ECC error threshold. The EMIF will generate an interrupt when the 1-bit ECC error count is greater than or equal to this threshold. A value of 0 will disable the generation of the interrupt.	RW	0x0
23:16	RESERVED		R	0x0
15:0	REG_1B_ECC_ERR_WIN	1-bit ECC error window in number of refresh periods. The EMIF will generate an interrupt when the 1-bit ECC error count is equal to or greater than the threshold within this window. A value of 0 will disable the window. Refresh period is defined by reg_refresh_rate in SDRAM Refresh Control register. The software can set this bitfield to 0x0 to reset the internal counter.	RW	0x0

- Single bit errors are corrected on-the-fly by ECC logic. Software does not need to handle occasional 1-bit errors. However, a large number of 1-bit errors would point to a potential problem in the environment or in the memory module. Application and diagnostic software can set up this threshold register, to interrupt only when 1-bit errors exceed the specified count and handle it an appropriate manner.

Step 4: (Optional) Clear the error status

Table 9. EMIF_1B_ECC_ERR_ADDR_LOG

Address offset	0x13C		
Physical Address		**Instance**	EMIF1
Description			
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REG_1B_ECC_ERR_ADDR																															

Bits	Field Name	Description	Type	Reset
31:0	REG_1B_ECC_ERR_ADDR	1-bit ECC error address. Most significant bits of the starting address(es) related to the SDRAM reads that had a 1-bit ECC error. This field displays up to four addresses logged in the 4 deep address logging FIFO. Writing a 0x1 will pop one element of the FIFO. Writing a 0x2 will pop all elements of the FIFO. Writing any other alue will have no effect.	RW	0x0

Table 10. EMIF_2B_ECC_ERR_ADDR_LOG

Address offset	0x140		
Physical Address		**Instance**	EMIF1
Description			
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REG_2B_ECC_ERR_ADDR																															

Bits	Field Name	Description	Type	Reset
31:0	REG_2B_ECC_ERR_ADDR	2-bit ECC error address. Most significant bits of the starting address of the first SDRAM burst that had the 2-bit ECC error. Writing a 1 will clear this field. Writing any other value has no effect.	RW	0x0

- This is an optional step. Before enabling or using ECC, it is a good programming practice to clear any stale ECC error status. Below registers should be cleared by writing 0x2 to EMIF_1B_ECC_ERR_ADDR_LOG and 0x1 to EMIF_2B_ECC_ERR_ADDR_LOG.
- Such stale errors might exist due to earlier execution of diagnostic software and are more common in development phase when ECC might not be setup or used correctly in software.
- This history is not maintained across SoC resets.

Step 5: Enable the interrupts from EMIF and WR_ECC_ERR_SYS. It can be generated for other errors in the EMIF subsystem as well.

Table 11. EMIF_SYSTEM_OCP_INTERRUPT_ENABLE_SET

Address Offset	0x0000 00B4		
Physical Address		**Instance**	EMIF1 EMIF2
Description	System OCP Interrupt Enable Set Register		
Type	RW		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																										ONEBIT_ECC_ERR_SYS	TWOBIT_ECC_ERR_SYS	WR_ECC_ERR_SYS	RESERVED		EN_ERR_SYS

Bits	Field Name	Description	Type	Reset
31:6	RESERVED		R	0x000 0000
5	ONEBIT_ECC_ERR_SYS	Enabled status of sysem ECC one bit error correction interrupt. Writing a 1 will enable the interrupt, and set this bit as well as the corresponding Interrupt Enable Clear Register. Writing a 0 has no effect.	RW W1toSet	0x0
4	TWOBIT_ECC_ERR_SYS	Enabled status of system ECC two bit error detection interrupt. Writing a 1 will enable the interrupt, and set this bit as well as the corresponding Interrupt Enable Clear Register. Writing a 0 has no effect.	RW W1toSet	0x0
3	WR_ECC_ERR_SYS	Enabled status of system ECC Error interrupt when a memory access is made to a non-quanta aligned location. Writing a 1 will enable the interrupt, and set this bit as well as the corresponding Interrupt Enable Clear Register. Writing a 0 has no effect.	RW W1toSet	0x0
2:1	RESERVED		R	0x0
0	EN_ERR_SYS	Enable set for system OCP interrupt for command or address error. Writing a 1 will enable the interrupt, and set this bit as well as the corresponding Interrupt Enable Clear Register. Writing a 0 has no effect.	RW W1toSet	0

- This generates the EMIF1_IRQ interrupt that can be monitored on any CPU by using the IRQ_CROSSBAR_105 instance in the SoC. The following is sample code for monitoring this on A15 using SYS/BIOS.

```
var IntXbar = xdc.useModule('ti.sysbios.family.shared.vayu.IntXbar');
var Hwi     = xdc.useModule('ti.sysbios.family.arm.gic.Hwi');

/* IRQ_CROSSBAR_105 EMIF1_IRQ. Interrupt 57 on A15 */
IntXbar.connectIRQMeta(57, 105);

/* Set up interrupt 57 on A15 */
var hwi_param_0 = new Hwi.Params;
hwi_param_0.arg = 0;
Hwi.create(57+32, '&emifErrIrqIsr', hwi_param_0);
```

Step 6: Initialize ECC enabled memory regions.

- DDR initialization is needed so that the ECC checksum memory gets initialized properly. Given that the DDR and ECC parity bits are uninitialized when the SoC is first powered on, it is important for the software to first initialize the whole memory after enabling the ECC.
  - This step would ensure that the parity bits are correctly set before the code access to ECC enabled regions.
  - This step would ensure that ECC error scenarios are not generated for uninitialized parity data.
  - This step can be done by either the CPU memset or the EDMA transfer to the memory region.
  - The initialization requirement is multiple of 2 bytes for narrow_mode =1, 4 bytes for narrow_mode = 0.
- When using an EDMA to initialize the memory, care should be taken to ensure the start address and end address is 32- or 16-bit aligned (based on normal or narrow mode) when initializing.

CAUTION

If CPU memset is used, cache must be disabled to prevent reads generated by the cache controller to ECC protected memory with uninitialized parity. In general, EDMA based memset is recommended for optimal performance.

Step 7: ECC programming is now complete, resume normal operation of the code.

2.3 Error Reporting

Additional steps related to the following functionalities, not listed above, available in SoC, present in software, and provided by TI are:

- Disable ECC
- Clearing interrupt status
- Disabling interrupts
- Getting RAW status of error
- Getting ECC error info

For more information, see the device-specific TRM.

NOTE: When cache policy of Write Back (WB)/ Write Allocate (WA) is enabled on the CPU, care should be taken to not have the cache lines being read from uninitialized ECC memory. Typically when the CPU cache is enabled in WB-WA mode and the CPU is trying to initialize the EMIF ECC enabled memory, the cache line would be first read into the cache leading to the ECC controller to start reporting ECC errors. Additionally, the initialization would reside in cache unless explicitly flushed to memory. The way to avoid errors from the ECC controller when using cache is to always initialize the ECC enabled memory before performing any read or write to cached ECC enabled memory region.

2.3.1 Identifying the Source of Non-Quanta-Aligned Writes

- Software should register for EMIF1_IRQ interrupt using the IRQ_CROSSBAR_105 instance of the SoC interrupt crossbar.
- Verify that interrupt is generated due to non-quanta-aligned write by checking that WR_ECC_ERR_SYS bit is set in EMIF_SYSTEM_OCP_INTERRUPT_RAW_STATUS (offset: 0xA4).
- Check the EMIF_OCP_ERROR_LOG (offset: 0xD0). Bits [7:0] indicate the ConnID of the master who generated this write. The 8-bit connID mapping is provided in the *Statistics Collector Master Address Mapping* table in the device-specific TRM.
- Once the source is identified, an OCP watch-point or MA watch-point can be used to trap the non-quanta-aligned access.

Alternate option: Route the EMIF interrupt to the same CPU that causes the non-quanta-aligned access. By looking at the call-stack, the section of code causing the error might be able to be localized. Looking at the disassembler output, you may be able to identify the source of this write. This method is not as precise because the actual write happens later due to presence of cache.

2.4 EMIF ECC Support in Driver Package

PDK TDA Driver package provides reference software implementation for using EMIF ECC. This package is a part of Vision SDK software package available on ti.com.

- Interface: pdk/packages/ti/csl/csl_emif.h
- Implementation: pdk/packages/ti/csl/src/ip/emif/V0/priv/emif.c
- Sample application: pdk/packages/ti/csl/example/ecc/ecc_test_app

Table 12. Programming Model for Using ECC

Programming Step	Starterware Function
EMIF ECC example	main.c: emifEccTest()
STEP 1/2: Enable EMIF ECC in control module and EMIF registers	This is done in GEL files or SBL when configuring the EMIF registers.
STEP 3: Set the address range for ECC operation and enabling the ECC	emif.c: EMIFConfigECCInitECCParams() emif.c: EMIFConfigECCEnableEcc()
STEP 4: (Optional) Set the threshold for 1-bit error interrupt	emif.c: EMIFConfigECCInitECCParams()
STEP 5: (Optional) Clear the error status	emif.c: EMIFConfigECCClrAllEccErrInfo()
STEP 6: Enable the interrupts	emif.c: EMIFEnableIntr()
STEP 7: Initialize ECC enabled memory regions	main.c: emifEccConfig()

NOTE: Some older versions of Vision SDK used ADAS Starterware™ Driver package for device drivers. Kindly refer accordingly as file names/paths, API names might differ slightly between PDK and Starterware.

2.5 Careabouts of Using EMIF ECC

There are few restrictions that need to be taken into account while using EMIF ECC. This is due to non-availability of read modify write support and silicon errata i882 related to ECC.

2.5.1 Accesses From CPUs and Periperals

2.5.1.1 Restrictions Due to Non-Availability of Read Modify Write ECC Support in EMIF

In normal mode, a 7-bit ECC is computed for each 32-bit word, whereas, in narrow mode, a 6-bit ECC is computed for each 16-bit word. EMIF does not provide read-modify-write support for ECC for all TDA devices except TDA2PX. This means that any sub-quanta write access (less than 32 bit for normal and less than 16 bit for narrow mode) generates an incorrect ECC and writes it into ECC memories. Any further reads to such corrupted regions will very likely generate false 1-bit or 2-bit errors. To ensure this does not happen, software design should take care of the following points.

2.5.1.2 Non-Cached CPU Accesses to EMIF

Any non-cached CPU access to EMIF can result in sub-quanta EMIF writes if the datatype being chosen does not match the ECC quanta size. Table 13 and Table 14 summarizes these constraints.

Table 13. ECC Correctness for 32-Bit EMIF for Non-Cached CPU Data Writes

	C datatype			
memory/core	**byte**	**halfword**	**word**	**Double**
A15	Not OK	Not OK	OK	OK
DSP	Not OK	Not OK	OK	OK
EVE	Not OK	Not OK	OK	OK
M4	Not OK	Not OK	OK	OK

Table 14. ECC Correctness for 16-Bit EMIF for Non-Cached CPU Data Writes

	C datatype			
memory/core	byte	halfword	word	Double
A15	Not OK	OK	OK	OK
DSP	Not OK	OK	OK	OK
EVE	Not OK	OK	OK	OK
M4	Not OK	OK	OK	OK

Apart from the information discussed above, even unaligned writes should be avoided. Unaligned writes can be produced by the compiler even if datatypes corresponding to word or double is used. For details regarding compiler optimization, see Section 2.5.2.

2.5.1.3 Cached CPU Accesses to EMIF

For cached CPU access to EMIF, the accesses are typically cache line aligned. Hence, the restriction of RMW is not applicable. However, the cache used must be “write-back, write-allocate”. Non “write-allocate” caches might generate non-quanta aligned write accesses. Besides this, there are few more points that need to be considered for the following CPUs.

- M4

  Cortex M4 has a write allocate cache. Hence one can set cache policy to "Write back, Write Allocate" in order to prevent sub quanta accesses to the memory. However this is not true during cache maintenance operations:

  – In IPU system, an access can result in a cache miss while maintenance operation is going on. This can happen with either single core or both cores running. In such cases:
    - Cache Read Miss will not Allocate
    - Cache Write Miss will be write-through No-Allocate
  – Hence, unaligned writes go through to DDR memory when a cache maintenance operation is ongoing and ECC error interrupts are generated.
  – This does not happen for normal cache eviction.
  – Cache maintenance operations are: Write back, pre fetch, invalidate, lock and unlock

  Below software workarounds can be used to mitigate this:

  – This workaround is applicable to systems where only single core of dual core IPU subsystem is used and other core is idle/WFE. Modify the cache wrapper APIs to follow the below sequence:
    - Disable_task_scheduling
    - Disable_interrupts
    - Do cache maintenance
    - Restore_interrupts
    - Restore_task_scheduling

  - This workaround is applicable to systems where both cores of dual core IPU subsystem are used. The idea is to stop the other core from doing anything when the first core calls cache maintenance API. Modify the cache wrapper APIs to follow the below sequence:
    - Disable_task_scheduling
    - Take_spinlock
    - Disable_interrupts
    - Stop_other_core
    - Do cache maintenance
    - Start_other_core
    - Restore_interrupts
    - Release_spinlock
    - Restore_task_scheduling

  Note that one CPU will have to be idle when the other is doing cache maintenance.

- A15

  Cortex A15 (ARMV7-A) has a complex cache architecture to support optimal performance along with cache coherency.

  If the cache is enabled (SCTLR.C set) and the memory page is configured for “write-back, write-allocate”, accesses to EMIF are compatible with 32-bit (or 16-bit) ECC quanta. Further, write streaming (ACTLR bit 24) enhancement, which bypasses the write-allocate and ensures 32-bit access that is compatible with 32-bit (or 16-bit) ECC quanta.

  From a software point of view, EMIF memory should be configured as “Normal” memory (as defined by ARM documentation) with memory attributes that ensure “write-back, write-allocate” behavior, This should ensure that there are no sub-quanta accesses generated when using cache.

- EVE

  Eve does not have data cache, it only has program cache. So the data access of EVE to the EMIF depends on the C datatype being used. If the C code does a byte or a half word access, this violates the alignment constraint and ECC can be corrupted.

- DSP

  When DSP comes out of reset, only L1D and L1P caches are enabled. L1D cache is not “write-allocate”. However, L2 cache does support “write-allocate” feature. Thus, software should ensure that initial boot-code on DSP uses L2 for stack. This is the default “.stack” section allocated by the linker when using SYS/BIOS. SYS/BIOS allows tasks to use different stacks, some of which might exist in DDR. Initial software should, therefore, enable L2 cache before SYS/BIOS resumes to ensure quanta-aligned DDR accesses. For a reference SYS/BIOS based implementation to support both EMIF ECC and DSP EDC, see Section 5.4.

2.5.1.4 Non CPU Access of EMIF Memory

- Non-CPU initiators like DMA or peripheral like VIP, USB, and so forth can generate non-quanta aligned accesses to EMIF and generate false errors. To avoid this, the following must be ensured:
  - Start the address of any write operation must be 32 bit (or 16 bit for narrow mode) aligned
  - In case of 1D accesses, length of each write access must be a multiple of 32 bit (or 16 bit for narrow mode)
  - In case of 2D accesses, the “pitch” or “stride” should be a multiple of 32 bit (or 16 bit for narrow mode)
- There are multiple masters in TDA2x SoC that can generate write-accesses to EMIF. A non-comprehensive list is given below:
  - EDMA (DSP, EVE, SYSTEM)
  - SDMA
  - GPU/BB2D
  - MMC
  - USB

  - GMAC
  - VIP
  - VPE
  - DSS
- For several of the above IPs, it might not be possible to program the descriptor to always produce quanta-aligned writes. This is because the DMA size is determined by the incoming packet length or pixel location in frame that is not in your control.

2.5.1.5 Controlling Quanta-Alignment on Different Peripherals

This section discusses how to limit non-quanta aligned for some of the peripherals.

- VIP/VPE

  VIP/VPE use VPDMA to access memories. VPDMA is configured using descriptors stored in memory.
  - Frame-Start address must be 32-bit aligned
  - Stride value must be multiple of 128 bit (IP requirement is 128 bit, ECC requirement is 32 bit)
  - Software drivers must make sure that any updates to write the VPDMA descriptor uses a 32-bit aligned access using “volatile” keyword as explained in 2.5.2
- DSS

  DSS primarily does read accesses, but it does make write-access when using “WRITEBACK” pipeline.
  - Frame-start for write-back must be 32-bit aligned.
  - Stride/pitch of the frame must be a multiple of 32 bit.
  - Values for DISPC_WB_PIXEL_INC not equal to 1 can generate ECC errors.
  - Frame width in bytes should be a multiple of 32 bits.
    - A width of 300 pixels in RGB24 format has a width of 900 bytes, which is not a multiple of 32 bits. This generates an error at the end of every line.
- EDMA
  - Start address of any transfer must be 32-bit aligned.
  - Address jump after ACNT or BCNT must be 32-bit aligned.
  - ACNT must be a multiple of 32 bit.
- SDMA
  - Start address must be 32-bit aligned.
  - Size of each transfer must be a multiple of 32 bits.
  - Data transfer element size must be 32 bit.
- GPU
  - 32-bit pixel formats like ARGB32 will ensure quanta-aligned accesses.
    - Other pixels format may generate non-quanta aligned write accesses and cause errors.

2.5.1.6 Debugger Access of EMIF via the Memory Browser/Watch Window

Debugger accesses to EMIF is mostly read. In case any write is done via debugger watch window or memory browser, care should be taken similar to what is described in Section 2.5.1.2 when cache is not enabled.

2.5.1.7 Software Breakpoints While Debugging

- Software breakpoints are typically implemented by modifying the instruction and replacing the opcode with a breakpoint opcode. As the opcode for “breakpoint” can be less than 32 bit in many cases, it becomes a quanta-unaligned write and causes ECC errors. Table 15 summarizes the CPU and the impact for Software breakpoints.

Table 15. Software Breakpoint With ECC

CPU	Software Breakpoint
A15	OK (not OK if using thumb mode)
DSP	Not OK
EVE	Not OK
M4	Not OK

- To overcome the above situations, the following approach is recommended:
  - Limit breakpoints to only hardware breakpoints
  - If software breakpoints are needed, ECC checks can be disabled without any changes in software using direct debugger writes to EMIF registers.
  - Alternately, memory section for code can be kept in the non-ECC protected region.
  - Since EMIF ECC supports two ranges for ECC, one section can be dedicated to “code” sections and other to “data”. The “code” section can be disabled when breakpoints are to be used by disabling the range in EMIF_ECC_CTRL_REG.

2.5.2 Compiler Optimizations

- Compiler can optimize a C code to produce unaligned access even if the native datatype used in of word size.

```
void fxn(int *a, int *b)
{
    *b = (0xFFFF00FF & *b) | (*a &0x0000FF00);
}

PRODUCES BELOW ASSEMBLY CODE

ldrb r3, [r0, #1]
strb r3, [r1, #1]
```

- The following steps are recommended to ensure no-sub-quanta accesses:
  - Screen the generated object code to check for any such byte/halfword store instruction:
    - The final executable (<filename>.out) can be disassembled using the disassembler available in all compiler distributions.
    - Search for STRB or STRH instructions
  - Rewrite the C code to eliminate the usage of byte/halfword store instruction
  - Use volatile keyword to direct the compiler to use only 32-bit aligned accesses to fix observed errors.

2.5.3 Silicon Errata Related to ECC

- i882
  - ECC generates non-quanta-aligned write error even on non-ECC protected regions. It also results in “abort” or “hard-fault” on the CPU generating the access.
  - As a result of this erratum, ECC cannot be used on TDA2x-SR1.1, TDA2Ex-SR1.0, TDA3x-SR1.0.
  - This erratum has been fixed in TDA2x-SR2.0, TDA2Ex-SR2.0, TDA3x-SR2.0 and higher. After the fix, EMIF will not return an error-response over the bus for non-aligned writes. Thus, there will not be any “aborts” or “hard-faults” in case of any non-quanta-aligned writes. However, EMIF correctly generates the WR_ECC_ERR_SYS interrupt for the non-quanta-aligned writes to ECC protected regions.

2.5.4 ECC With 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.

However, ECC is supported only on EMIF1 interface. This implies ECC protection will not apply to entire address range when interleaving is active. At system integration, this can be handled in 3 possible scenarios which we will discuss in the further sections.

2.5.4.1 Scenario 1: 1GB on EMIF, 512 MB on EMIF2, ECC on Non-Interleaved 512MB

- 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
  - LISA_MAP_1 = 0xC050_0120
    - This will map the address range 0xC000_0000 to 0xDFFF_FFFF to EMIF1 only in an non-interleaved fashion
  - All other LISA_MAP_x are assumed to not override the LISA_MAP_0/1 configuration.
- Assume that you want to protect a single address range 0xC000_000 to 0xC1FF_FFFF. This section is not part of the interleaved memory section. In this case, the EMIF_ECC_ADDRESS_RANGE_1 register value can be set to 0x21FF_2000. This is calculated as follows:
  - Start address = 512MB (physical address in the memory connected to EMIF) = 0x2000_0000
  - End address = Start address + 0x01FF_FFFF = 0x21FF_FFFF
- In this scenario, you can choose to protect the entire 512 MB address range from 0xC000_0000 to 0xDFFF_FFFF.

NOTE: Start and end addresses are less than 512MB (0x2000_0000) corresponds to the scenario described in Section 2.5.4.2.

2.5.4.2 Scenario 2: 512MB on EMIF1, 1GB on EMIF2, ECC on Interleaved 512MB

CAUTION

Only alternate sets of 128 bytes will have ECC protection in this scenario. For example, data at 0x8000_0000 to 0x8000_007F will be protected by ECC because it goes to the EMIF1 interface, but data at 0x8000_0080 to 0x8000_00FF will not be protected since it goes to the EMIF2 interface. decide if this is an acceptable level of protection based on your use-case.

- 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.
  - LISA_MAP_1 = 0xC050_0220
    - This will map the address range 0xC000_0000 to 0xDFFF_FFFF to EMIF2 only in an non-interleaved fashion
  - All other LISA_MAP_x are assumed to not override the LISA_MAP_0/1 configuration.
- Since data in address range 0x8000_0000 to 0xBFFF_FFFF is shared between EMIF1 and EMIF2 – all data in this range cannot be protected by ECC. Assume that only a single address range 0x9000_0000 to 0xAFFF_FFFF needs to be protected. In this case, the EMIF_ECC_ADDRESS_RANGE_1 register value can be set to 0x17FF_0800. This is calculated as follows:
  - Start address = (0x9000_000 – 0x8000_0000)/2 = 0x0800_0000
  - End address = (0xAFFF_FFFF – 0x8000_0000)/2 = 0x17FF_FFFF
  - The division by 2 is needed since the physical address within the memory modules used will be half due to interleaving. For the address range, the actual physical address range used in memories connected to EMIF1 will be 0x0800_0000 to 0x17FF_FFFF.

NOTE: Start and end address cannot be more than 0x1FFF_FFFF since only 512MB memory is connected to EMIF1.

3 OCMC ECC

3.1 Overview

TDA2x/TDA2Ex/TDA3x SoCs support Error Checking and Correction (ECC) on the On-Chip Memory (OCMC RAM) subsystem. This is available for all instances of OCMC RAM in the SoC.

Overview of the OCMC ECC features is as below:

- Error correction and detection - Single Error Correction and Dual Error Detection (SECDED)
  - 9-bit Hamming ECC calculated using 128-bit data word concatenated with 18 memory address bits [21:4]
  - Hamming distance of 4 (single error detection/correction and double error detection; triple error detection (TED) NOT supported).
  - ECC parity is ensured to be valid for all write transactions to enabled region
    - 128-bit aligned writes have no additional overhead for parity generation
    - Non-128-bit aligned writes will maintain valid parity by doing a read-modify-write operation in hardware. This results in some performance overhead.
  - DED
    - Address where double-bit errors and count of double-bit errors are logged in ECC registers
    - Data is returned without any correction
    - An optional error response (“data-abort”) might be sent instead to the initiator in case of DED.

  - SEC
    - Address corresponding to the SEC will be logged in registers for all SEC errors. Statistics counter will be incremented appropriately based type of the errors
    - Single-bit errors detected in data section of the code are corrected and corrected data is returned to the requestor immediately.
    - If the single-bit error is detected, in the address or in ECC code, the data read from memory is returned to the requestor as is and the SEC error is logged in appropriate registers.
    - Hardware Automated write back of correctable detected error: For non-128-bit aligned writes, “read-modify-write” accesses are generated. Any single-bit errors detected in the “read” phase will be can be optionally corrected and written back to the memory.
  - Enable/Disable/Test-Suspend Mode Control through control register
- ECC Error Status Reporting features:
  - Corrected Error Address Trace History Buffer (FIFO) - Depth of 4
  - Non-correctable error address trace history buffer (including DED) – Depth of 4
  - Interrupt Generation for correctable/uncorrectable detected errors
- ECC Diagnostics Configuration
  - SEC/DED/Addr Error Event Counters
  - Programmable SEC/DED/Addr Error Event Counter Exception Threshold registers
  - Corrected single error bit distribution history
  - Register control for enable and disabling of diagnostics
  - Configuration registers and ECC status accessible through OCP MMR interface (L4)
  - Exclude repeated addresses from correctable error address trace history

3.2 Programming Model

1. Enable OCMC RAM ECC.

Figure 1. CFG_OCMC_ECC_MEM_BLK

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												CFG_ECC_ENABLED_128K_BLK																			
R												R/W																			

Bits	Field Name	Description	Type	Reset
31:20	RESERVED		R	0x0
19:0	CFG_ECC_ENABLED_128K_BLK	ECC memory block enable bits. The active level of each bit is 0x1. Bit [0] -> Address offset range 0x0 to 0x1FFFF Bit [1] -> Address offset range 0x20000 to 0x3FFFF ... Bit [19] -> Address offset range 0x260000 to 0x27FFFF	RW	0x0

- Setting CFG_OCMC_MODE
  - ECC can be enabled for entire OCMC RAM by setting CFG_OCMC_MODE to 0x2.
  - ECC can be enabled only for specified 128kB blocks by setting CFG_OCMC_MODE to 0x3 and setting appropriate bits on CFG_ECC_ENABLED_128K_BLK.
  - Setting CFG_ECC_SEC_AUTO_CORRECT to 1, will write corrected data back to memory when performing a “read-modify-write” operation which is triggered by sub-128-bit aligned writes.
  - To disable ECC, CFG_OCMC_MODE must be set to 0x0.
  - Diagnostic software can set CFG_OCMC_MODE to 0x1 to access ECC parity codes.

2. Initialize ECC parity bits.
   a. This step would ensure that the parity bits are correctly set before the any access to ECC protected regions and prevent false errors due to the uninitialized parity data.
   b. This step can be done by either the CPU memset or the EDMA transfer to the memory region.

> **CAUTION**
>
> If CPU memset is used, cache must be disabled to prevent reads generated by the cache controller to ECC protected memory with uninitialized parity. In general, EDMA based memset is recommended for optimal performance.

3. Clear error history.

Figure 2. CFG_OCMC_ECC_CLEAR_HIST

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R															
15	14	13	12	11	10	9	4	3		3		2		1	
Reserved								CLEAR_SEC_BIT_DISTR		CLEAR_ADDR_ERR_CNT		CLEAR_DED_ERR_CNT		CLEAR_SEC_ERR_CNT	
R								R/W		R/W		R/W		R/W	

Bits	Field Name	Description	Type	Reset
31:4	RESERVED		R	0x0
3	CLEAR_SEC_BIT_DISTR	Clear stored single error correction (SEC) bit distribution history. Returns 0 on read. 0x0: Reserved (not used) 0x1: Cleares the following registers: • STATUS_SEC_ERROR_DISTR_0 • STATUS_SEC_ERROR_DISTR_1 • STATUS_SEC_ERROR_DISTR_2 • STATUS_SEC_ERROR_DISTR_3 • STATUS_SEC_ERROR_DISTR_4	RW	0x0
2	CLEAR_ADDR_ERR_CNT	Clear stored address error history. Returns 0 on read. 0x0: Reserved (not used) 0x1: Clears the STATUS_ERR[23:20] ADDR_ERROR_CNT bit field and the ADDRERR FIFO	RW	0x0
1	CLEAR_DED_ERR_CNT	Clear stored double error detection (DED) history. Returns 0 on read. 0x0: Reserved (not used) 0x1: Clears the STATUS_ERR[19:16] DED_ERROR_CNT bit field and the DED FIFO	RW	0x0
0	CLEAR_SEC_ERR_CNT	Clear stored single error correction history. Returns 0 on read. 0x0: Reserved (not used) 0x1: Clears the STATUS_ERROR_CNT[15:0] SEC_ERROR_CNT bit field and the SEC FIFO	RW	0x0

- Such stale errors might exist due to earlier execution of diagnostic software and are more common in development phase when ECC might not be setup/used correctly in software.
- This history is not maintained across SoC resets.

4. Enable interrupts.

Figure 3. Missing Title

31	30	29	28	27	26	25	24
Reserved							
R							
23	22	21	20	19	18	17	16
Reserved							
R							
15	14	13	12	11	10	9	8
Reserved	CBUF_SHORT_FRAME_DETECT_FOUND	CBUF_UNDERFLOW_ERR_FOUND	CBUF_OVERFLOW_WRAP_ERR_FOUND	CBUF_OVERFLOW_MID_ERR_FOUND	CBUF_READ_SEQUENCE_ERR_FOUND	CBUF_VBUF_READ_START_ERR_FOUND	CBUF_READ_OUT_OF_RANGE_ERR_FOUND
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
7	6	5	4	3	2	1	0
CBUF_WRITE_SEQUENCE_ERR_FOUND	CBUF_VBUF_WRITE_START_ERR_FOUND	CBUF_WR_OUT_OF_RANGE_ERR_FOUND	CBUF_VIRTUAL_ADDR_ERR_FOUND	OUT_OF_RANGE_ERR_FOUND	ADDR_ERR_FOUND	DED_ERR_FOUND	SEC_ERR_FOUND
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bits	Field Name	Description	Type	Reset
31:15	RESERVED		R	0x0
14	CBUF_SHORT_FRAME_DETECT_FOUND	CBUF detected short frame	RW	0x0
13	CBUF_UNDERFLOW_ERR_FOUND		RW	0x0
12	CBUF_OVERFLOW_WRAP_ERR_FOUND		RW	0x0
11	CBUF_OVERFLOW_MID_ERR_FOUND		RW	0x0
10	CBUF_READ_SEQUENCE_ERR_FOUND		RW	0x0
9	CBUF_VBUF_READ_START_ERR_FOUND		RW	0x0
8	CBUF_READ_OUT_OF_RANGE_ERR_FOUND		RW	0x0
7	CBUF_WRITE_SEQUENCE_ERR_FOUND		RW	0x0
6	CBUF_VBUF_WRITE_START_ERR_FOUND		RW	0x0
5	CBUF_WR_OUT_OF_RANGE_ERR_FOUND		RW	0x0
4	CBUF_VIRTUAL_ADDR_ERR_FOUND		RW	0x0
3	OUT_OF_RANGE_ERR_FOUND		RW	0x0
2	ADDR_ERR_FOUND		RW	0x0
1	DED_ERR_FOUND		RW	0x0
0	SEC_ERR_FOUND		RW	0x0

Figure 4. CFG_OCMC_ECC_ERROR

31	30	29	28	25	24	23	22	21	20	19	18	17	16
Reserved					CFG_DISCARD_DUP_ADDR	CFG_ADDR_ERR_CNT_MAX				CFG_DED_CNT_MAX			
R					R/W	R/W				R/W			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CFG_SEC_CNT_MAX															
R/W															

Bits	Field Name	Description	Type	Reset
31:25	RESERVED		R	0x0
24	CFG_DISCARD_DUP_ADDR	Do not save duplicate error address. This bit applies to the SEC, DED and ADDRERR FIFOs. 0x0: Save the duplicated addresses 0x1: Save only the unique addresses	RW	0x0
23:20	CFG_ADDR_ERR_CNT_MAX	Number of ADDR errors to trigger an interrupt (The value configured must be > 0 to generate an interrupt).	RW	0x1
19:16	CFG_DED_CNT_MAX	Number of DED errors to trigger an interrupt (The value configured must be > 0 to generate an interrupt).	RW	0x1
15:0	CFG_SEC_CNT_MAX	Number of SEC error to trigger an interrupt (The value configured must be > 0 to generate an interrupt).	RW	0x1

- Each OCMC RAM provides two interrupt lines as shown in the figure above.
  - Suggested software partitioning is to use OCMC_RAMx_IRQ to handle ECC errors and OCMC_RAMx_IRQ_CBUF to handle CBUF related errors.
  - When using OCMC_RAMx_IRQ interrupt line, software should enable the interrupt using INTR0_ENABLE_SET register.
  - When using OCMC_RAMx_IRQ_CBUF interrupt line, software should enable the interrupt using INTR1_ENABLE_SET register.
  - In the INTRx_ENABLE_SET register, bit[2:0] correspond to ECC errors.
  - Interrupt can be configured to be triggered only after a pre-defined threshold count of errors which can be set in CFG_OCMC_ECC_ERROR register.

5. Resume normal code execution.

3.3 Error Reporting

For details on the following registers, see the device-specific TRM.

- OCMC ECC reports three types of errors
  - SEC in data bits – Single bit errors with correction
  - DED in data bits – Double bit error (detection only)
  - ADDRESS (SEC/DED) – Since ECC parity uses a combination of ADDRESS and DATA bits, bit-error might be detected in the ADDRESS portion of the parity. ADDRESS errors are triggered under such scenarios
- STATUS_ERROR_COUNT
  - This provides the count of different errors
- STATUS_SEC_ERROR_TRACE/STATUS_DED_ERROR_TRACE
  - These provide the address location of SEC/DED errors
  - These registers implement a FIFO – each read will pop one entry out of the FIFO

- STATUS_ADDR_TRANSLATION_ERROR_TRACE
  - These provide the address location of ADDRESS errors
  - These registers implement a FIFO – each read will pop one entry out of the FIFO.
- STATUS_SEC_ERROR_DISTR_0/1/2/3/4
  - The first four register show the position of the bit in a 128-bit word where a single bit error was detected.
  - The fifth register indicates where bit-position where error occurred in ECC parity.

4 IPU EDC

4.1 Overview

The Cortex M4 based IPU sub-system include ECC protection for L1 cache and L2 RAM. It includes SEC and DED support for L1 cache-data, L1 tags and L2 RAM.

This feature is available only on TDA3x SoC only.

4.2 Programming Model

Since IPU has a Unicache – common cache for program and data, it can be enabled only when program is running from a non-cached location. It is, therefore, recommended to implement the IPU ECC functions to enabled from within SBL.

1. Disable and flush cache.
   a. Cache can be disabled by setting CACHE_CONFIG[1] to zero.
   b. Existing cache content can be flushed by using cache maintenance registers.
      i. Set CACHE_MTSTART to 0x0.
      ii. Set CACHE_MTEND to 0xFFFF_FFFF
      iii. Set CACHE_MAINT to 0x18
      iv. Wait for CACHE_MAINT to become zero.
2. Make section where current code is executing non-cacheable.
   a. In reference SBL provided by TI as part of VisionSDK and Starterware, SBL executes from OCMC region. OCMC is mapped in IPU Unicache AMMU by using the 0 medium page (CACHE_MMU_MED_ADDR_0, CACHE_MMU_MED_XLTE_0 and CACHE_MMU_MED_POLICY_0).
   b. The page can be disabled by setting CACHE_MMU_MED_POLICY_0 to 0x0.
3. Enable ECC using ECC_CFG register.

Figure 5. ECC_CFG

31	30	29	28	27	26	25	24
Reserved							
R							
23	22	21	20	19	18	17	16
Reserved							
R							
15	14	13	12	11	10	9	8
Reserved	L2RAM_SEC_AUTO_EN	L2RAM_SRESP_EN	L2RAM_DATA_MASK	L2RAM_CODE_MASK	L2RAM_ECC_EN	L1TAG_SEC_AUTO_EN	Reserved
R	R/W	R/W	R/W	R/W	R/W	R/W	
7	6	5	4	3	2	1	0
L1TAG_DATA_MASK	L1TAG_CODE_MASK	L1TAG_ECC_EN	L1DATA_SEC_AUTO_EN	L1DATA_SRESP_EN	L1DATA_DATA_MASK	L1DATA_CODE_MASK	L1DATA_ECC_EN
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bits	Field Name	Description	Type	Reset
31:15	RESERVED	Reserved. Read returns 0s.	R	0x00 0000
14	L2RAM_SEC_AUTO_EN	Enables auto-correction of data in case of SEC error in L2 RAM	RW	0
13	L2RAM_SRESP_EN	Enables error response to master in case of DED error in L2 RAM	RW	0
12	L2RAM_DATA_MASK	L2 RAM ECC Code Mask register. Enabling this will mask any write to data block while ECC will be updated on any writes to the memory. This is for test purpose.	RW	0
11	L2RAM_CODE_MASK	L2 RAM ECC Code Mask register. Enabling this will mask any write to ECC code block of memory. This is for test purpose.	RW	0
10	L2RAM_ECC_EN	L2 RAM ECC enable	RW	0
9	L1TAG_SEC_AUTO_EN	Enables auto-correction of data in case of SEC error in L1 Tag	RW	0
8	RESERVED	Reserved.	RW	0
7	L1TAG_DATA_MASK	L1 Tag ECC Code Mask register. Enabling this will mask any write to data block while ECC will be updated on any writes to the memory. This is for test purpose.	RW	0
6	L1TAG_CODE_MASK	L1 Tag ECC Code Mask register. Enabling this will mask any write to ECC code block of memory. This is for test purpose.	RW	0
5	L1TAG_ECC_EN	L1 Tag ECC enable	RW	0
4	L1DATA_SEC_AUTO_EN	Enables auto-correction of data in case of SEC error in L1 Data	RW	0
3	L1DATA_SRESP_EN	Enables error response to master in case of DED error in L1 Data	RW	0
2	L1DATA_DATA_MASK	L1 Data ECC Code Mask register. Enabling this will mask any write to data block while ECC will be updated on any writes to the memory. This is for test purpose.	RW	0
1	L1DATA_CODE_MASK	L1 Data ECC Code Mask register. Enabling this will mask any write to ECC code block of memory. This is for test purpose.	RW	0
0	L1DATA_ECC_EN	L1 Data ECC enable	RW	0

- ECC generation is enabled
  - For L1 Tag, by setting L1TAG_ECC_EN
  - For L1 Data, by setting L1DATA_ECC_EN
  - For L2 RAM, by setting L2RAM_ECC_EN
- SEC can be enabled
  - For L1 Tag, by setting L1TAG_SEC_AUTO_EN
  - For L1 Data, by setting L1DATA_SEC_AUTO_EN
  - For L2 RAM by setting L2RAM_SEC_AUTO_EN
- Data-aborts can be generated for DED by setting L1DATA_SRESP_EN, L1TAG_SRESP_EN and L2RAM_SRESP_EN for L1 Data, L1 Tag and L2 RAM, respectively.
- Support for diagnostics
  - Diagnostic software can generate ECC errors for testing by setting appropriate xxx_MASK bits.
  - Example: Setting L2RAM_ECC_MASK causes only data memory to be updated without any corresponding write to ECC memory. Any further read to the same address will generate an ECC error.
  - Example: Setting L2RAM_DATA_MASK causes only ECC memory to be updated without any corresponding write to L2RAM memory. Any further read to the same address will generate an ECC error.

4. Enable cache.
   a. Cache is enabled by setting CACHE_CONFIG[1] to 1.
   b. Ensure that at least one 32kB section is marked as cacheable in the Unicache AMMU registers.
5. Initialize ECC parity bits.
   a. ECC corresponding to L1 Data and L1 Tag can be initialized by preloading a 32kB (size of L1 cache) memory section using the cache maintenance registers.
      i. Assume a 32kB memory section starting at address “A” is marked as cacheable in the Unicache AMMU registers.
      ii. Set CACHE_MTSTART = A
      iii. Set CACHE_MTEND = A + 32kB – 1
      iv. Set CACHE_MAINT to 0x1
      v. Wait for CACHE_MAINT to become 0.
   b. ECC corresponding to L2RAM can be initialized by either using a CPU memset or the EDMA transfer to the entire L2RAM.

CAUTION

If CPU memset is used, cache must be disabled to prevent reads generated by the cache controller to ECC protected memory with uninitialized parity. In general, EDMA based memset is recommended for optimal performance.

6. Enable cache for the code section that was disabled in STEP 2 and resume the normal code execution.
7. Error handling.
   a. IPU has dedicated interrupt lines Unicache and L2RAM ECC errors.

Table 16. IPU ECC Error Interrupt Mapping

Interrupt Line	Interrupt Description
IPU_IRQ_80	L1TAG_ECC_SEC_IRQ
IPU_IRQ_81	L1TAG_ECC_DED_IRQ
IPU_IRQ_82	L1DATA_ECC_SEC_IRQ
IPU_IRQ_83	L1DATA_ECC_DED_IRQ
IPU_IRQ_84	L2_RAM_ECC_SEC_IRQ
IPU_IRQ_85	L2_RAM _ECC_DED_IRQ

5 DSP EDC

5.1 Overview

The C66x DSP subsystem provides error detection mechanisms for L1P Cache/RAM and L2 Cache/RAM. For more details, see the *TMS320C66x DSP CorePac User's Guide*. The following is a brief summary of the available features:

- L1P Cache/RAM only supports error detection
- L1D Cache/RAM supports error detection and correction (SEC/DED) on TDA2PX device
- L2 Cache/RAM supports error detection and correction (SEC/DED)
- All 64-bit aligned writes to L1P will generate a valid parity irrespective of whether error detection logic is enabled.
- All 128-bit aligned writes to L2 will generate a valid parity irrespective of whether error detection logic is enabled.

- Any non-aligned writes (64b for L1P, 128b for L2) will mark the parity as invalid.
- When error detection/correction logic is enabled, error detection/correction will only occur for memory segments with valid parity.

5.2 Programming Model

1. Initialize parity bits for L1P and L2 memory from HOST-CPU before DSP boot.
   a. Initialization of L1P parity
      i. If L1P is used only as cache, initialization for L1P memory is not required. This is because DSP cache controller will ensure 64-bit aligned writes to L1P and, thus, maintain a valid parity for complete L1P memory.
      ii. If L1P is used as RAM, EDMA-based memset must be done for the region allocated as RAM.
      iii. The initialization time can be optimized by avoiding sections that would get initialized due to code loading.

CAUTION

CPU-based memset cannot be used since parity is generated only for 64-bit writes.

   b. Initialization of L2 parity
      i. At reset, L2 is marked as RAM only. Therefore, the HOST-CPU that boots the DSP must ensure that L2 memory is initialized using a CPU-based or EDMA-based memset.
      ii. This step can be optimized by initializing only that part of L2 that is used as RAM.

NOTE: When using EMIF ECC, some part of L2 will always be used as RAM to allocate boot-time stack as explained in Section 2.5.1.3. This section must be initialized by HOST-CPU before booting or loading any code section in L2RAM.

CAUTION

CPU-based memset cannot be used since parity is generated only for 128-bit writes.

2. CPU-based memset cannot be used since parity is generated only for 128-bit writes.
   a. Enable L2 error detection and correction logic.
      i. Set register L2EDCMD (address: 0x0184_6008) to value 0x1
      ii. Register L2EDCEN(address: 0x0184_6030) provides additional control as to for which accesses is the EDC logic is active. By default, all accesses are checked by EDC logic.
   b. Enable L1P error detection logic
      i. Set register L1PEDCMD (address: 0x0184_6408) to value 0x1
   c. Diagnostic software can suspend parity generation by setting SUSP bit in L2EDCMD and L1PEDCMD registers. Refer to TMS320C66x DSP CorePac documentation (sprugw0c) for more details.
3. Resume normal code execution

5.3 Error Reporting

- The interrupt events shown in Table 17 are generated by the L1P/L2 EDC module.

Table 17. L1P/L2 EDC Error Events

EVT Number	EVT ID	Description
113	L1P_ED	Single bit error detected in L1P
116	L2_ED1	Corrected bit error detected in L2
117	L2_ED2	Un-corrected bit error detected in L2

- L2EDSTAT(address: 0x0184_6004)/L1PEDSTAT(address: 0x0184_6404) provide additional information about the error
- L2EDADDR(address: 0x0184_600C)/L1PEDADDR(address: 0x0184_640C) provide additional information about the location of the error
- Statistics for L2 EDC errors are available in L2EDCPEC (address: 0x0184_6018) and L2EDNPEC (address: 0x0184_601C) for correctable and non-correctable errors, respectively.
- EDC errors from DSP can be routed to external CPUs using the DSPx_IRQ_DSP_ERR line. This interrupt can be enabled using the DSP_SYS_ERRINT_IRQENABLE_SET register. The mapping of EDC interrupts is provided in the table *DSP ERRINT Interrupt Mapping* table in the device-specific TRM. The interrupt can be routed to any CPU in the SoC by using the appropriate DSPx_IRQ_DSP_ERR instance in the SoC interrupt crossbar.

5.4 Example Configuration for SYS/BIOS

```
/* SYSBIOS CFG configuration */
var Cache = xdc.useModule('ti.sysbios.family.c66.Cache');
Cache.initSize.l1pSize = Cache.L1Size_32K;
Cache.initSize.l1dSize = Cache.L1Size_32K;
Cache.initSize.l2Size  = Cache.L2Size_32K;
/* Configure MARs, by default cache is enabled for the entire memory region */
for (var i = 0; i < Program.cpu.memoryMap.length; i++)
{
   memSegment = Program.cpu.memoryMap[i];
   Cache.setMarMeta(memSegment.base, memSegment.len, Cache.Mar_ENABLE);
}

var Reset = xdc.useModule('xdc.runtime.Reset');
Reset.fxns[Reset.fxns.length++] = "&Utils_dspEccEnable";

/* Enable in C-code using starterware API */
Void Utils_dspEccEnable(Void)
{
    /* Enable ECC for L2 */
    ECCDspEnable(ECC_DSP_MEM_TYPE_L2,
                 ECC_ENABLE,
                 DSP_ECC_TIMEOUT);

    /* Disable Cache */
    Cache_disable(Cache_Type_L1P);

    /* Invalidate L1P Cache */
    Cache_invL1pAll();

    /* Enable ECC for L1P */
    ECCDspEnable(ECC_DSP_MEM_TYPE_L1P,
                 ECC_ENABLE,
                 DSP_ECC_TIMEOUT);

    /* Enable cache */
    Cache_enable(Cache_Type_L1P);
}
```

With the above configuration the DSP boot flow is as follows:

- SBL releases DSP subsystem reset, but keeps DSP CPU in reset. SBL initializes L2RAM with EDMA to initialize parity for L2 memory.
- Release CPU reset - .stack section must point to L2RAM
- Execute Utils_dspEccEnable – This will start L1P and L2 EDC functionality.
- Enable L2 cache as per configuration file – SYS/BIOS generates the code for this. It is executed immediately after the Reset.fxns[] defined in the configuration file. This will allow L2 cache to setup to ensure quanta-aligned to EMIF ECC regions.
- Setup MAR registers – SYS/BIOS generates the code for this. This step actually enables the cache functionality for data accesses.

6 EVE EDC

6.1 Overview

EVE supports parity-based error detection for all internal data memories (including DMEM, WBUF, IBUFLA, IBUFLB, IBUFHA, and IBUFHB) and for program cache. Parity is calculated on a write access size granularity of 8 bits (that is, there is 1 bit of parity per byte of memory). This parity is calculated for each read-access and checked against the previously stored parity. For additional details, see the *Memory Error Detection* section under the *Embedded Vision Engine* chapter in the device-specific TRM.

6.2 Programming Model

1. Initialize parity bits for all memories from HOST-CPU before EVE boot.
   a. DMEM, WBUF, IBUFLA, IBUFLB, IBUFHA and IBUFHB can be initialized using a CPU-based or an EDMA based memset.
   b. The initialization time can be optimized by avoiding sections that would get initialized due to code loading.
2. Enable parity checks in EVE register from HOST-CPU before EVE boot.

Table 18. EVE Error-Detection Control Register

Address	Register	Description
0x4208_0080	EVE_PMEM_ED_CTL	Parity checks for Program Cache
0x4208_0090	EVE_DMEM_ED_CTL	Parity checks for DMEM
0x4208_00A0	EVE_WBUF_ED_CTL	Parity checks for WBUF
0x4208_00B0	EVE_IBUF_ED_CTL	Parity checks for all IBUFxx

3. Boot EVE and resume normal code execution.

6.3 Error Reporting

- To receive interrupts for parity errors on EVE
  – Set the appropriate bits in EVE_ED_LCL_IRQENABLE_SET as defined in the “EVE Local and Output Error Detect Error Interrupt Mapping” section in the device-specific TRM.
  – This interrupt will be routed to EVE INTC0 on bit 28.
  – EVE NMI/exception handler must also be set up correctly as parity errors generate an error response to the read initiator.
- EVE NMI/exception handler must also be set up correctly as parity errors generate an error response to the read initiator.
  – Set the appropriate bits in EVE_ED_OUT_IRQENABLE_SET as defined in the *EVE Local and Output Error Detect Error Interrupt Mapping* section in the device-specific TRM.
  – This can be routed to any CPU by using the EVEx_IRQ_OUT0 crossbar instance in the SoC interrupt crossbar.

Revision History

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

Changes from A Revision (July 2016) to B Revision	Page
• Update was made to Table 1.	2
• Update was made in Section 2.1.	2
• Updates were made in Section 2.4.	11
• Update was made in Section 2.5.1.1.	11
• Update was made in Section 2.5.1.3.	12
• Updates were made in Section 3.2.	18
• Update was made in Section 4.2.	22
• Update was made in Section 5.1.	24

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources 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.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

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