Application Report
Lit. Number – December, 2012

Build Robust System on KeyStone Devices

Brighton Feng, Zhan Xiang, Jane Lu *Communication Infrastructure*

ABSTRACT

For a complex system, robustness is very important. The KeyStone devices provided lots hardware protection mechanisms to help user to implement a robust system, such as the memory protection, EDC/ECC and System trace. This application note discusses how to use these features to build a robust system on KeyStone device. Example codes are provided along with this application note.

Contents

Figures

Tables

1 Introduction

The KeyStone devices provide many features, which can support users to build robust application on it. Following figure shows those features.

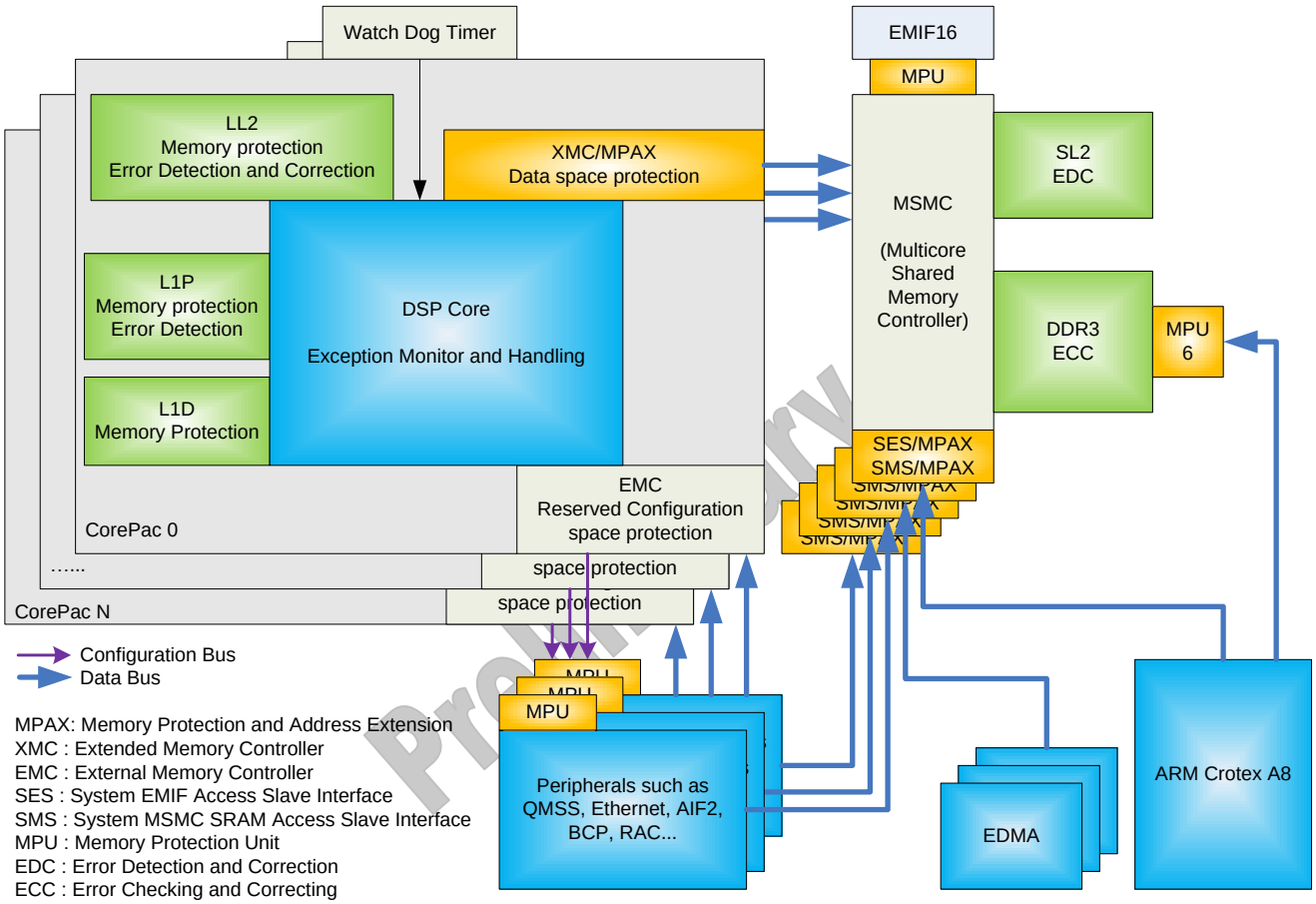


Figure 1. Robust system on Keystone devices

Above figure shows, LL2, L1P and L1D integrate memory protection modules inside them; LL2, SL2 and DDR controller integrate error detection and correction modules inside them; L1P integrates error detection module inside it.

In above figure, MPAX and MPU are actually modules attached on bus, which monitor the bus to detect and prevent illegal transfer.

Each DSP CorePac has one MPAX, which monitor the bus connecting to MSMC.

For other masters in the system, they are categorized by Privilege ID. For each Privilege ID, the MSMC integrates two MPAXs to monitor the transfer with that Privilege ID. One MPAX on SES, which protects access to DDR3; the other MPAX on SMS, which protects access to SL2. Refer to the device-specific data manual for Privilege ID of each master in the device.

Some MPUs are attached to the configuration ports of some peripherals to protect illegal access. Not all peripherals are protected by MPU, refer to the device-specific data manual for the list of peripherals protected by MPU.

Each CorePac has a watchdog timer to watch the activity of core, it can trigger NMI (NonMaskable Interrupt) or reset to the core if the core dies.

EMC can prevent DSP core from access reserved configuration space; XMC can prevent DSP core from access reserved data space.

All these modules can trigger exceptions to DSP core when problems happen, DSP core exception monitor module can records these status and trigger exception service routine of user's application to take proper actions.

This application note discusses the usage of these features. Example code/project is provided along with this application note. The example code is based on register layer CSL (Chip Support Layer). Most code use the register pointer defined like below:

```
#include <cslr_cgem.h>
#include <cslr_xmc.h>
#include <cslr_msmc.h>
CSL_CgemRegs * CGEM_regs = (CSL_CgemRegs *)CSL_CGEM0_5_REG_BASE_ADDRESS_REGS;
CSL_XmcRegs * XMC_regs = (CSL_XmcRegs *) CSL_XMC_CONFIG_REGS;
CSL_MsmcRegs * msmcRegs = (CSL_MsmcRegs *)CSL_MSMC_CONFIG_REGS;
```

The documentation about these features are distributed across the documentation of the various subsystems. The References section at the end of this application note lists all those documents. We assume user has read those documents before read this application note, so, this application note only intends to provide complementary information.

The information in this document applies to TCI6614/2, C6670, C6678/4/2, and the example code was verified on TCI6614 EVM, C6670 EVM and C6678 EVM. For other devices of KeyStone series, there may be minor difference (such as event number). Refer to device specific manual for details.

2 Memory Protection

There is another application note, "Memory Protection On Keystone Devices (SPRWIKI9012)", discusses the memory protection features on Keystone devices, there are many useful information not covered by user's guides.

Following table summarizes the difference of different memory protection modules.

Table 1. Different memory protection modules

Memory protection module	Position in System	page/segment/range		Comments
		Number	Size	
L1D memory protection	Slave port	16	2KB	Each CorePac has one set of L1D, L1P, LL2 and XMC.
L1P memory protection	Slave port	16	2KB	
LL2 memory protection	Slave port	32	(LL2 size)/32	
XMC MPAX	Master port	16	Programmable	

SMS MPAX	Master port	8	Programmable	Each Privilege ID has a pair of SMS and SES.
SES MPAX	Master port	8	Programmable	
MPU	Slave port	1~16	Programmable	Refer to device specific data manual for number of ranges and programmable size for each MPU.

There are multiple masters and multiple slaves in the system. The protection module in a slave's input port can prevent illegal access to this slave from all of the system masters; the protection module in a master's output port can prevent illegal access from this master to all of the system slaves.

Each page or segment or range's memory protection attributes are programmable.

2.1 L1 and LL2 Memory Protection

Refer to "Memory Protection" sections of "TMS320C66x CorePac User's Guide (SPRUGW0)" for basic information of L1 and LL2 Memory Protection.

L1 and LL2 memory protection only differentiates seven external requestor IDs, but there are 16 possible privilege IDs in the system. By default, system privilege ID 0~5 map to AID 0~5 of CorePac, all other privilege IDs map to AIDx.

The mapping between CorePac AID to system Privilege ID is programmable in the EMC, refer to section "External Memory Controller (EMC)" of the "TMS320C66x CorePac User's Guide (SPRUGW0)" for more information.

Please note, the AID of IDMA is the number of the CorePac it belongs to; the privilege ID of an EDMA transfer is the number of the CorePac which programs it.

Normally, L1 are configured as cache, for this case, all Memory Protection Page Attribute registers of L1 should be cleared to 0 to prevent any other masters' access.

The registers of the memory protection modules in CorePac (including L1, LL2 and **XMC/MAPX)** can be protected with a lock. By default, these registers are unlocked. User's software can lock these registers with a user defined key, these registers can not be accessed until it is unlocked with the same key.

2.2 Shared Memory Protection – MPAX

Refer to "Extended Memory Controller (XMC)" section of "TMS320C66x CorePac User's Guide (SPRUGW0)" for basic information of shared memory protection for CorePac; Refer to "Memory Protection and Address Extension (MPAX)" section of "KeyStone Architecture Multicore Shared Memory Controller User Guide (SPRUGW7)" for basic information of shared memory protection for other masters in the system.

Below is a sample configuration table for XMC/MPAX in the example code along with this application note. Each row represents the configuration of one segment in MPAX.

```
MPAX_Config XMC_MPAX_cfg_table[]=
```

```
{
    /*BADDR         RADDR           SegementSize      AccessPermisionMask
    32-bit virtual  36-bit physical in byte, must     Access types allowed
    base address    address right   be power of 2     in this address range
                    shift by 4                                        */
    {0x0C000000,    0x00C000000>>4,  2*1024*1024,    MP_SR|MP_SW|MP_SX|MP_UR|MP_UW|MP_UX},/*SL2*/
    {0x10000000,    0x010000000>>4,  0x04000000,     MP_SR|MP_SW|MP_UR|MP_UW},/*LL2 global*/
    {0x18000000,    0x00C000000>>4,  2*1024*1024,    MP_SR|MP_SW|MP_UR|MP_UW},/*SL2 remap */
    {0x20000000,    0x020000000>>4,  0x04000000,     MP_SR|MP_SW|MP_UR|MP_UW},/*peripherals*/
    {0x30000000,    0x030000000>>4,  0x08000000,     MP_SR|MP_SW|MP_UR|MP_UW},/*peripherals*/
    {0x40000000,    0x040000000>>4,  0x10000000,     MP_SR|MP_SW|MP_UR|MP_UW},/*HyperLink*/
    {0x60000000,    0x060000000>>4,  0x10000000,     MP_SR|MP_SW|MP_UR|MP_UW},/*PCIE*/
    {0x21000000,    0x100000000>>4,  4*1024,         MP_SR|MP_SW|MP_UR|MP_UW},/*DDR3 config*/
    {0x80000000,    0x800000000>>4,  0x10000000,     MP_SR|MP_SW|MP_SX|MP_UR|MP_UW|MP_UX},/*DDR3*/
    {0x90000000,    0x810000000>>4,  0x10000000,     MP_SR|MP_SW|MP_UR|MP_UW}/*DDR3*/
};
```

Logical address lower than 0x0C00_0000 will not go to XMC. Accesses to addresses 0x0000_0000 to 0x07FF_FFFF are decoded internally to C66x CorePac. This address range includes the internal and external configuration busses, as well as the L1D, L1P and L2 memories.

Logical address between 0x0C00_0000 and 0x0FFF_FFFF will always go through L1 cache and prefetch buffer (for read), MAR registers for this space are hard wired, not programmable. Accesses to this space will NOT go through L2 Cache before go to MPAX of XMC, this is called “fast SL2 RAM path”.

Logical address equal or higher than 0x1000_0000 goes through L2 Cache Controller firstly and then goes to MPAX of XMC, this normal path adds latency of one cycle.

So, based on above configuration example, access SL2 through logical address of 0x0C00_0000 is faster than through remapped logical address of 0x1800_0000. But the MAR (Memory Attribute Register) corresponding to 0x1800_0000 is programmable, which can make the access to SL2 through 0x1800_0000 non-cacheable or non-prefetchable.

Attentions should be paid to the RADDR (Replacement address), it is (36-bit physical address)>>4. The common wrong configuration for DDR3 is:

```
    {0x80000000,    0x800000000>>4,  0x10000000,     MP_SR|MP_SW|MP_SX|MP_UR|MP_UW|MP_UX},/*DDR3*/
    {0x90000000,    0x900000000>>4,  0x10000000,     MP_SR|MP_SW|MP_UR|MP_UW}/*DDR3*/
```

or

```
    {0x80000000,    0x80000000,  0x10000000,     MP_SR|MP_SW|MP_SX|MP_UR|MP_UW|MP_UX},/*DDR3*/
    {0x90000000,    0x90000000,  0x10000000,     MP_SR|MP_SW|MP_UR|MP_UW}/*DDR3*/
```

Please note, the DDR3 physical address start from 0x8:0000_0000, while 0x9:0000_0000 is 4GB apart from the start address, it is normally an invalid address in most systems.

In real system, user should fully utilize all segments of MPAX to divide memory space into small pieces as much as possible, and carefully set limited access permission for each segment.

Unused address should not be mapped. Access to unmapped address will be denied by MPAX, and exception will be reported for it, this is very helpful for capturing software error.

SMS/MPAX only allows access to SL2. Below is an example configuration for it:

```
MPAX Config SMS MPAX cfg table[]=
{
    /*BADDR         RADDR           SegementSize      AccessPermisionMask
    32-bit virtual  36-bit physical in byte, must     Access types allowed
    base address    address right   be power of 2     in this address range
```

```
                    shift by 4                                                  */
    {0x0C000000,    0x00C000000>>4,  2*1024*1024,    MP_SR|MP_SW|MP_SX|MP_UR|MP_UW|MP_UX},/*SL2*/
    {0x18000000,    0x00C000000>>4,  2*1024*1024,    MP_SR|MP_SW|MP_UR|MP_UW}/*SL2 remap */
};
```

SES/MPAX is used to protect the access to DDR3. Below is an example configuration for it:

```
MPAX Config SES MPAX cfg table[]=
{
    /*BADDR         RADDR           SegementSize      AccessPermisionMask
    32-bit virtual  36-bit physical in byte, must     Access types allowed
    base address    address right   be power of 2     in this address range
                    shift by 4                                          */
    {0x21000000,    0x100000000>>4,  4*1024,          MP_SR|MP_SW|MP_UR|MP_UW},/*DDR3 config*/
    {0x80000000,    0x800000000>>4,  0x10000000,      MP_SR|MP_SW|MP_SX|MP_UR|MP_UW|MP_UX},/*DDR3*/
    {0x90000000,    0x810000000>>4,  0x10000000,      MP_SR|MP_SW|MP_UR|MP_UW}/*DDR3*/
};
```

If two masters exchange data in shared memory, user must make sure the address used by the two masters are mapped to same physical address.

Please note, the privilege ID of an EDMA transfer is the number of the CorePac that programs it.

CAUTION:

A MPAX segment can only be modified when there is no access to the space of this segment. Any data in cache or prefetch buffer from the space of this segment must be writeback and invalidated before the segment modification.

It is recommended to configure the MPAX at the very beginning of application software before any shared memory is used. Code and data used for CorePac MPAX configuration should be allocated in LL2.

If a MPAX segment must be modified on-the-fly, the safer way is, to write the new configuration to a unused higher segment, and then clear the old segment. This is based on the fact that higher numbered segments take precedence over lower numbered segments.

Following procedure needs to be followed before the MPAX segment registers are modified:

1. Write back and invalidate affected cache lines. Even for non-writable memory, CACHE_wbInvL2 () rather than CACHE_invL2 should be used.
2. Invalidate the prefetch buffer if prefetching is enabled for affected memory locations.
3. Execute “MFENCE” instruction to ensure all write-back invalidate have completed.

The registers of MPAX in CorePac is also protected by the memory protection registers’ lock of CorePac. While the memory protection page attribute registers of each MPAX of SES and SMS are protected by separated lock for each SES and SMS in MSMC. All other registers of MSMC are protected by another lock for non-MPAX registers of MSMC.

2.3 Peripherals Configuration port protection – MPU

Refer to “KeyStone Architecture Memory Protection Unit User Guide (SPRUGW5)” for basic information of MPU.

MPU0, MPU1, MPU2 and MPU3 are same on all devices. The number of additional MPUs, the number of ranges supported by each MPU, and the reset values of the MPU configuration registers are different on different device. Refer to “Memory Protection Unit (MPU)” section of device specific data manual for more information.

The key different between MPU and MPAX is that, if the transfer address does not match any address range, then the transfer is allowed by MPU; while MPAX denies the transfer if the address does not match any segment.

Please note, the MPU units assume accesses to be allowed by default, if not specifically denied by the MPPA settings. In particular, the MPU first checks the transfer’s privilege ID against the AID bit settings of the MPPA registers. If the AID bit corresponding to the privilege ID is 0, then the range will not be checked (and access is allowed). For example, setting MPPA= 0 allows all access to the range; to disable any access to a range, the MPPA should be set as 0x03FFFC00. This is also different from the MPPA setting of L1 and LL2 memory protection, which denies access from AID with corresponding bit in MPPA is 0.

In the case that a transfer matches multiple address ranges of MPU, all the overlapped ranges must allow the access, otherwise the access is not allowed. The final permissions given to the access are the lowest of each type of permission from any hit range. Therefore, if a transfer matches 2 ranges, one that is RW and one that is RX, then the final permission is just R. This is also different from MPAX. If a given address falls into more than one MPAX segments, higher numbered segment take precedence over lower numbered segments. The MPAX only consults the highest numbered segment among all matches to determine the permission, and ignores all other matches.

Below is a sample configuration table for MPU1 in the example code along with this application note. Each row represents the configuration of one range in MPU.

```
MPU Range Config MPU1 cfg table[]=
{
  /*StartAddr  EndAddr     AccessPermisionMask*/
  {0x34020000, 0x3403FFFF, MP_AID11|MP_NS|MP_EMU|MP_SR|MP_UR}, /*Queue 0~8191, deny write from AID11*/
  {0x34020000, 0x34027FFF, MP_AID8_15|MP_NS|MP_EMU|MP_SR|MP_UR},/*Queue 0~2047,deny write from AID8~15*/
  {0x34038000, 0x3403FFFF, MP_AID0_7|MP_NS|MP_EMU|MP_SR|MP_UR},/*Que 6144~8191,deny write from AID0~7*/
};
```

With above configuration, the queues are protected as:

- Queue 0~2047 are only writeable (PUSH) by AID0~7
- Queue 2048~6143 are writeable (PUSH) by all AIDs except for AID11
- Queue 6144~8191 are only writeable (PUSH) by AID8~15 except for AID11

The MPU6 was specially designed for the ARM on TCI6614/12 to protect illegal access from ARM to DDR3. Please note, it is using the lower 32-bit of the physical DDR address (0x00000000~0xFFFFFFFF) for range configuration.

Please note, to clear the MPU exception/interrupt event, EOI register should be written with **0** at the last step of the service routine.

The MPU events in TCI6614/2 are different from other KeyStone devices. All the MPU0~7 events in TCI6614/2 are combined into single event and connected to the CIC0 as one system event. We have to clear the MPU event flags in MPU first and then are able to clear the CIC flag because the TCI6614/2 MPU event is level interrupt event, not pulse interrupt event. While for pulse interrupt events, CIC flag should be cleared firstly and then clear the source flag.

Additionally, MPU5 (for BCP) in TCI6614 is only accessible when BCP is enabled through PSC, that is, access to MPU5 register in TCI6614 without enabling BCP will trigger bus access error.

2.4 Reserved space protection

Reserved space (invalid address) is protected against abnormal access. Reads from invalid address return garbage; writes to invalid address are blocked. Access to reserved space can generate exceptions, this is very helpful for capturing bug of software.

DSP core reads from any invalid address will always trigger L1D memory protection exception because this access finally goes through L1D controller.

DSP core executes from invalid address will trigger Instruction fetch exception.

For invalid write, depends on the address, different exception event will be triggered.

When DMA access invalid address, bus error will also be reported in the DMA module. The DMA error event can also be routed to DSP core as exception.

3 EDC/ECC

EDC (error detection and correction) or ECC (Error Checking and Correcting) are designed for soft error. A soft error is a signal or datum which is wrong, but is not assumed to imply a breakage. After observing a soft error, there is no implication that the system is any less reliable than before. In the spacecraft industry this kind of error is called a single-event upset. In a memory system, a soft error changes an instruction in a program or a data value. Soft errors typically can be remedied by rebooting the device, which a hardware failure typically can not be recovered by rebooting. A soft error will not damage a system's hardware; the only damage is to the code or data that is being processed. The causes of soft error include:

1. Alpha Particle Emissions and Cosmic rays creating energetic neutrons and protons. The rate at which this happens is determined by the geographical location of the device and the surrounding environment. Normally, it may only happen several times in several years in one device.

2. Soft errors can also be caused by random noise, interference or signal integrity problems, such as inductive or capacitive crosstalk on board. If the soft error rate is much higher than the rate caused by above item 1 in theory, hardware design should be checked to find additional causes. A common cause is supplying lower voltage than expected, which makes the device more sensitive to noise or interference.

EDC or ECC mechanism are implemented through keystone all levels of memory, following table compares these mechanism in different memory modules:

Table 2. EDC/ECC mechanism implement in keystone memory

memory	error detection	Error correction	Segment line size
L1P	1 bit	N/A	64bit
CorePac L2	2 bit	1bit	128bit
Shared L2 (MSMC)	2 bit	1bit	256bit
DDR	2 bit	1bit	64bit

3.1 L1P Error Detection

For L1P and LL2 EDC basic information, please refer to "TMS320C66x DSP CorePac User Guide (SPRUGW0)"

Parity bits generation and checking: the parity info is generated by 64bit aligned DMA write or L1P cache fetch. Non 64-bit aligned DMA access will invalid the parity info. The L1P EDC logic will check the parity info for 256bit aligned program fetch or 64bit aligned DMA read.

Error Detection Setup: L1P error detection feature is disabled by default after the device is reset. Once the "EN" bit of L1PEDCMD register is set, the ED logic of all L1P memory is enabled. Below is example code to enable L1P ED function along with this application codes:

```
void L1P_EDC_setup()
{
  Uint32 preL1PMPPA[16];

  /* 1. Disable the EDC */
  CSL_CGEM_disablePMCErrorDetection();

  /* 2. Clear any EDC errors */
  CSL_CGEM_clearPMCErrorDetectionStatus(1, 1);

  /* 3. Memory Scrubbing with IDMA, generate the parity bits*/
  memcpy(preL1PMPPA, (void *)CGEM_regs->L1PMPPA, 64);//save protection attributes
  L1P_memory_protection_cfg(0xFFFF); //enable IDMA access to L1P
  IDMA_copy(0x00E00000, 0x00E00000, 32*1024, DMA_WAIT);
  L1_MPPA_setup(CGEM_regs->L1PMPPA, preL1PMPPA);//restore protection for L1

  /* 4. Enable the EDC*/
  CSL_CGEM_enablePMCErrorDetection();
}
```

Notes: L1P ED function could not work alone without enabling Local L2 EDC, we're debugging the root cause and will update this document once it is clear.

Error Handling for L1P cache access: When there is a parity error for program fetch from the L1P cache, there's no dedicated system level event for it, however, error detection logic sends a direct exception event to the DSP (IERR.IFX event) and user can use internal exception event to capture this error. The L1PEDSTAT register PERR bit field will also be set. The L1PEDARRD register will record the address info which consist the error bit. In the exception service routine of L1P error, the cache line include the error address should be invalidated.

Error Handling for DMA access: When there is a parity error for access from DMA/IDMA, #113 system event is generated. User can use this event to capture the error. The L1PEDSTAT register DERR bit field will be set and L1PEDARRD register will record the address info which consist the error bit.

L1P EDC function verification: The L1P EDC logic can be suspended by set the SUSP bit of L1PEDCMD register. With this feature, the software can simulator EDC error and verify EDC function. The example along with this application code provide the demo code to verify the L1P EDC function, the related function is `L1P_ED_test()`.

3.2 LL2 Error Detection and Correction

Parity bits generation and checking: the parity info is generated by 128-bit segment L2 memory write. Non 128-bit aligned address write or less than 128-bit write will invalid the parity information. The LL2 EDC logic will check the parity info for each 128-bit aligned memory read. Please refer to "TMS320C66x DSP CorePac User Guide (SPRUGW0)" for more information.

Error Detection and Correction Setup: LL2 EDC feature is disabled by default after the device is reset. Unlike some C64+ DSP, the keystone DSP could not enable EDC by memory page category. Once the EDC is enabled, the EDC logic works for the whole corePac L2 memory range. However, it could be enabled for different memory access requestor separately: L1D controller, L1P controller or DMA controller. For example, if user only need check EDC for codes section, below three fields need to be enabled:

1. Set EN bit of L2EDCMD register to enable the LL2 EDC logic
2. Set the L1PSEN bit of L2EDCEN register to enable L2 SRAM memory EDC logic for L1P access
3. Set the L1PCEN bit of L2EDCEN register to enable L2 Cache memory EDC logic for L1P access

The LL2 EDC logic does not initialize the parity RAM when transitioning from the disabled to the enabled state. Thus, upon entering the enabled state, there may be invalid parity values in the parity RAM whose corresponding valid bits are also set. The EDC setup sequence stated in "TMS320C66x DSP CorePac User Guide (SPRUGW0)" must be followed. Below is example code to enable L2 EDC function along with this application note:

```
void LL2_EDC_setup()
{
  int i;
  unsigned int uiByteCnt= 512*1024;
  TDSP_Board_Type DSP_Board_Type;

  /* 1. Disable the EDC */
  CSL_CGEM_disableL2EDC();

  /* 2. Clear any EDC errors */
  CSL_CGEM_clearL2EDCErrorStatus(1, 1, 1, 1);

  /* 3. Memory Scrubbing with IDMA, generate the parity bits*/
  DSP_Board_Type = KeyStone_Get_dsp_board_type();
  if((C6670_EVM == DSP_Board_Type)
    ||(TCI6614_EVM == DSP_Board_Type))
  {
```

```
    uiByteCnt= 1024*1024;
  }

  /*Each IDMA can transfer up to 65532 bytes,
  here we transfer 32KB each time*/
  for(i=0; i< (uiByteCnt>>15); i++)
  {
    IDMA_copy((0x00800000 + i*(1<<15)), (0x00800000 + i*(1<<15)),
      (1<<15), DMA_WAIT);
  }

  /* 4. Enable the EDC*/
  CSL_CGEM_enableL2EDC();
  CGEM_regs->L2EDCEN= (1<<CSL_CGEM_L2EDCEN_DL2CEN_SHIFT)
    |(1<<CSL_CGEM_L2EDCEN_DL2SEN_SHIFT)
    |(1<<CSL_CGEM_L2EDCEN_PL2CEN_SHIFT)
    |(1<<CSL_CGEM_L2EDCEN_PL2SEN_SHIFT)
    |(1<<CSL_CGEM_L2EDCEN_SDMAEN_SHIFT);

}
```

Error Handling for access from L1D controller: Error Detection is performed on all data fetches from LL2 by L1D cache without any correction. The error will be reported through No.117 system event (L2_ED2: Uncorrected bit error detected) to DSP core whatever it is 1-bit error or multi-bit error.

Error Handling for access from L1P controller and DMA controller: 1-bit error is corrected and No.116 system event (L2_ED1: Correctible bit error detected) is used to report this error. 2-bit error could be detected and No.117 system event is used to report this error.

Details of 1-bit error handling for different memory access requestor are listed as following table:

Table 3. 1 bit error handling of LL2 EDC

Requester	L1D	L1P	DMA
Status register (L2EDSTAT)	"DERR" field = 1 " NERR" field : no update "BITPOS" field: all zero	"PERR" field = 1 " NERR" field = 1 "BITPOS" field: error bit position	"DMAERR" field = 1 " NERR" field = 1 "BITPOS" field: error bit position
Error address (L2EDADDR)	Error address is recorded	Error address is recorded	Error address is recorded
Correctable error counter L2EDCPEC	No update	Increase by 1	Increase by 1
Non-correctable error counter L2EDNPEC	Increase by 1	No update	No update

The error counters (L2EDCPEC, L2EDNPEC) is very useful, allows software in a long-running system to assess the rate and pattern of parity-error occurrences.

Details of 2-bit error handling for different memory access requestor are listed as following table:

Table 4. 2 bit error reporting of LL2 EDC

Requester	L1D	L1P	DMA
Status register (L2EDSTAT)	"DERR" field = 1 " NERR" field : no update "BITPOS" field: all zero	"PERR" field = 1 " NERR" field = 2 "BITPOS" field: all zero	"DMAERR" field = 1 " NERR" field = 2 "BITPOS" field: all zero
Error address (L2EDADDR)	Error address is recorded	Error address is recorded	Error address is recorded
Correctable error counter L2EDCPEC	No update	No update	No update
Non-correctable error counter L2EDNPEC	Increase by 1	Increase by 1	Increase by 1

If there's multi bit error larger than 2 bits, the EDC logic may detect it as 1-bit or 2-bit error, or the EDC can not detect it. That is why we say EDC can only detect 2-bit error or correct 1-bit error.

Normally, the rate of soft error is very low, one-bit error occurs firstly, and after relative long time, the second error bit may occurs. Since one-bit error is correctible and two-bit error is not correctible, we should try to correct one bit error before the second error bit happens.

The operation to correct one-bit error is normally called "scrubbing". To scrub a range of memory, the program initiates an IDMA with the source and destination addresses equal to each other; the byte count is set to cover the desired block. The address range must be 128-bit aligned and a multiple of 128-bits for the entire range to be scrubbed. As the IDMA reads the block of memory from LL2, the EDC hardware corrects any single-bit errors that might be present on 128-bit words that have valid parity. When IDMA writes the data back to L2, the EDC generates parity for the corrected data and marks it valid.

Scrubbing is normally done in the interrupt service routine of 1-bit error for the error data block. But some data may not be accessed after 1-bit error happens and before 2-bit error happens, 1-bit error can not be reported automatically without access. To prevent this case, we should periodically scrub whole memory space to correct potential 1-bit error. Below is the example code for LL2 EDC scrubbing along with this application note.

```
Uint32 uiLL2_scrub_addr=0x800000;
void LL2_EDC_scrub(Uint32 uiByteCnt)
{
  Uint32  uiLL2EndAddress= 0x00880000;
  TDSP_Board_Type DSP_Board_Type;

  uiByteCnt &= 0xFFFFFFF0;  //size must be multiple of 128 bits

  IDMA_copy(uiLL2_scrub_addr, uiLL2_scrub_addr, uiByteCnt, DMA_NO_WAIT);

  uiLL2_scrub_addr+= uiByteCnt;
```

```
  DSP_Board_Type = KeyStone_Get_dsp_board_type();
  if((C6670_EVM == DSP_Board_Type)
    ||(TCI6614_EVM == DSP_Board_Type))
  {
    uiLL2EndAddress= 0x00900000;
  }

  //wrap back
  if(uiLL2_scrub_addr >= uiLL2EndAddress)
    uiLL2_scrub_addr=0x800000;
}
```

Normally, this function can be called in a timer interrupt. For example, if call this function in a timer interrupt with period of 600 seconds like below:

```
      /*scrub 1024 bytes in LL2 for EDC*/
      LL2_EDC_scrub(1024);
```

Then, it will take about 7 days to scrub 1MB memory.

Since the scrubbing operation competes against normal memory operation, thus affects the performance of normal memory operation, so, it should not be done too frequently, but it should be done before 2-bit error happens. This is a trade-off up to system designer.

LL2 EDC function verification: The LL2 EDC logic can be suspended by set the SUSP bit of L2EDCMD register. By this feature, the software can simulate EDC error and verify EDC function. The example along with this application note provide the demo code to verify the LL2 EDC function, the related function is `LL2_EDC_test()`.

3.3 SL2 Error Detection and Correction

For Shared L2 EDC basic information, please refer to ”KeyStone Architecture Multicore Shared Memory Controller User Guide (SPRUGW7)”.

Parity bits generation and checking: there’re two mechanism used for MSMC parity info generations and detection:

1. When there are 256-bit segment memory write regardless which master is, the parity info is updated and set as valid. Less than 256-bit write will invalid the parity info. The parity info is tracked when there’s 256-bit segment memory read requested by DSP masters.

2. The MSMC contains a background error correction hardware called the Scrubbing Engine that periodically refreshes the parity bits for the memory. The period cycle is set by bitfield REFDEL of SMEDCC register and the scrubbing burst size is four 32-byte chunks in a period. Besides refreshing the parity bits, the scrubbing engine will also report the EDC error once it detects 1bit or 2 bits error. Details mechanism are described in the MSMC user guider.

MSMC hardware invalidates the parity info when the DSP is reset and it will re-initialize the parity info. Software must consult the PRR (Parity RAM Ready) bit in the SMEDCC before it makes the first read to MSMC memory.

Error Detection and Correction Setup: The SL2 EDC logic of scrubbing engineer is enabled once the device is reset and it will generate the parity info at the background. The software does not need to use DMA to scrubbing memory as it does for the LL2 EDC, it just need to check the PRR (Parity RAM Ready) bit in the SMEDCC to make sure the parity bit is generated. In order to enable error correction, the ECM bit of SMEDCC should be enabled as well. Please be aware that, error correction adds 1 cycle latency for data read from SL2.

Below is example code to enable MSMC EDC function along with this application note:

```
void KeyStone_SL2_EDC_enable(Uint32 scrubCnt)
{
  if(msmcRegs->CFGLCKSTAT&CSL_MSMC_CFGLCKSTAT_WSTAT_MASK)
    CSL_MSMC_unlockNonMPAX();

  /*Software must wait for the PRR (Parity RAM Ready) bit before making
  the first read access to MSMC RAM after reset.*/
  while(0==(msmcRegs->SMEDCC&CSL_MSMC_SMEDCC_PRR_MASK));

  /* set scrubbing period value */
  if(scrubCnt>255)
    scrubCnt= 255;
  CSL_MSMC_setCounterBankRefreshRead(scrubCnt); //the scrubbing engine works every
scrubCnt*1024 cycle*/

  /* clear EDC errors and enable EDC event*/
  msmcRegs->SMIRC = 0xf;
  msmcRegs->SMIESTAT |= (CSL_MSMC_SMIESTAT_NCSIE_MASK
                | CSL_MSMC_SMIESTAT_CSIE_MASK
                | CSL_MSMC_SMIESTAT_NCEIE_MASK
                | CSL_MSMC_SMIESTAT_CEIE_MASK);

  //enable SL2 EDC
  CSL_MSMC_setECM(1);

  CSL_MSMC_lockNonMPAX();
}
```

Error reporting mechanism: the MSMC user guider has details info about error reporting mechanism, it is summarized as below table.

Table 5. 1 bit error reporting of MSMC EDC/ECC

requester	Scrubbing engine access	None Scrubbing engine access
Status register (SMESTAT)	“CSES” field = 1	“CEES” field = 1
Error address	Error address is recorded by “SMCEA” register	Error address is recorded by “SMCERRAR” and “SMCERRXR” register
Correctable error counter	“SCEC” field of SMSECC register is increased by 1	No counter register
Non-correctable error counter	No update	No counter register

Please note, the error address reported by the scrubbing engine is offset address starts from 0, while the error address recorded for non-scrubbing access is the address of SL2 in the device, which starts from 0x0C000000.

Table 6. 2 bit error reporting of MSMC EDC/ECC

requester	Scrubbing engine access	None Scrubbing engine access
Status register (SMESTAT)	"NCSES" field = 1	N"CEES" field = 1
Error address	Error address is recorded by "SMNCEA"	Error address is recorded by "SMNCERRAR" and "SMNCERRXR" register
Correctable error counter	No update	No counter register
Non-correctable error counter	"SNCEC" field of SMSECC register is increased by 1	No counter register

MSMC EDC function verification: The MSMC EDC logic can be suspended by set the PFn bits (bit 0~3) of SMEDCTST register. The offset address of SMEDCTST is 0x58. The PFn bit fields are a single bit per SL2 RAM bank (PF0~3 for bank 0~3) that disable the write enable to the parity RAMs. In effect, this freezes the parity RAMs corresponding to the bank thus allows a program to inject faults by deliberately corruption the parity state associated with a SL2 memory location and test the working of the detection and correction logic. The sequence for doing this is:

1. Write a know value to a location of the bank being tested. This will initialize the parity bits for that location correctly.
2. Freeze the parity values by writing a 1 to the appropriated PF bit in the SMEDCTST.
3. Modify the location written by writing into any byte at the location, the value written should modify 1 bit if the correction is being checked and 2 bits if detection is being checked. The parity value for the location is now out of sync with the data value in the location.
4. read the location back. This would result in the parity error of the chosen type.

The example along with this application code provide the demo code to verify the SL2 EDC function, the related function is SL2_EDC_test().

3.4 DDR3 ECC

For DDR ECC basic information, please refer to "KeyStone Architecture DDR3 Memory Controller's user guider (SPRUGV8)".

Parity bits generation and checking: Eight-bit ECC is calculated over 64-bit data quanta for all 64-bit aligned accesses that are within the address ranges protected by ECC. The address ranges are specified in the ECC Address Range 1 and 2 register. ECC is disabled by default and can be enabled by setting the ECC EN bit to 1 in the ECC Control Register.

Error reporting mechanism: The ECC is read and verified during reads. If there is a one-bit error, the DDR3 memory controller corrects the data and sends it on the read interface. If there are two or more bit errors, the DDR3 memory controller sends a read ECC error interrupt. Please be aware that a write access with byte count that is not a multiple of 64-bit quanta, or with a non-64-bit-aligned address performed within the address range protected by ECC, will result in a write ECC error interrupt. There is only one interrupt from the DDR3 controller that is routed to CIC for CorePac and the CPU/2 EDMA channel controller. The source of the ECC error interrupt can be checked in the Interrupt Raw Status Register.

DDR ECC function verification: the DDR ECC function could not be verified by software method. The way TI validated this in the lab was to enable the ECC, write to the DDR3 memory space, then used a wire to ground one of the DDR data pins and read it back to generate an error.

DDR ECC verification is still undergoing. This section will be updated once the verification complete.

4 Other Robust features

4.1 Watch Dog Timer

Refer to “Watchdog Timer Mode” section of “KeyStone Architecture Timer64 User Guide (SPRUGV5)” basic information of watch dog time.

Timer0 through (N-1) are dedicated to each of the N CorePacs as a watchdog. Timer8 is dedicated to the ARM as a watchdog timer in TCI6614/12.

When operating in watchdog mode, the timer counts down to 0 and generates an event. It is a requirement that software writes to the timer before the count expires, after which the count begins again. If the count ever reaches 0, the timer event output is asserted. Watch dog timer event may trigger local core reset, device reset or NMI exception, this can be selected by programming ‘‘Reset Mux (RSTMUXx) Register’’ described in device specific data manual.

It is more flexible to let watch dog event trigger the NMI exception, in the NMI exception service routine, the error reason and some key status information may be recorded or reported to host for failure analysis, and then, software can reset the device if it can not be recovered.

4.2 EDMA error detection

Refer to “Error Interrupts” section of “KeyStone Architecture Enhanced Direct Memory Access (EDMA3) Controller User Guide (SPRUGS5)” for basic information of EDMA CC errors.

Refer to “Error Generation” section of “KeyStone Architecture Enhanced Direct Memory Access (EDMA3) Controller User Guide (SPRUGS5)” for basic information of EDMA TC errors.

All EDMA error events should be routed to CorePac as exception.

Event miss error is most common EDMA CC error, which means the EDMA can not complete the transfer as quick as required or false event triggers unexpected EDMA transfer.

BUS error is most common EDMA TC error, which normally means the EDMA access wrong address (reserved address or protected address).

4.3 Interrupt drop detection

Interrupt drop or miss is a very common issue (also often be neglected) in real-time system. Interrupt drop detection is a very useful feature to capture this exception. Refer to "Interrupt Error Event" section of "TMS320C66x DSP CorePac User Guide (SPRUGW0)" for basic information of interrupt drop detection.

Interrupt drop detection should be enabled for the interrupts which are intentionally routed to DSP core and serviced by software. This can be enabled by following code **after all** other interrupt configuration:

```
CGEM_regs->INTDMASK= ~IER; /*only monitor drop of enabled interrupts*/
```

Please note, when you enable interrupt drop detection and debug your program with breakpoint or single step with CCS/emulation. It is highly possible that interrupt drop error is reported because interrupt is not serviced during emulation halt. If you want to ignore it, you can temporarily disable interrupt drop detection for some or all interrupts during these debug activities, but please do not forget to re-enable them when you release your program.

5 Exception handling

Refer to "CPU Exceptions" section of "TMS320C66x DSP CPU and Instruction Set Reference Guide (SPRUGH7)" for basic information of exception handling.

Refer to "Interrupt Controller" section of "TMS320C66x DSP CorePac User Guide (SPRUGW0)" for basic information of interrupt or exception events routing.

5.1 Exception events routing

All error events originate from or triggered by CorePac are directly routed to INTC of CorePac. The errors should be treated as exception are listed in following table.

Table 7. Exception events directly connected to CorePac

Event Number	Event	Description
10	MSMC_mpf_error_n	MSMC Memory protection fault caused by privilege ID of this CorePac. For example, an EDMA programmed by this CorePac triggers MSMC memory protection.
96	INTERR	Dropped CPU interrupt event
97	EMC_IDMAERR	Invalid IDMA parameters
110	MDMAERREVT	XMC VBUSM error event, which may be caused by violation of MPAX protection or bus error return from destination.

113	L1P_ED	Single bit error detected during DMA accesses L1P. Single bit L1P error when DSP core executes code will trigger internal instruction fetch exception.
117	LL2_ED2	Uncorrectable error detected in LL2
119	SYS_CMPA	Memory Protection Fault for local configuration of INTC and power control
120	L1P_CMPA	memory protection fault for DSP core accesses L1P
121	L1P_DMPA	memory protection fault for DMA accesses L1P
122	L1D_CMPA	memory protection fault for DSP core accesses L1D (and other memory read finally goes through the L1D controller)
123	L1D_DMPA	memory protection fault for DMA accesses L1D
124	LL2_CMPA	memory protection fault for DSP core accesses L2
125	LL2_DMPA	memory protection fault for DMA accesses L2
126	EMC_CMPA	memory protection fault for other local configuration space between 0x01000000 - 0x01BFFFFF
127	EMC_BUSERR	Bus Error Interrupt for global configuration space between 0x01C00000 - 0x07FFFFFF

Some other non-fatal error events, such as correctable LL2 EDC error, should be routed to interrupt instead of exception.

The error events originate from or triggered by shared modules in a device are routed to CIC. Refer to “KeyStone Architecture Chip Interrupt Controller (CIC) User Guide (SPRUGW4)” for basic information of CIC.

The CIC events should be treated as exceptions are listed in following table.

Table 8. Exception events routed through CIC

Event Number	Event	Description
0	EDMACC_ERRINT	EDMA3CC1 error interrupt
2,3,4,5	EDMATC_ERRINT	EDMA3CC1 EDMA3TC0,1,2,3 error interrupt
16	EDMACC_ERRINT	EDMA3CC2 error interrupt
18,19,20,21	EDMATC_ERRINT	EDMA3CC2 EDMA3TC0,1,2,3 error interrupt
32	EDMACC_ERRINT	EDMA3CC3 error interrupt
34,35	EDMATC_ERRINT	EDMA3CC3 EDMA3TC0,1 error interrupt
99	MSMC_dedc_nc_error	Non-correctable soft error detected on SRAM read
100	MSMC_scrub_nc_error	Non-correctable soft error detected during scrub cycle
102~109	MSMC_mpf_error	Memory protection fault indicators for system master with PrivID 8~15
110	DDR3_ERR	DDR3 ECC Error Interrupt
170~173	MSMC_mpf_error	Memory protection fault indicators for system master with PrivID 4~7 **on DSP with four CorePac**

		(TCI6614, C6670…)
90, 92, 94, 96	MPU_INTD	Access violation interrupt of MPU0~3 **on C6678, C6670**
174, 180	MPU_INTD	Access violation interrupt of MPU4,5 **on C6670**
23	MPU_Combined_Address_Error	MPU0~7 reserved registers access violation interrupt combined output **on TCI6614/2**
37	MPU_Combined_PROT_Error	MPU0~7 protection violation interrupt combined output **on TCI6614/2**

Each of these exception events should only be routed to one of the CorePac. Normally, all of these events are routed to one CorePac, which is a kind of master in the system.

Following figure shows the switches in DSP core controls the exception handling.

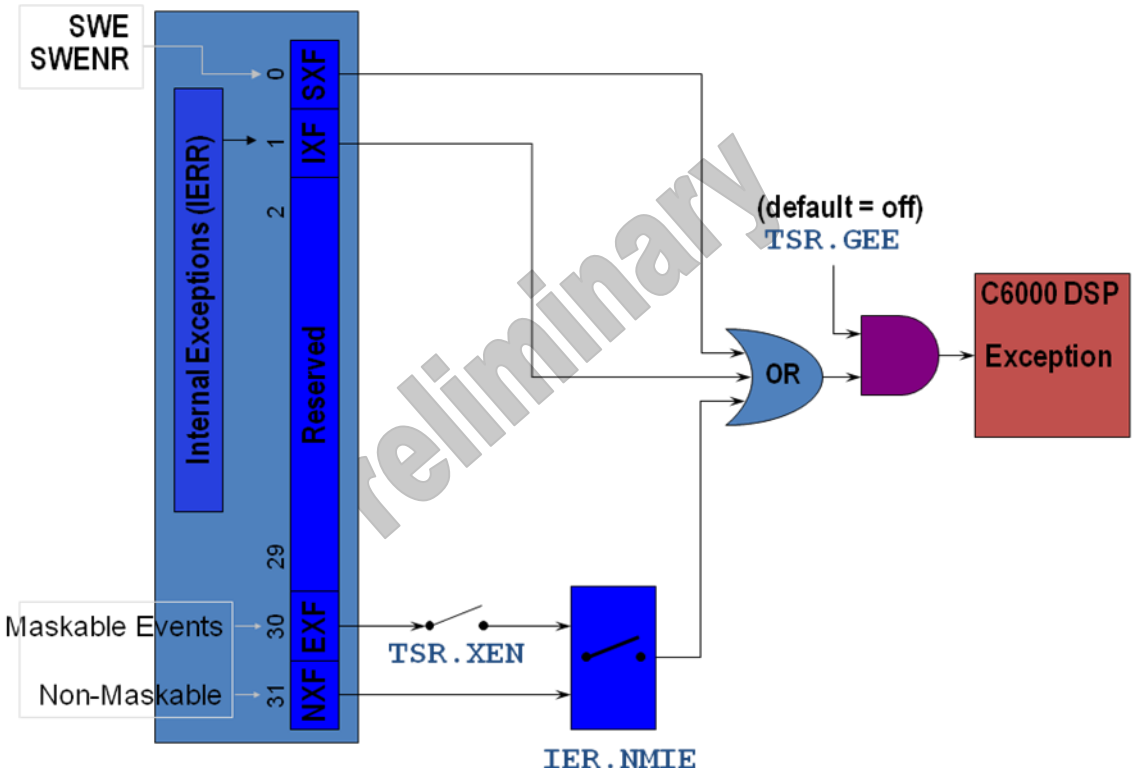


Figure 2. Exception switches

Once TSR.GEE and IER.NMIE are set by software, they can not be cleared by software, they can only be cleared by reset.

TSR.XEN can be set and cleared by software. XEN will be automatically cleared by hardware when enter the exception service routine, and will automatically restored to original state when exit exception service routine.

So, by default, in exception service routine, TSR.GEE=1, IER.NMIE=1 and TSR.XEN=0.

5.2 Exception Service Routine

Exception routine should record or report the exception cause and related information, which will be used for failure analysis.

The key information should be recorded is NRP. NRP is Return Pointer of exception, which is normally used to determine the position where the exception triggered.

In practice, the delay between the illegal operation and the NRP is about 10 to 100 DSP core cycles, which depends on many factors including the operation type, the target module which generates exception event... For example, for a write operation to a register protected by MPU, the delay includes: the delay of the write instruction from DSP core to the register; and the delay for the error event routing from the MPU to CIC and then to CorePac exception module. So, when we know the NRP, we should search backward (10~100 cycles) from that position to find the problematic operation.

However, NRP is meaningless for some exceptions, such as instruction fetch exception and illegal opcode exception. This normal happens when program jump to an invalid address, the NRP also points to an invalid address. What we really want to know is what happens before the program jump to invalid address, but this can not be deduced from the NRP. For this case, B3, A4, B4, B14 and B15 may be helpful. B3 may contain the return point of the last function called; A4 and B4 may contain the parameters of last function call; B15 is the stack pointer; B14 is the data pointer to some global variables. Refer to section “7.4 Function Structure and Calling Conventions” of “TMS320C6000 Optimizing Compiler User's Guide (SPRU187)” for more details. According to this information, we may deduce what happens before the program jumps into invalid address. Please note, B3, A4, B4 only contain these valuable information before they are modified in the last function, so they are not always useful. In practice, the possibility of the B3, A4, B4 contain valuable information is high, so it is worth checking.

The general registers can not be recorded in C code, assembly code must be written to record them. Below is an example which records B3, A4, B4, B14, B15 registers to “exception_record” and then call “Exception_service_routine”

```
       .ref Exception_service_routine
       .ref exception_record
;---------------------------------------------------------------
       .sect ".text"
;interrupt vector for NMI
NMI_ISR:
       STW         B1,*-B15[1]

       ;save some key registers when exception happens
       MVKL  exception_record,B1
       MVKH  exception_record,B1

       STW         B3, *+B1[0]
       STW         A4, *+B1[1]
       STW         B4, *+B1[2]
       STW         B14, *+B1[3]
```

```
        STW         B15, *+B1[4]

        ;jump to exception service routine
        MVKL        Exception_service_routine, B1
        MVKH        Exception_service_routine, B1
        B           B1

        LDW         *-B15[1],B1
        NOP         4
```

Other basic information should be recorded include: EFR, IERR, NTSR, TSCL/TSCH. EFR is used to determine the type of exception: internal, external or NMI. If it is internal exception, the cause of the internal exception is recorded in IERR. NTSR records the DSP core status when the exception happens. TSCL/TSCH is normally recorded to determine how long the device has run before the exception happens.

For external exceptions, the INTC and CIC flag registers should be checked to determine the cause of the exception. For each specific exception, there may be specific status registers can be checked, recorded or reported. For example, for the memory protection exception, the key information need be recorded is fault address. Refer to the module's user's guide for more details of those status or flags.

Normally, the exception service routine save these exception information into a data structure like below.

```
typedef  struct{
  volatile Uint32 MEXPFLAG[4];  /*copy of the MEXPFLAG0..3 registers */
  volatile Uint32 CIC_STATUS[6];  /*copy of the CIC status registers */
  Exception_Info info;
} External_Exception_Status;

typedef  union  {
  volatile Uint32 IERR; /*copy of the IERR register */
  External_Exception_Status ext_sts;
} Exception_Status;

typedef struct {
  volatile Uint32 B3; /*copy of B3 register (return pointer of caller) */
  volatile Uint32 A4; /*copy of the A4 register (first input parameter of caller)*/
  volatile Uint32 B4; /*copy of the B4 register (second input parameter of caller)*/
  volatile Uint32 B14; /*copy of the B14 register (data pointer)*/
  volatile Uint32 B15; /*copy of the B15 register (stack pointer)*/
  volatile Uint32 TSCL; /*copy of the TSCL register (time stamp)*/
  volatile Uint32 TSCH; /*copy of the TSCH register (time stamp)*/
  volatile Uint32 NTSR; /*copy of the NTSR register */
  volatile Uint32 NRP; /*copy of the NRP register */
  volatile Uint32 EFR; /*copy of the EFR register */
  Exception_Status status;
} Exception_Record;
```

The information in the data structure can be transferred to a host in the exception service routine, or be dumped when a user try to analyze the error.

Normally, the error handled by exception service routine are fatal error. User should not expect return from exception service routine. Additionally, it is not always possible to safely return from the exception handling routine. Conditions that can prevent a safe return from exceptions include:

1. SPLOOPs that are terminated by an exception cannot be resumed correctly. The SPLX bit in NTSR should be verified to be 0 before returning.
2. Exceptions that occur when interrupts are blocked cannot be resumed correctly. The IB bit in NTSR should be verified to be 0 before returning.
3. Exceptions that occur at any point in the code that cannot be interrupted safely (for example, a tight loop containing multiple assignments) cannot be safely returned to. The compiler will normally disable interrupts at these points in the program; check the GIE bit in NTSR to be 1 to verify that this condition is met.
4. NRP is not in valid address space.

So, normally, exception service routine is ended with a while(1) loop.

By default, in exception service routine, TSR.GEE=1, IER.NMIE=1 and TSR.XEN=0. That is, NMI and internal exception is enabled during exception service routine.

When an enabled exception happens in the first exception service routine, then the reset vector is used when redirecting program execution to service the second exception. In this case, NTSR and NRP are left unchanged. TSR is copied to ITSR and the current PC is copied to IRP. TSR is set to the default exception processing value and the NMIE bit in IER is cleared by hardware in this case preventing any further external exceptions.

Normally, the reset vector in interrupt service table is just a jump to the beginning of program, such as, _c_int00. For this case, a nested exception will actually restart the program. However, this is not expected by most users, we normally hope program stops at the end of exception service routine when exception happens. To avoid this, an additional service routine for nested exception should be added, and user should modify the reset vector to jump to the nested exception service routine. On KeyStone device, boot loader does not rely on the reset vector to start program, so modifying the reset vector does not affect the boot loading.

6 STM usage

The System Trace Module(STM) provide a system view debug method for the complex system. The design idea for the STM is to provide a hardware method to record the system information and get the whole picture under the multi-core or multi-task system.

There are some common problems for the real-time system debug:

1. For real-time application, or test an integrated complex system, it’s difficult to halt any running task without affecting the application;
2. The traditional debug method is the printf , it’s too intrusive, for example it’s not good way to debug the time critical task(ISR);

3. For shared resource conflict debug, it's difficult for the software to get the actual access data and conflict time window, which is the most common issue in the applications.

Following figure shows the System Trace Module architecture of KeyStone devices. The software message is generated by the DSP core or EDMA, in the application, the software or DMA write the STM messages to the dedicated memory address, which route the message to the STM module, when the software or the DMA read these memory address, it's always return 0.

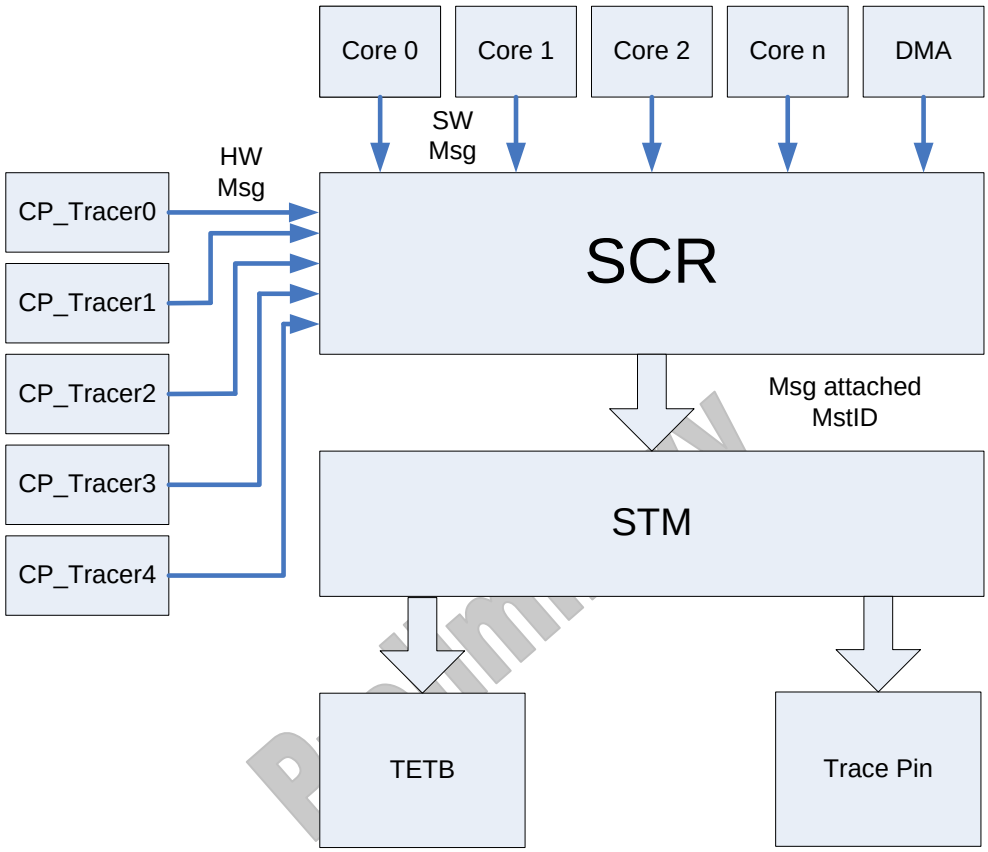


Figure 3. STM architecture on Keystone Devices

The hardware message on keystone device is generated from the CP_Tracer(Common Platform Tracer), which is widely used on TI digital devices, it monitors the CBA bus event from master to the slave, and the software configure the CP_Tracer to collect the interested CBA bus event, then the CP_Tracer generate the hardware message to the STM module. Following figure shows the CP_Tracer architecture of Keystone devices. For the CP_Tracer detail information, please refer to the debug and trace user guide.

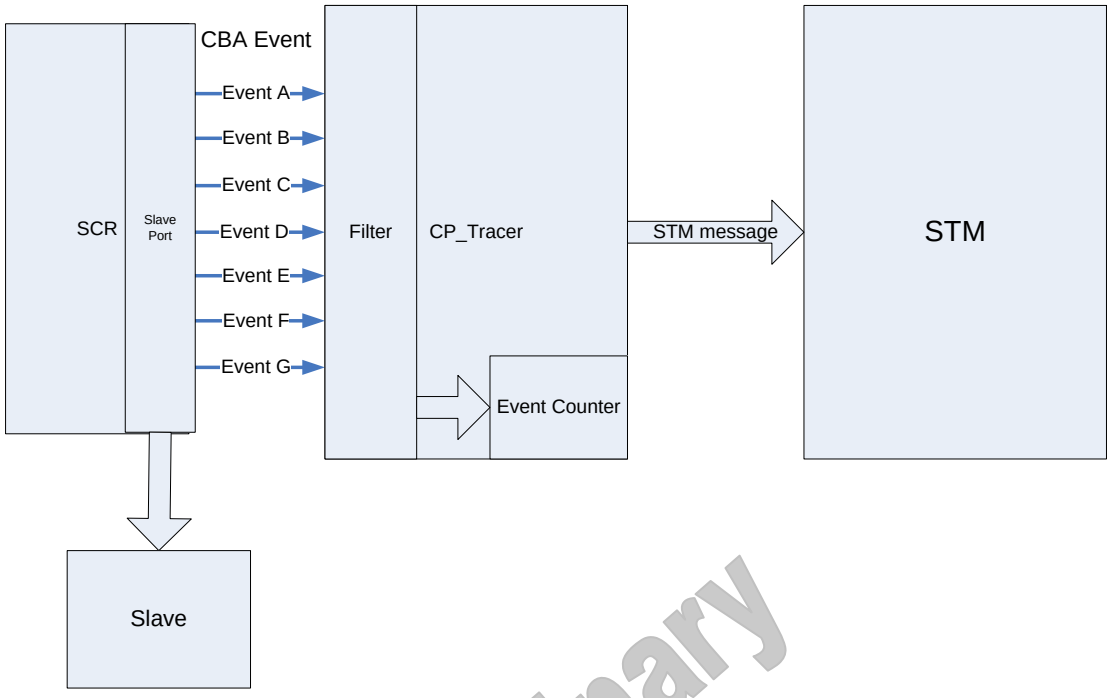


Figure 4. CP_Tracer Architecture of KeyStone devices

The STM module on Keystone devices is consist of the following parts, following figure shows the STM block diagram:

1. Software message MstID enable and mask which control the authorized master send the software message to the STM Channel;
2. Hardware message MstID enable, on Keystone device, only CP_Tracer can send the hardware message to the STM, and the MstID is fixed to 128;
3. STM software channel, there are 1M memory address reserved for STM software channel, the software or DMA write data to these address will send the request to the STM module, the 1M address space is partitioned to 256x4K range, each 4K range corresponds to one channel;
4. STM message interleaving, the STM module will interleave the software message and hardware message into the continuous message stream, it provides the whole picture for the hardware and software transactions, which is very useful to system analysis;
5. STM message export module, there are 2 modules can export the STM message. One is the 32K TETB and another is the Trace pins to the emulator. The TETB can be dumped on real-time system without emulator support, which is very useful for the field debug. The Trace pins will collect the trace data to the CCS, it's easy way to use the STM messages;

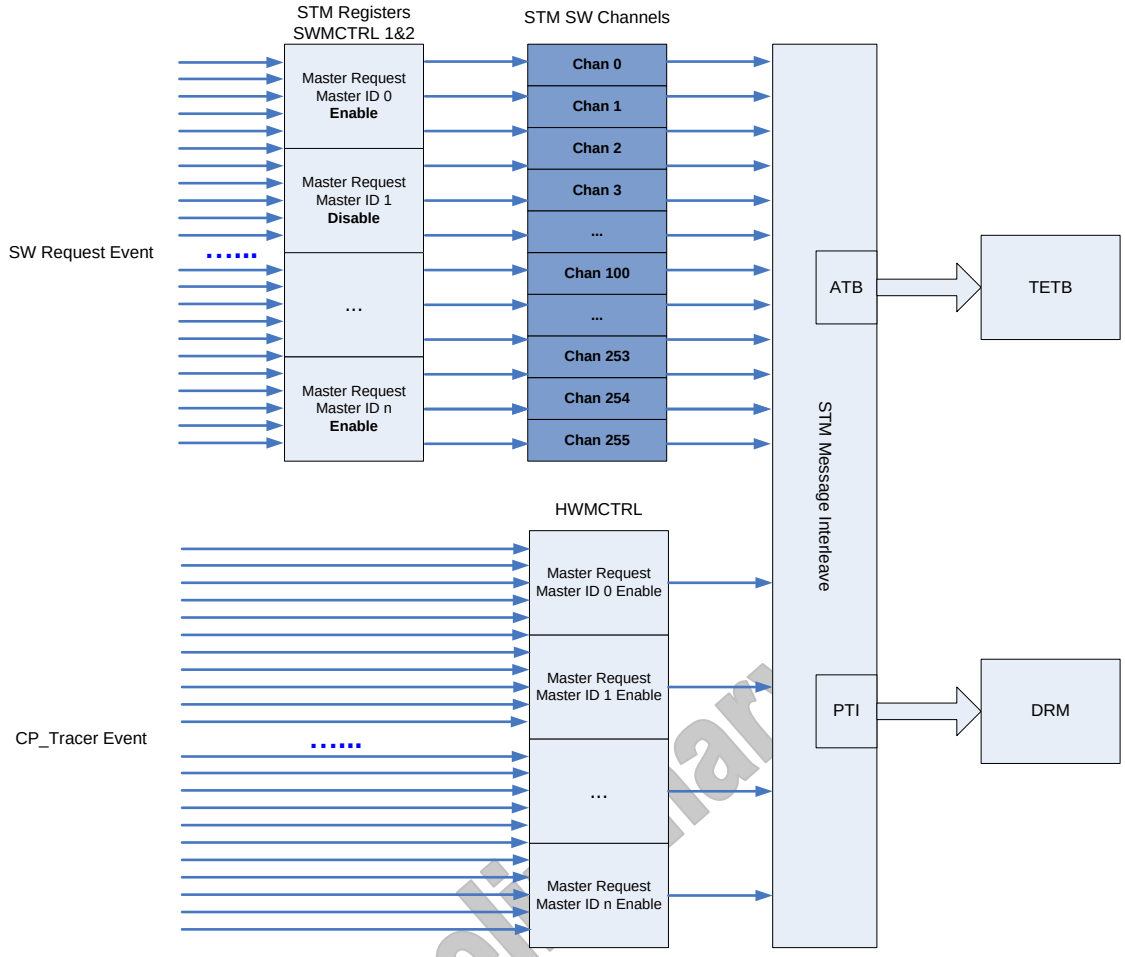


Figure 5. STM block diagram

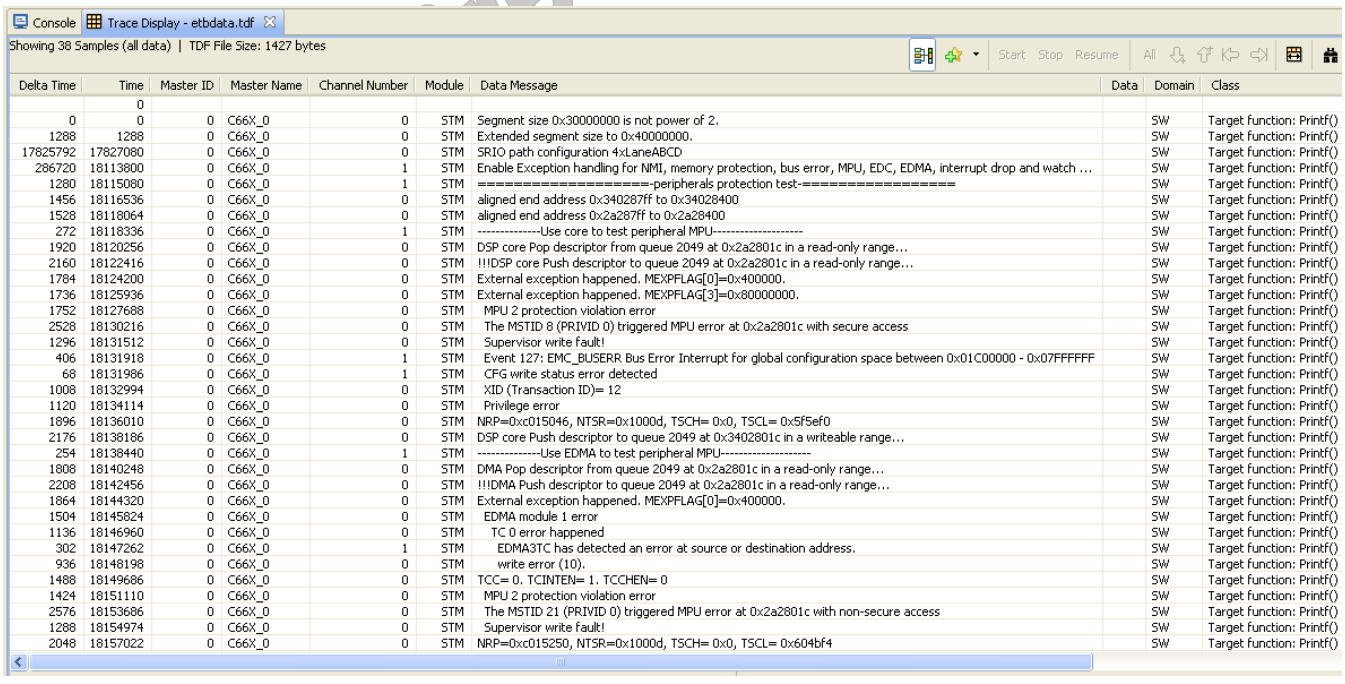
Each STM software message channels is allocated 4K address space, and it's dived into two part. The upper half address space of one channel is for STM message without timestamp; the rest address space is for the STM message with the timestamp. It means when the upper address space is accessed, no timestamp is attached to the generated STM software message, if the other half address space is accessed, the timestamp will be attached to the generated STM software messages, following figure shows the software message channel structure.

20-bit STM address		STM Messages			
		Data	Time Stamp	Channel	Size
0x00000	0x007FF	x		0	4K bytes
0x00800	0x00FFF	x	x		
0x01000	0x017FF	x		1	4K bytes
0x01800	0x01FFF	x	x		
….		….			
0xFF000	0xFF7FF	x		255	4K bytes
0xFF800	0xFFFFF	x	x		

Figure 6. STM software message structure

In the example code, the STMLib and ETBLib are used to implement the printf for debug. If the application wants to store the debug information to the ETB buffer, only need to define the STM_DEBUG at KeyStone_common.h and add the CTOOLs lib to the project, the ctools lib can be downloaded at the following link https://gforge.ti.com/gf/project/ctoolslib/frs/.

If the STM_DEBUG is used, after test complete, the test result will be stored in the Robust_etbdata.bin, and the project also provided the bin2tdf_stm.bat to convert the bin file to the tdf file which can be parsed by the CCS debug analysis tools. Following Figure shows the trace analyzer in CCS.


Console | Trace Display - etbdata.tdf

Showing 38 Samples (all data) | TDF File Size: 1427 bytes

Start Stop Resume All

Delta Time	Time	Master ID	Master Name	Channel Number	Module	Data Message	Data	Domain	Class
	0								
0	0	0	C66X_0	0	STM	Segment size 0x30000000 is not power of 2.		SW	Target function: Printf()
1288	1288	0	C66X_0	0	STM	Extended segment size to 0x40000000.		SW	Target function: Printf()
17825792	17827080	0	C66X_0	0	STM	SRIO path configuration 4xLaneABCD		SW	Target function: Printf()
286720	18113800	0	C66X_0	1	STM	Enable Exception handling for NMI, memory protection, bus error, MPU, EDC, EDMA, interrupt drop and watch …		SW	Target function: Printf()
1280	18115080	0	C66X_0	1	STM	==========-peripherals protection test-==========		SW	Target function: Printf()
1456	18116536	0	C66X_0	0	STM	aligned end address 0x340287ff to 0x34028400		SW	Target function: Printf()
1528	18118064	0	C66X_0	0	STM	aligned end address 0x2a287ff to 0x2a28400		SW	Target function: Printf()
272	18118336	0	C66X_0	1	STM	----------Use core to test peripheral MPU----------		SW	Target function: Printf()
1920	18120256	0	C66X_0	0	STM	DSP core Pop descriptor from queue 2049 at 0x2a2801c in a read-only range…		SW	Target function: Printf()
2160	18122416	0	C66X_0	0	STM	!!!DSP core Push descriptor to queue 2049 at 0x2a2801c in a read-only range…		SW	Target function: Printf()
1784	18124200	0	C66X_0	0	STM	External exception happened. MEXPFLAG[0]=0x400000.		SW	Target function: Printf()
1736	18125936	0	C66X_0	0	STM	External exception happened. MEXPFLAG[3]=0x80000000.		SW	Target function: Printf()
1752	18127688	0	C66X_0	0	STM	MPU 2 protection violation error		SW	Target function: Printf()
2528	18130216	0	C66X_0	0	STM	The MSTID 8 (PRIVID 0) triggered MPU error at 0x2a2801c with secure access		SW	Target function: Printf()
1296	18131512	0	C66X_0	0	STM	Supervisor write fault!		SW	Target function: Printf()
406	18131918	0	C66X_0	1	STM	Event 127: EMC_BUSERR Bus Error Interrupt for global configuration space between 0x01C00000 - 0x07FFFFFF		SW	Target function: Printf()
68	18131986	0	C66X_0	1	STM	CFG write status error detected		SW	Target function: Printf()
1008	18132994	0	C66X_0	0	STM	XID (Transaction ID)= 12		SW	Target function: Printf()
1120	18134114	0	C66X_0	0	STM	Privilege error		SW	Target function: Printf()
1896	18136010	0	C66X_0	0	STM	NRP=0xc015046, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x5f5ef0		SW	Target function: Printf()
2176	18138186	0	C66X_0	0	STM	DSP core Push descriptor to queue 2049 at 0x3402801c in a writeable range…		SW	Target function: Printf()
254	18138440	0	C66X_0	1	STM	----------Use EDMA to test peripheral MPU----------		SW	Target function: Printf()
1808	18140248	0	C66X_0	0	STM	DMA Pop descriptor from queue 2049 at 0x2a2801c in a read-only range…		SW	Target function: Printf()
2208	18142456	0	C66X_0	0	STM	!!!DMA Push descriptor to queue 2049 at 0x2a2801c in a read-only range…		SW	Target function: Printf()
1864	18144320	0	C66X_0	0	STM	External exception happened. MEXPFLAG[0]=0x400000.		SW	Target function: Printf()
1504	18145824	0	C66X_0	0	STM	EDMA module 1 error		SW	Target function: Printf()
1136	18146960	0	C66X_0	0	STM	TC 0 error happened		SW	Target function: Printf()
302	18147262	0	C66X_0	1	STM	EDMA3TC has detected an error at source or destination address.		SW	Target function: Printf()
936	18148198	0	C66X_0	0	STM	write error (10).		SW	Target function: Printf()
1488	18149686	0	C66X_0	0	STM	TCC= 0. TCINTEN= 1. TCCHEN= 0		SW	Target function: Printf()
1424	18151110	0	C66X_0	0	STM	MPU 2 protection violation error		SW	Target function: Printf()
2576	18153686	0	C66X_0	0	STM	The MSTID 21 (PRIVID 0) triggered MPU error at 0x2a2801c with non-secure access		SW	Target function: Printf()
1288	18154974	0	C66X_0	0	STM	Supervisor write fault!		SW	Target function: Printf()
2048	18157022	0	C66X_0	0	STM	NRP=0xc015250, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x604bf4		SW	Target function: Printf()

Figure 7. STM Trace example

7 Example Project

The example codes along with this application note can be run on TCI6614 EVM, C6670 EVM and C6678 EVM.

The directory structure of the projects is as below:

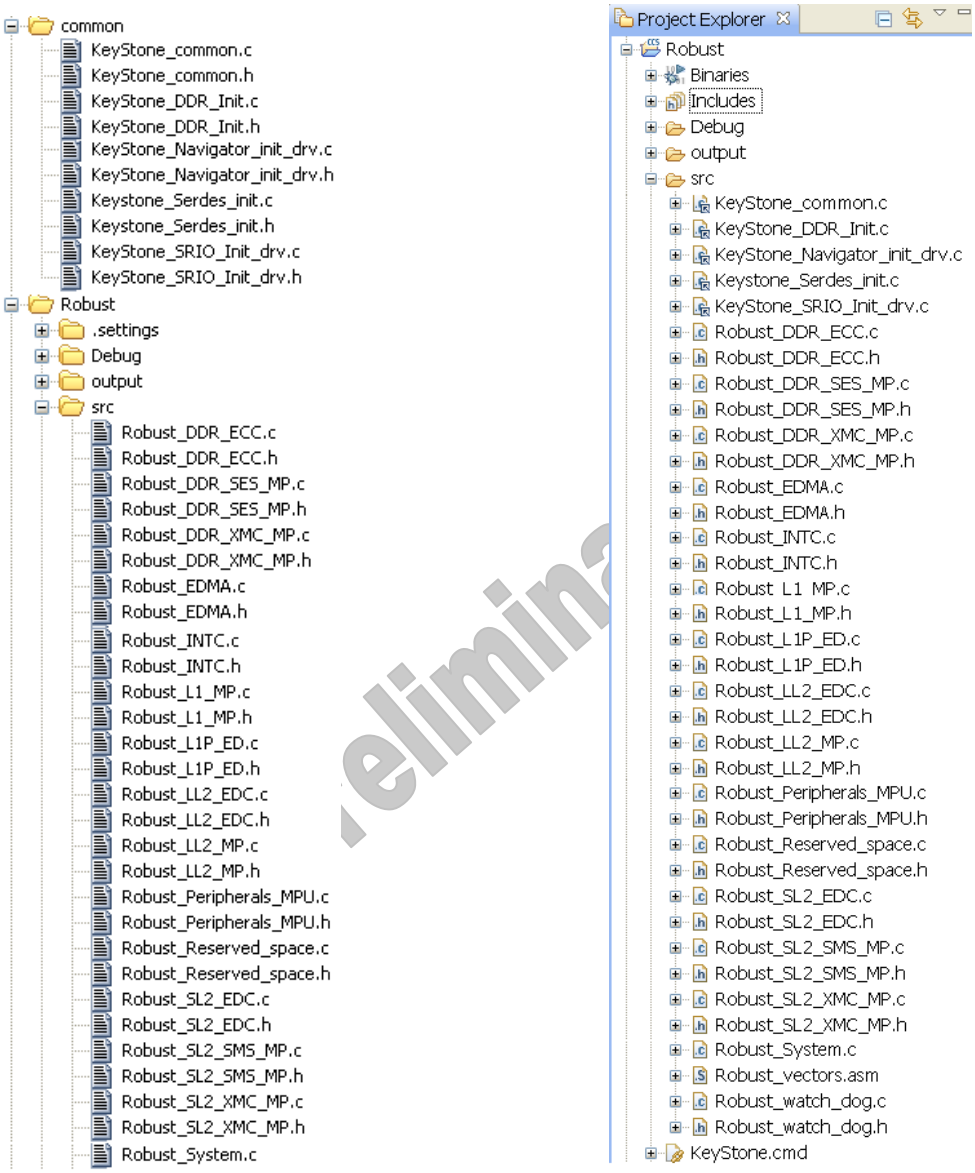


Figure 8. Directory structure of example codes

The “common” folder includes some common code such as DDR initialization, and DMA, timer, Multicore navigator, SRIO driver… The initialization code of memory protection, EDC and exception handling code are in the KeyStone_common.c.

Each .C file in “src” folder includes the code for a test case. The main function is in the “Robust_System.c”. At the beginning of the “Robust_System.c”, there is a set of macro switches as below, each of them can be used to enable or disable one test case.

```
#define L1D_MP_TEST                1
#define L1P_MP_TEST                1
#define LL2_MP_TEST                1
#define XMC_SL2_MP_TEST            1
#define XMC_DDR_MP_TEST            1
#define SMS_SL2_MP_TEST            1
#define SES_DDR_MP_TEST            1
#define PERIPHERAL_MP_TEST         1
#define L1P_ED_TEST                1
#define LL2_EDC_TEST               1
#define SL2_EDC_TEST               1
#define DDR_ECC_TEST               1            /*only for EVM with ECC memory*/
#define EDMA_ERROR_TEST            1
#define WATCH_DOG_TEST             1
#define RESERVED_SPACE_TEST        1
#define INT_DROP_TEST              1
```

If you enable multiple of the test cases, they should be executed one by one at a time. It is possible the test flow stops after one test case because the program can not safely return from the exception service routine triggered by that test. If this happens, you should disable that test case and run the test again to try other test cases.

The steps to run the examples on EVM are:

1. unzip the package of the example project into your workspace folder.
2. import the project into your workspace.
3. Rebuild the project if you modify any code, you may need change the CSL including path in the compiler option.
4. Load the program into core 0 of a DSP, and then run it.
5. Check the stdout window of CCS for test result.

Below is the test result on TCI6614 EVM.

```
Initialize DSP main clock = 122.88MHzx236/29 = 999MHz
Initialize DDR speed = 66.667x20/1= 1333.3
SRIO path configuration 4xLaneABCD
Enable Exception handling...
==================-L1D memory protection test-==================
!!!DMA write to L1D at 0x00F00000...
External exception happened. MEXPFLAG[0]=0x800000.
External exception happened. MEXPFLAG[3]=0x8000000.
  EDMA module 0 error
    TC 0 error happened
      EDMA3TC has detected an error at source or destination address.
      write error (10). TCC= 0. TCINTEN= 1. TCCHEN= 0
  Event 123: DMC_DMPA DMA memory protection fault for L1D
  memory protection exception caused by master with ID 0 at 0xf00000
  Supervisor Write violation
NRP=0xc0124b8, NTSR=0x1000d, TSCH= 0x0, TSCL= 0xc594e5
 B3=0xc012468, A4=0x0, B4= 0x1, B14= 0x839fc0, B15= 0x838fb0

!!!DSP core write to L1D at 0x00F00100...
```

```
External exception happened. MEXPFLAG[3]=0x4000000.
  Event 122: DMC_CMPA CPU memory protection fault for L1D (and other memory read
finally goes through the L1D controller)
  memory protection exception caused by local access at 0xf00100
  Supervisor Write violation
NRP=0xc0124fa, NTSR=0x1000d, TSCH= 0x0, TSCL= 0xc6ff1d
 B3=0xc0124e0, A4=0x2b, B4= 0xbfbfbfbf, B14= 0x839fc0, B15= 0x838fb0

!!!DSP core write to CGEM_regs->L1DMPPA[0] register at 0x184ae40 without unlocking...
External exception happened. MEXPFLAG[3]=0x4000000.
  Event 122: DMC_CMPA CPU memory protection fault for L1D (and other memory read
finally goes through the L1D controller)
  memory protection exception caused by local access at 0x184ae40
  Supervisor Write violation
NRP=0xc012548, NTSR=0x1000d, TSCH= 0x0, TSCL= 0xc80c2a
 B3=0xc012528, A4=0x57, B4= 0x184ae40, B14= 0x839fc0, B15= 0x838fb0
====================-L1P memory protection test-==================
!!!DMA write to L1D at 0x00E00000...
External exception happened. MEXPFLAG[0]=0x800000.
External exception happened. MEXPFLAG[3]=0x2000000.
  EDMA module 0 error
    TC 0 error happened
      EDMA3TC has detected an error at source or destination address.
      write error (10). TCC= 0. TCINTEN= 1. TCCHEN= 0
  Event 121: PMC_DMPA DMA memory protection fault for L1P
  memory protection exception caused by master with ID 0 at 0xe00000
  Supervisor Write violation
NRP=0xc0123b8, NTSR=0x1000d, TSCH= 0x0, TSCL= 0xc8e9a2
 B3=0xc012368, A4=0x0, B4= 0x1, B14= 0x839fc0, B15= 0x838fb0

!!!DSP core write to CGEM_regs->L1PMPPA[0] register at 0x184a640 without unlocking...
External exception happened. MEXPFLAG[3]=0x1000000.
  Event 120: PMC_CMPA CPU memory protection fault for L1P
  memory protection exception caused by local access at 0x184a640
  Supervisor Write violation
NRP=0xc01240c, NTSR=0x1000d, TSCH= 0x0, TSCL= 0xca8754
 B3=0xc0123f0, A4=0x57, B4= 0x1800000, B14= 0x839fc0, B15= 0x838fb0
====================-LL2 memory protection test-==================
EDMA write to 0x10828000, which is allowed for external access...

!!!EDMA write to 0x10820000, which is protected against external access...
External exception happened. MEXPFLAG[0]=0x800000.
External exception happened. MEXPFLAG[3]=0x20000000.
  EDMA module 0 error
    TC 0 error happened
      EDMA3TC has detected an error at source or destination address.
      write error (10). TCC= 0. TCINTEN= 1. TCCHEN= 0
  Event 125: UMC_DMPA DMA memory protection fault for L2
  memory protection exception caused by master with ID 0 at 0x820000
  Supervisor Write violation
NRP=0xc010dee, NTSR=0x1000d, TSCH= 0x0, TSCL= 0xd89b30
 B3=0xc010d9a, A4=0x0, B4= 0x1, B14= 0x839fc0, B15= 0x838fb0

!!!DSP core write to CGEM_regs->L2MPPA[0] register at 0x184a200 without unlocking...
External exception happened. MEXPFLAG[3]=0x10000000.
  Event 124: UMC_CMPA CPU memory protection fault for L2
  memory protection exception caused by local access at 0x184a200
  Supervisor Write violation
NRP=0xc010e48, NTSR=0x1000d, TSCH= 0x0, TSCL= 0xda387e
 B3=0xc010e28, A4=0x56, B4= 0xffffffff, B14= 0x839fc0, B15= 0x838fb0
====================-XMC SL2 memory protection test-==============
```

```
DSP core write to 0xc110400 which is write-only...
DSP core read from 0xc12c808 which is read-only...

!!!DSP core read from 0xc11883f which is write-only...
External exception happened. MEXPFLAG[3]=0x4000000.
  Event 122: DMC_CMPA CPU memory protection fault for L1D (and other memory read
finally goes through the L1D controller)
  memory protection exception caused by local access at 0xc11883f
  Supervisor Read violation
NRP=0xc01204c, NTSR=0x1000d, TSCH= 0x0, TSCL= 0xeaa097
 B3=0xc012034, A4=0x38, B4= 0x20, B14= 0x839fc0, B15= 0x838fb0

!!!DSP core write to 0xc12d44c which is read-only...
External exception happened. MEXPFLAG[3]=0x4000.
  Event 110: MDMAERREVT XMC VBUSM error event
  memory protection exception caused by local access at 0xc12d440
  Supervisor Write violation
NRP=0xc01208a, NTSR=0x1000d, TSCH= 0x0, TSCL= 0xeb9614
 B3=0xc012070, A4=0x36, B4= 0xc12d44c, B14= 0x839fc0, B15= 0x838fb0

!!!DSP core write to XMC_regs->XMPAX[0].XMPAXL register at 0x8000000 without
unlocking...
External exception happened. MEXPFLAG[3]=0x4000.
  Event 110: MDMAERREVT XMC VBUSM error event
  memory protection exception caused by local access at 0x8000000
  Supervisor Write violation
NRP=0xc0120c8, NTSR=0x1000d, TSCH= 0x0, TSCL= 0xec9fcf
 B3=0xc0120ac, A4=0x5b, B4= 0x8000000, B14= 0x839fc0, B15= 0x838fb0
====================-XMC DDR memory protection test-==============
DSP core write to 0x81000400 which is write-only...
DSP core read from 0x81020808 which is read-only...

!!!DSP core read from 0x8100883f which is write-only...
External exception happened. MEXPFLAG[3]=0x4000000.
  Event 122: DMC_CMPA CPU memory protection fault for L1D (and other memory read
finally goes through the L1D controller)
  memory protection exception caused by local access at 0x8100883f
  Supervisor Read violation
NRP=0xc0122b4, NTSR=0x1000d, TSCH= 0x0, TSCL= 0xfd0700
 B3=0xc0122a0, A4=0x39, B4= 0x20, B14= 0x839fc0, B15= 0x838fb0

!!!DSP core write to 0x8102144c which is read-only...
External exception happened. MEXPFLAG[3]=0x4000.
  Event 110: MDMAERREVT XMC VBUSM error event
  memory protection exception caused by local access at 0x81021440
  Supervisor Write violation
NRP=0xc0122f2, NTSR=0x1000d, TSCH= 0x0, TSCL= 0xfdfcf9
 B3=0xc0122d8, A4=0x37, B4= 0x8102144c, B14= 0x839fc0, B15= 0x838fb0
====================-SMS SL2 memory protection test-==============
EDMA copy from 0xc128000 (read-only) to 0xc100000 (write-only)...

!!!EDMA copy from 0xc108000 (write-only) to 0xc129000 (read-only)...
External exception happened. MEXPFLAG[0]=0x800400.
  EDMA module 0 error
    TC 0 error happened
      EDMA3TC has detected an error at source or destination address.
      read error (2). TCC= 0. TCINTEN= 1. TCCHEN= 0
  Event 10 : MSMC_mpf_error_n MSMC Memory protection fault indicators for local CorePac
  memory protection exception caused by master 16 (PrivID= 0) at address 0xc108000
NRP=0xc00cc8c, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x10df64d
 B3=0xc00cc66, A4=0x0, B4= 0x1, B14= 0x839fc0, B15= 0x838f68
```

```
!!!DSP core write to msmcRegs->SMS_MPAX_PER_PRIVID[0].SMS[0].MPAXL register at
0xbc00200 without unlocking...
External exception happened. MEXPFLAG[3]=0x4000.
  Event 110: MDMAERREVT XMC VBUSM error event
  MDMA write status error detected
  XID (Transaction ID)= 8
  Privilege error
NRP=0xc00cd02, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x10f889d
 B3=0xc00ccd4, A4=0x6f, B4= 0xbc00200, B14= 0x839fc0, B15= 0x838f68
--------------------SRIO SMS SL2 memory protection test---------------
SRIO copy from 0xc12a000 (read-only) to 0xc108000 (write-only)...

!!!SRIO copy from 0xc12a000 (read-only) to 0xc129000 (read-only)...
External exception happened. MEXPFLAG[0]=0x800000.
  memory protection exception caused by master 54 (PrivID= 9) at address 0xc129000
NRP=0xc00cfc6, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x110be59
 B3=0xc00cf92, A4=0xc12a020, B4= 0x3, B14= 0x839fc0, B15= 0x838f68
====================-SES DDR memory protection test-==============
EDMA copy from 0x81024000 (read-only) to 0x81010000 (write-only)...

!!!EDMA copy from 0x81018000 (write-only) to 0x81025000 (read-only)...
External exception happened. MEXPFLAG[0]=0x800400.
  EDMA module 0 error
    TC 0 error happened
      EDMA3TC has detected an error at source or destination address.
      read error (1). TCC= 0. TCINTEN= 1. TCCHEN= 0
  Event 10 : MSMC_mpf_error_n MSMC Memory protection fault indicators for local CorePac
  memory protection exception caused by master 16 (PrivID= 0) at address 0x81018000
NRP=0xc00c54c, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x120c9d0
 B3=0xc00c526, A4=0x0, B4= 0x1, B14= 0x839fc0, B15= 0x838f68

!!!DSP core write to msmcRegs->SES_MPAX_PER_PRIVID[0].SES[0].MPAXL register at
0xbc00600 without unlocking...
External exception happened. MEXPFLAG[3]=0x4000.
  Event 110: MDMAERREVT XMC VBUSM error event
  MDMA write status error detected
  XID (Transaction ID)= 12
  Privilege error
NRP=0xc00c5c2, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x1225bbf
 B3=0xc00c594, A4=0x6f, B4= 0xbc00600, B14= 0x839fc0, B15= 0x838f68
--------------------SRIO SES DDR3 memory protection test---------------
SRIO copy from 0x81026000 (read-only) to 0x81018000 (write-only)...

!!!SRIO copy from 0x81026000 (read-only) to 0x81025000 (read-only)...
External exception happened. MEXPFLAG[0]=0x800000.
  memory protection exception caused by master 54 (PrivID= 9) at address 0x81025000
NRP=0xc00c874, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x1239329
 B3=0xc00c852, A4=0x81026020, B4= 0x4, B14= 0x839fc0, B15= 0x838f68
====================-peripherals protection test-=================
!!!Write to PLL register at 0x02310110 which is protected by MPU0...
External exception happened. MEXPFLAG[0]=0x800000.
External exception happened. MEXPFLAG[3]=0x80000000.
  MPU 0 protection violation error
  The MSTID 8 (PRIVID 0) triggered MPU error at 0x2310110 with secure access
  Supervisor write fault!
  Event 127: EMC_BUSERR Bus Error Interrupt for global configuration space between
0x01C00000 - 0x07FFFFFF
  CFG write status error detected
  XID (Transaction ID)= 13
  Privilege error
```

```
NRP=0xc013664, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x124aaed
 B3=0xc013644, A4=0x45, B4= 0x2310110, B14= 0x839fc0, B15= 0x838fb0
MPU 2 setup 11 of 16 ranges.

DSP core Pop descriptor from queue 2049 at 0x2a2801c in a read-only range...

!!!DSP core Push descriptor to queue 2049 at 0x2a2801c in a read-only range...
External exception happened. MEXPFLAG[0]=0x800000.
External exception happened. MEXPFLAG[3]=0x80000000.
  MPU 2 protection violation error
  The MSTID 8 (PRIVID 0) triggered MPU error at 0x2a2801c with secure access
  Supervisor write fault!
  Event 127: EMC_BUSERR Bus Error Interrupt for global configuration space between
0x01C00000 - 0x07FFFFFF
  CFG write status error detected
  XID (Transaction ID)= 15
  Privilege error
NRP=0xc013592, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x126ad1a
 B3=0xc013564, A4=0x50, B4= 0xc133101, B14= 0x839fc0, B15= 0x838f98

DSP core Push descriptor to queue 2049 at 0x3402801c in a writeable range...
====================L1P ED test=================================
!!!manually generate one bit error in L1P cache for function at 0x800760, and then
execute it...
internal excpetion happened. IERR=0x9.
  Instruction fetch exception
  Opcode exception
  L1P Cache parity check error caused by program fetch at address 0x800760
NRP=0x800742, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x1289879
 B3=0xc012d74, A4=0x1846408, B4= 0x1840024, B14= 0x839fc0, B15= 0x838fa8

!!!manually generate one bit error in L1P cache at 0xe00760, and then read it by DMA...
External exception happened. MEXPFLAG[3]=0x20000.
  Event 113: PMC_ED Single bit error detected during DMA read
  L1P RAM parity check error caused by DMA at address 0xe00760
NRP=0xc012dda, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x129bb2e
 B3=0xc012dd0, A4=0xe00740, B4= 0x40, B14= 0x839fc0, B15= 0x838fa8
====================-LL2 EDC test-================================
--------------------LL2 data EDC test----------------------------
!!!manually generate 3 bit error in data at 0x83a240, and then read it...
External exception happened. MEXPFLAG[3]=0x200000.
  Event 117: UMC_ED2 Uncorrected bit error detected
  LL2 EDC error (non-correctable) at address 0x83a240 caused by L1D access.
  total non-correctable error number= 1, total correctable error number= 0.
NRP=0xc00d9ec, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x12acd22
 B3=0xc00d9d4, A4=0x1846008, B4= 0xcc, B14= 0x839fc0, B15= 0x838fa8

!!!manually generate 2 bit error in data at 0x83a240, and then read it...
External exception happened. MEXPFLAG[3]=0x200000.
  Event 117: UMC_ED2 Uncorrected bit error detected
  LL2 EDC error (non-correctable) at address 0x83a240 caused by L1D access.
  total non-correctable error number= 2, total correctable error number= 0.
NRP=0xc00da50, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x12c0d5d
 B3=0xc00da38, A4=0x1846008, B4= 0x1, B14= 0x839fc0, B15= 0x838fa8

!!!manually generate one bit error in data at 0x83a240, and then read it...
External exception happened. MEXPFLAG[3]=0x200000.
  Event 117: UMC_ED2 Uncorrected bit error detected
  LL2 EDC error (non-correctable) at address 0x83a240 caused by L1D access.
  total non-correctable error number= 3, total correctable error number= 0.
NRP=0xc00daa8, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x12d4e08
```

```
 B3=0xc00da98, A4=0x1846008, B4= 0x6c, B14= 0x839fc0, B15= 0x838fa8

!!!scrub the corrupted data to fix the error...
  LL2 EDC (correctable) at bit 102 of address 0x83a240 caused by DMA access.
  total non-correctable error number= 3, total correctable error number= 1.
IRP= 0xc00dae2, ITSR= 0xd. TSCH= 0x0, TSCL= 0x12e5f6f
read the data again...(no error happens again)
--------------------LL2 code EDC test----------------------------
!!!manually generate one bit error in a function at 0x800660, and then execute the
function...
  LL2 EDC (correctable) at bit 13 of address 0x800660 caused by L1P access.
  total non-correctable error number= 3, total correctable error number= 3.
IRP= 0x800646, ITSR= 0xd. TSCH= 0x0, TSCL= 0x12f588e

one bit error was corrected with previous execution. Execute the function again...(no
error happens again)
====================-SL2 EDC test-================================
!!!manually generate one bit error in a function at 0xc016ec0, and then execute the
function...
SL2 Correctable error occurred at bit 13 of address 0xc016ec0 by PrivID 0 (from C66x
CorePacs)
IRP= 0xc016eb8, ITSR= 0xd. TSCH= 0x0, TSCL= 0x1306b47

one bit error was corrected with previous execution. Execute the function again...(no
error happens again)

!!!manually generate one bit error data at 0xc136000, and then read it...
SL2 Correctable error occurred at bit 8 of address 0xc136000 by PrivID 0 (from C66x
CorePacs)
IRP= 0xc00f7c4, ITSR= 0xd. TSCH= 0x0, TSCL= 0x131358d

!!!manually generate one bit error data at 0xe0000000, and then read it...
SL2 Correctable error occurred at bit 8 of address 0xc136000 by PrivID 0 (from C66x
CorePacs)
IRP= 0xc00f7c6, ITSR= 0xd. TSCH= 0x0, TSCL= 0x131faeb
====================-DDR ECC test-================================
!!!write one byte to 0x88000001 in DDR ECC range ...
External exception happened. MEXPFLAG[0]=0x800000.
External exception happened. MEXPFLAG[3]=0x4000.
  DDR ECC error happened during read write
  Event 110: MDMAERREVT XMC VBUSM error event
  MDMA write status error detected
  XID (Transaction ID)= 7
  Data error
NRP=0xc014478, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x1330e4b
 B3=0xc014448, A4=0x36, B4= 0x88000001, B14= 0x839fc0, B15= 0x838fb0
====================-EDMA Error test-============================
!!!start a NULL transfer with EDMA CC0 Channel 0, to trigger event miss error...
External exception happened. MEXPFLAG[0]=0x800000.
  EDMA module 0 error
    EDMA Channel 0 event missed.
NRP=0xc0133b8, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x1342b69
 B3=0xc01329c, A4=0x1, B4= 0x1, B14= 0x839fc0, B15= 0x838fb0

!!!start a transfer to address 0 with EDMA CC1 Channel 2, to trigger EDMA bus error...
External exception happened. MEXPFLAG[0]=0x800000.
  EDMA module 1 error
    TC 2 error happened
      EDMA3TC has detected an error at source or destination address.
      read error (1). TCC= 2. TCINTEN= 1. TCCHEN= 0
NRP=0xc013460, NTSR=0x1000d, TSCH= 0x0, TSCL= 0x13501ea
```

```
 B3=0xc013428, A4=0x0, B4= 0x4, B14= 0x839fc0, B15= 0x838fb0
==================-watch dog timer test-=======================
start watch-dog timer...
service watch-dog 1 times, at time counter = 4
service watch-dog 2 times, at time counter = 6646748
service watch-dog 3 times, at time counter = 7229707
service watch-dog 4 times, at time counter = 7020074
service watch-dog 5 times, at time counter = 6726619
service watch-dog 6 times, at time counter = 6643649
service watch-dog 7 times, at time counter = 7002163
service watch-dog 8 times, at time counter = 6770232
service watch-dog 9 times, at time counter = 6738152
service watch-dog 10 times, at time counter = 6510560
stop servicing watch-dog, it will timeout and trigger NMI after 3000 ms...
NMI exception happened, normally you should reset the DSP to recover from the problem!
NRP=0xc014108, NTSR=0x1000d, TSCH= 0x0, TSCL= 0xaf397e41
 B3=0xc0140e4, A4=0x4b, B4= 0x1079551, B14= 0x839fc0, B15= 0x838f88
==================-Reserved space access test-=================
--------------------access zero address-----------------------
!!!read from reserved address 0x0...
External exception happened. MEXPFLAG[3]=0x4000000.
  Event 122: DMC_CMPA CPU memory protection fault for L1D (and other memory read
finally goes through the L1D controller)
  memory protection exception caused by local access at 0x0
  Supervisor Read violation
NRP=0xc0136a6, NTSR=0x1000d, TSCH= 0x1, TSCL= 0x2f6a969e
 B3=0xc013698, A4=0x25, B4= 0x28000c2, B14= 0x839fc0, B15= 0x838fa0

!!!write to reserved address 0x0...
External exception happened. MEXPFLAG[3]=0x10000000.
  Event 124: UMC_CMPA CPU memory protection fault for L2
  memory protection exception caused by local access at 0x0
  Supervisor Write violation
NRP=0xc0136e6, NTSR=0x1000d, TSCH= 0x1, TSCL= 0x2f6b83ef
 B3=0xc0136d4, A4=0x25, B4= 0x0, B14= 0x839fc0, B15= 0x838fa8
--------------------access address exceed LL2-----------------
!!!read from reserved address 0xa00000...
External exception happened. MEXPFLAG[3]=0x4000000.
  Event 122: DMC_CMPA CPU memory protection fault for L1D (and other memory read
finally goes through the L1D controller)
  memory protection exception caused by local access at 0xa00000
  Supervisor Read violation
NRP=0xc0136a6, NTSR=0x1000d, TSCH= 0x1, TSCL= 0x2f6c72f3
 B3=0xc013698, A4=0x2a, B4= 0x28000c2, B14= 0x839fc0, B15= 0x838fa0

!!!write to reserved address 0xa00000...
External exception happened. MEXPFLAG[3]=0x10000000.
  Event 124: UMC_CMPA CPU memory protection fault for L2
  memory protection exception caused by local access at 0xa00000
  Supervisor Write violation
NRP=0xc0136e6, NTSR=0x1000d, TSCH= 0x1, TSCL= 0x2f6d62b8
 B3=0xc0136d4, A4=0x2a, B4= 0xa00000, B14= 0x839fc0, B15= 0x838fa8
------------access local reserved configuration space----------
!!!read from reserved address 0x1000000...
External exception happened. MEXPFLAG[3]=0x4000000.
  Event 122: DMC_CMPA CPU memory protection fault for L1D (and other memory read
finally goes through the L1D controller)
  memory protection exception caused by local access at 0x1000000
  Supervisor Read violation
NRP=0xc0136a6, NTSR=0x1000d, TSCH= 0x1, TSCL= 0x2f6e550f
 B3=0xc013698, A4=0x2b, B4= 0x28000c2, B14= 0x839fc0, B15= 0x838fa0
```

```
!!!write to reserved address 0x1000000...
External exception happened. MEXPFLAG[3]=0x40000000.
  Event 126: EMC_CMPA CPU memory protection fault for local configuration space between
0x01000000-0x01BFFFFF
  memory protection exception caused by local access at 0x1000000
  Supervisor Write violation
NRP=0xc0136e4, NTSR=0x1000d, TSCH= 0x1, TSCL= 0x2f6f4561
 B3=0xc0136d4, A4=0x2b, B4= 0x1000000, B14= 0x839fc0, B15= 0x838fa8
---access reserved cfg space between INTC and power control----
!!!read from reserved address 0x18001c4...
External exception happened. MEXPFLAG[3]=0x4000000.
  Event 122: DMC_CMPA CPU memory protection fault for L1D (and other memory read
finally goes through the L1D controller)
  memory protection exception caused by local access at 0x18001c4
  Supervisor Read violation
NRP=0xc0136a6, NTSR=0x1000d, TSCH= 0x1, TSCL= 0x2f703bc7
 B3=0xc013698, A4=0x2b, B4= 0x18001c4, B14= 0x839fc0, B15= 0x838fa0

!!!write to reserved address 0x18001c4...
External exception happened. MEXPFLAG[3]=0x800000.
  Event 119: SYS_CMPA CPU Memory Protection Fault for local configuration of INTC and
power control
NRP=0xc0136e6, NTSR=0x1000d, TSCH= 0x1, TSCL= 0x2f712c2f
 B3=0xc0136d4, A4=0x2b, B4= 0x18001c4, B14= 0x839fc0, B15= 0x838fa8
------------access global reserved configration space----------
!!!read from reserved address 0x1cf0000...
External exception happened. MEXPFLAG[3]=0x84000000.
  Event 122: DMC_CMPA CPU memory protection fault for L1D (and other memory read
finally goes through the L1D controller)
  memory protection exception caused by local access at 0x1cf0000
  Supervisor Read violation
  Event 127: EMC_BUSERR Bus Error Interrupt for global configuration space between
0x01C00000 - 0x07FFFFFF
  CFG read status error detected
  XID (Transaction ID)= 12
  Addressing error
NRP=0xc0136a4, NTSR=0x1000d, TSCH= 0x1, TSCL= 0x2f71edc7
 B3=0xc013698, A4=0x2b, B4= 0x28000c2, B14= 0x839fc0, B15= 0x838fa0

!!!write to reserved address 0x1cf0000...
External exception happened. MEXPFLAG[3]=0x80000000.
  Event 127: EMC_BUSERR Bus Error Interrupt for global configuration space between
0x01C00000 - 0x07FFFFFF
  CFG write status error detected
  XID (Transaction ID)= 15
  Addressing error
NRP=0xc0136f0, NTSR=0x1000d, TSCH= 0x1, TSCL= 0x2f731225
 B3=0xc0136d4, A4=0x2b, B4= 0x1cf0000, B14= 0x839fc0, B15= 0x838fa8
-------------access reserved XMC configuration space-----------
!!!read from reserved address 0x8000100...
External exception happened. MEXPFLAG[3]=0x4004000.
  Event 110: MDMAERREVT XMC VBUSM error event
  MDMA read status error detected
  XID (Transaction ID)= 12
  Privilege error
  Event 122: DMC_CMPA CPU memory protection fault for L1D (and other memory read
finally goes through the L1D controller)
  memory protection exception caused by local access at 0x8000100
  Supervisor Read violation
NRP=0xc0136a4, NTSR=0x1000d, TSCH= 0x1, TSCL= 0x2f73fe47
```

```
 B3=0xc013698, A4=0x2b, B4= 0x28000c2, B14= 0x839fc0, B15= 0x838fa0

!!!write to reserved address 0x8000100...
External exception happened. MEXPFLAG[3]=0x4000.
  Event 110: MDMAERREVT XMC VBUSM error event
  memory protection exception caused by local access at 0x8000100
  Supervisor Write violation
NRP=0xc0136ea, NTSR=0x1000d, TSCH= 0x1, TSCL= 0x2f751dd9
 B3=0xc0136d4, A4=0x2b, B4= 0x8000100, B14= 0x839fc0, B15= 0x838fa8
-----------------access XMC unmapped space--------------------
!!!read from reserved address 0x1c000000...
External exception happened. MEXPFLAG[3]=0x4004000.
  Event 110: MDMAERREVT XMC VBUSM error event
  MDMA read status error detected
  XID (Transaction ID)= 6
  Privilege error
  Event 122: DMC_CMPA CPU memory protection fault for L1D (and other memory read
finally goes through the L1D controller)
  memory protection exception caused by local access at 0x1c000000
  Supervisor Read violation
NRP=0xc0136a4, NTSR=0x1000d, TSCH= 0x1, TSCL= 0x2f760f0c
 B3=0xc013698, A4=0x2c, B4= 0x28000c2, B14= 0x839fc0, B15= 0x838fa0

!!!write to reserved address 0x1c000000...
External exception happened. MEXPFLAG[3]=0x4000.
  Event 110: MDMAERREVT XMC VBUSM error event
  memory protection exception caused by local access at 0x1c000000
  Supervisor Write violation
NRP=0xc0136ea, NTSR=0x1000d, TSCH= 0x1, TSCL= 0x2f772e9b
 B3=0xc0136d4, A4=0x2c, B4= 0x1c000000, B14= 0x839fc0, B15= 0x838fa8
-----------------execute from reserved space-------------------
!!!execute at reserved address 0x0...
internal excpetion happened. IERR=0x1.
  Instruction fetch exception
NRP=0x0, NTSR=0x1000d, TSCH= 0x1, TSCL= 0x2f781ef7
 B3=0xc013744, A4=0x26, B4= 0x0, B14= 0x839fc0, B15= 0x838fa0
Exception happened at a place can not safely return!
```

8 References

1. Memory Protection On Keystone Devices (SPRWIKI9012)
2. TMS320C66x DSP CPU and Instruction Set Reference Guide (SPRUGH7)
3. TMS320C66x DSP CorePac User Guide (SPRUGW0)
4. KeyStone Architecture Multicore Shared Memory Controller User Guide (SPRUGW7)
5. KeyStone Architecture Memory Protection Unit (MPU) User Guide (SPRUGW5)
6. KeyStone Architecture Enhanced Direct Memory Access (EDMA3) Controller User Guide (SPRUGS5)
7. KeyStone Architecture Timer64 User Guide (SPRUGV5)
8. KeyStone Architecture Chip Interrupt Controller (CIC) User Guide (SPRUGW4)
9. KeyStone Architecture DDR3 Memory Controller's user guider (SPRUGV8)
10. TMS320C6000 Optimizing Compiler User's Guide (SPRU187)
11. "Interrupts", "MPU" and "Memory Map Summary" sections in Device specific Data Manuals