



ECC Injection Tests in Traveo MCUs - KBA225469

This guide explains Error Correction Code (ECC) Injection Tests in Traveo MCUs.

In Traveo MCUs, there is built-in ECC logic that helps in detecting and correcting single-bit errors or in detecting 2- or multi-bit errors. This built-in ECC logic is available within many peripheral memories of Traveo MCUs such as TCRAM, TCFLASH, VRAM, Work Flash, IRC Vector Address RAM, CAN FD RAM, and SYSRAM.

When a product is deployed in the field, an error in these memories can happen due to [Soft Errors](#). If the error results in a 1-bit flip of the memory, the hardware by itself can correct this 1-bit flip when the affected memory is read. On the other hand, if the flips are in multiple bit locations, the hardware can trigger an Interrupt intimating the user that a serious issue has occurred and suitable actions (for example, a Software Reset) should be done.

Traveo MCUs also provide necessary hardware and Test Registers (known as Injection Registers) that could simulate the data corruptions and check if the ECC logic is able to detect or correct these errors. These tests can be performed as part of the functional safety module associated with the product design.

This guide focuses on the details of these Injection Registers and how they can be used to simulate the errors. The guide also provides sample code fragments that can be used to test the ECC feature available within a peripheral memory.

Note that [Green Hills Software \(GHS\)](#) has been used for code development and testing of all codes provided in this guide.

1. ECC for TCFLASH

TCFLASH is capable of correcting 1-bit errors and detect up to 2-bit errors using 8-bit error correcting codes for every 64 bits in the memory. ECC is generated upon write to the address in the TCFLASH and error correction/detection is performed upon read.

See the Platform manual [TCFLASH chapter Section 3.8](#) for more details on ECC generation, Syndrome and ECC registers.

If ECC test functionality is enabled, upon detecting 1-bit errors, the TCFLASH corrects the bit and generates an interrupt. If errors are detected in two or more bits, the TCFLASH cannot correct the error but can detect the 2-bit error. This causes bus error that in turn leads to Synchronous External Abort.

The Interrupt Vector map is shown below; IRQ No. 8 is used for TCFLASH 1-bit ECC correction interrupt. Note that the interrupt is generated upon detecting and correcting 1-bit ECC error and TCFCFG_FSECIR. SECINT bit is set.

Figure 1. TCFLASH ECC IRQ

1. IRQ Map

This section shows list of interrupt vector.

This list shows the assignment of interrupt vectors/interrupt control registers.

Vector of not-implemented function is not supported. See the chapter of function list.

IRQ No.	Detail	IRQ Priority Register	Vector Address Register
0	Reserved	-	-
1	System Control Status	IRC0_IROPL0 : IRQPL1	IRC0_IRQVA1
2	HW-WDT Pre-warning	IRC0_IROPL0 : IRQPL2	IRC0_IRQVA2
3	SW-WDT Pre-warning	IRC0_IROPL0 : IRQPL3	IRC0_IRQVA3
4 to 7	Reserved	-	-
8	TCFLASH RDY, Hang up, Single Bit Error	IRC0_IROPL2 : IRQPL8	IRC0_IRQVA8
9	Reserved	-	-

To test the TCFLASH ECC functionality via AXI (Advanced eXtensible Interface), appropriate error injection registers need to be used. By enabling the error injection feature, any read access to the TCFLASH memory location will result in an ECC error as the data bits read will be XORed with the ECC error injection mask to flip the bit accordingly.

1.1 Registers Needed for TCFLASH ECC Tests

Table 1. TCFLASH ECC Registers

Abbreviated Register Name	Register Name	Register Details
TCFCFG_FCPROTKEY	TCFLASH Configuration Protection Key register	Protection key register to unlock write to TCFLASH registers once
TCFCFG_FECCCTRL	TCFLASH ECC Control register	ECC control register to enable or disable ECC
TCFCFG_FDATEIR	TCFLASH Data Bit Error-Injection register	Error injection bit positions in 64- bit data
TCFCFG_FECCEIR	TCFLASH ECC it Error Injection register	Error injection bit positions in 8- bit ECC
TCFCFG_FSTAT	TCFLASH Status register	Status to indicate interrupt and mask
TCFCFG_FECCEAR	TCFLASH ECC Error Address register	Indicates ECC error address in case of 1-bit error

Abbreviated Register Name	Register Name	Register Details
TCFCFG_FUCEAR	TCFLASH Uncorrectable Error Address register	Indicates the ECC error address location upon detecting 2-bit non-correctable error

1.2 Sample Codes

1.2.1 1-bit ECC Error Injection with AXI Access

Consider the following function:

```

/*Testing ECC functionality with 1-bit Error Injection */
static void Test_Ecc_1bit(void)
{
    /* Unlock */
    TCFCFG_FCPROTKEY = 0xCF61F1A5;
    /* Inject Error (bit 0) in LS 32 bit */
    TCFCFG_FDATEIR_L = 0x01U;
    /* Unlock */
    TCFCFG_FCPROTKEY = 0xCF61F1A5;
    /* Inject Error in MS 32 bit (not set) */
    TCFCFG_FDATEIR_H = 0U;
    /* Unlock */
    TCFCFG_FCPROTKEY = 0xCF61F1A5;
    /* Enable 1-bit ECC detection Interrupt*/
    TCFCFG_FSECIR    = 1U;
    /* Clear Cache */
    DSB();
    /* Read the location to inject error*/

```



```
Read_FG = *((uint32_t*)0x01B000FC);
```

```
}
```

Note that the byte setting TCFCFG_FDATEIR_L to 0x01 will inject an error into the 0th bit of the lower 32 bits of the data. This function enables the Interrupt Service Routine (ISR) when a 1-bit ECC error is detected and corrected. The IRQ 8 is mapped to TCFLASH Single Bit Error occurrence and associated ISR can be used to log any information about the ECC error. A sample code to handle this is shown below:

```
/* ISR for the 1-bit ECC detection and correction by TCFLASH via AXI access. */
FN_IRQ_DEFINE_BEGIN(testecc, INTERRUPTS_IRQ_NUMBER_8)
{

/* 1-bit ECC error detected while accessing a location of TCFLASH via AXI
Clear Interrupt source */

    TCFCFG_FCPROTKEY = 0xCF61F1A8;
    TCFCFG_FSECIR |= 0x10U;

}
FN_IRQ_DEFINE_END()
```

As shown in Figure 2, after executing read to a location in TCFLASH (via AXI), the single-bit ECC error is injected into the data field of that location and an ECC error is detected, and corrected, the control goes to the ISR.

Figure 2. TCFLASH 1-Bit ECC Code

```

static void Test_Ecc_1bit(void)
{
    /* Unlock */
    TCFCFG_FCPROTKEY = 0xCF61F1A5;
    /* Inject Error(bit 0) in LS 32 bit */
    TCFCFG_FDATEIR_L = 0x01U;
    /* Unlock */
    TCFCFG_FCPROTKEY = 0xCF61F1A5;
    /* Inject Error in MS 32 bit(not set) */
    TCFCFG_FDATEIR_H = 0U;
    /* Unlock */
    TCFCFG_FCPROTKEY = 0xCF61F1A5;
    /* Enable 1-bit ECC detection Interrupt*/
    TCFCFG_FSECIR = 1U;
    /* Clear Cache */
    DSB();
    /* Read the location to inject error*/
    Read_FG = *((uint32_t*)0x01B000FC);
}

```

Figure 3. TCFLASH 1-Bit Interrupt Handler

```

/* ISR for the 1-bit ECC detection and correction by TCFLASH via AXI access. */
FN_IRQ_DEFINE_BEGIN(testecc, INTERRUPTS_IRQ_NUMBER_8)
{
    /* 1-bit ECC error detected while accessing a locaiton of TCFLASH via AXI
    Clear Interrupt source */
    TCFCFG_FCPROTKEY = 0xCF61F1A5;
    TCFCFG_FSECIR |= 0x10U;
}
FN_IRQ_DEFINE_END()

```

The register values can also be monitored to check the ECC error address, ECC error detection interrupt, and the Syndrome value. In GHS, this can be done using **View > Registers** option after entering Debug mode.

Figure 4. TCFLASH Error registers

<input type="checkbox"/> TCFCFG_FSECIR	0xce010001	
SECIE	0x1	
SECIC	0x0	
SECINT	0x1	
SYN	0xce	
<input checked="" type="checkbox"/> TCFCFG_FECCEAR	0x01b000e0	

From the value of the syndrome, it can be inferred that bit 0 is flipped and an error was detected during ECC check. Note that the 128-bit aligned address is reported in the TCFCFG_FECCEAR register due to the properties of the cache.

1.2.2 2-bit ECC Error Injection via AXI Access

Consider the following function:

```

/* Testing the ECC functionality of TCFLASH with 2-bit Error injection */
static void Test_Ecc_2bit(void)
{

    TCFCFG_FCPROTKEY = 0xCF61F1A5; /* Unlock */
    TCFCFG_FDATEIR_L = 0x03U;      /* Inject Error in LS 32 bit */
    TCFCFG_FCPROTKEY = 0xCF61F1A5; /* Unlock */
    TCFCFG_FDATEIR_H = 0x00U;      /* Inject Error in MS 32 bit */
    TCFCFG_FCPROTKEY = 0xCF61F1A5; /* Unlock */
    TCFCFG_FSECIR    = 1U;         /* Enable 1-bit ECC detection Interrupt*/

    /* Read the location to inject 2-bit error,
    Control would enter Data Abort handler*/

```

```
Read_FG = *((uint32_t*)0x01B000F0);
```

```
}
```

Writing 0x03 to TCFCFG_FDATEIR_L injects 2-bit errors into the TCFLASH location. 2-bit ECC errors can be detected but cannot be corrected; this will result in a bus error as shown in Figure 5:

Figure 5. TCFLASH 2-Bit ECC Code

```

/* Testing the ECC functionality of TCFLASH with 2-bit Error injection */
static void Test_Ecc_2bit(void)
{
    TCFCFG_FCPROTKEY = 0xCF61F1A5; /* Unlock */
    TCFCFG_FDATEIR_L = 0x03U; /* Inject Error in LS 32 bit */
    TCFCFG_FCPROTKEY = 0xCF61F1A5; /* Unlock */
    TCFCFG_FDATEIR_H = 0x00U; /* Inject Error in MS 32 bit */
    TCFCFG_FCPROTKEY = 0xCF61F1A5; /* Unlock */
    TCFCFG_FSECIR = 1U; /* Enable 1-bit ECC detection Interrupt*/

    /* Read the location to inject 2 bit error,
    Control would enter Data Abort handler*/
    Read_FG = *((uint32_t*)0x01B000F0);
}

```

Figure 6. Data Abort for 2-Bit TCFLASH Error

```
static void DefaultDataAbortExceptionHandler(void)
{
    // IRQs are disabled now.
    // Please refer to the ARM documentation on how to handle this exception.

    // For debugging purpose only
    #ifdef DEBUG
    // Collect some information about the cause of the data abort
    volatile uint32_t u32DataFaultStatus    = MRC( 15, 0, 5, 0, 0 );
    volatile uint32_t u32AuxDataFaultStatus = MRC( 15, 0, 5, 1, 0 );
    volatile uint32_t u32DataFaultAddress   = MRC( 15, 0, 6, 0, 0 );
    #endif // DEBUG

    → while (1)
    {
        •   NOP();
        •
    }
}
```

Bus error results in Synchronous External Data Abort as shown below. The DFAR register holds the address, which when accessed, results in the abort.

Figure 7. Registers During Abort Mode

[-] DFSR	0x00001008
[-] ExT	0x1
[-] WnR	0x0
[-] FS[4]	0x0
[-] FS[3:0]	0x8
[+] IFSR	0x00000000
[+] ADFSR	0x00000000
[+] AIFSR	0x00000000
[+] DFAR	0x01b000f0

The Syndrome indicates that the ECC error detected is a 2-bit error; the error can be detected but not corrected. Also, the ECC address is reported in TCFCFG_FUCEAR.

Figure 8 TCFLASH Error registers during Abort

[-] TCFCFG_FUCEDIR	0x05010000	
UCEDIC	0x0	
UCEDINT	0x1	
SYN	0x05	
[+] TCFCFG_FUCEAR	0x01b000e0	

The reported value in TCFCFG_FECCEAR is 0x01b00e0 instead of 0x01b00f0 because of the memory region attribute of write-back cacheable for the TCFLASH region. Because the cache lane is 32-byte aligned, wrap-around occurs as shown below while trying to access 0x01b000f0 as follows:

Access to 0x01b000f0 -> cache controller fetches from: 0x01b000f0, 0x01b000f8, 0x01b000e0, 0x01b000e8

Because TCFCFG_FECCEAR is updated with the 128 bit-aligned address that was last read, you can observe 0x01b000e0. If the memory region is made non-cacheable, you would see the 128 bit-aligned read address, which in this case would be 0x01b000f0.

Note the following important points while performing TCFLASH ECC error injection:

- If ECC is enabled, error injection happens globally across the address range. i.e., errors are injected in any read location in TCFLASH via both AXI and TCM space.



- While testing TCFLASH ECC via AXI, it is recommended to place the test handler (ISR) in a RAM location or execute the handler via TCM space (disable ECC checks in CPU). If not, it may trigger ECC error detection continuously in case of 1-bit ECC and may result in prefetch aborts in case of 2-bit ECC because any read to the code region also gets injected with errors; the CPU prefetch unit will continue fetching code from flash memory that it wants to execute.

You can also choose to inject errors into the ECC bits in the TCFLASH instead of data bits by writing to FECCEIR register.

See ARM manuals for details on ECC error checks in case of TCM access.

2. ECC for IRC Vector Address RAM

The interrupt controller block in Traveo MCU is equipped with an SRAM which has the ECC protection function. This SRAM holds the vector address of the mapped interrupt vectors. If a 1-bit error (correctable) occurs in this SRAM memory, the `IRCN_EEI:EEIS` bit is set and an IRQ is generated if the `IRQVAr` register is read by the CPU or from the IRQ processing stage.

If an error of 2 or more bits (uncorrectable) occurs, the `IRCN_EEI:EENS` bit is set and an NMI is generated if the `IRQVAr` register is read by the CPU. If the SRAM read happens from the Interrupt Controller, a transition is made to the function pointer indicated by `IRCN_IRQEEVA`. Therefore, also make sure that a valid address is set in `IRCN_IRQEEVA`.

Note that IRQ number and NMI number for single-bit error and multi-bit errors are respectively specified in the IRQ Map and NMI Map sections of the Series Hardware Manual.

Figure 9. IRQ for IRC Vector Address RAM Single-Bit Error

1. IRQ Map

This section shows list of interrupt vector.

This list shows the assignment of interrupt vectors/interrupt control registers.

Vector of not-implemented function is not supported. See the chapter of function list.

IRQ No.	Detail	IRQ Priority Register	Vector Address Register
0	Reserved	-	-
1	System Control Status	IRC0_IRQPL0 : IRQPL1	IRC0_IRQVA1
2	HW-WDT Pre-warning	IRC0_IRQPL0 : IRQPL2	IRC0_IRQVA2
3	SW-WDT Pre-warning	IRC0_IRQPL0 : IRQPL3	IRC0_IRQVA3
4 to 7	Reserved	-	-
8	TCFLASH RDY, Hang up, Single Bit Error	IRC0_IRQPL2 : IRQPL8	IRC0_IRQVA8
9	Reserved	-	-
10	Work FLASH Hang up	IRC0_IRQPL2 : IRQPL10	IRC0_IRQVA10
11 to 13	Reserved	-	-
14	System RAM Single Bit Error	IRC0_IRQPL3 : IRQPL14	IRC0_IRQVA14
15	Backup RAM / CAN FD RAM(ch 0, 1, 5, 6) Single Bit Error	IRC0_IRQPL3 : IRQPL15	IRC0_IRQVA15
16	IRC Vector Address RAM Single Bit Error	IRC0_IRQPL4 : IRQPL16	IRC0_IRQVA16
17 to 19	Reserved	-	-

Figure 10. NMI for IRC Vector Address RAM 2-Bit Error

2. NMI Map

This section shows NMI map.

NMI of not-implemented function is not supported. See the chapter of function list.

NMI No.	Detail	Priority Level	IRQ/NMI Vector Address
0	NMIX pin(Ext-IRC)	IRC0_NMIPL0 : NMIPL0	IRC0_NMIVA0
1 to 2	Reserved	-	-
3	Reserved	-	-
4	LVDs IRQ	IRC0_NMIPL1 : NMIPL4	IRC0_NMIVA4
5	CSV, Profile	IRC0_NMIPL1 : NMIPL5	IRC0_NMIVA5
6	HW-WDT	IRC0_NMIPL1 : NMIPL6	IRC0_NMIVA6
7	SW-WDT	IRC0_NMIPL1 : NMIPL7	IRC0_NMIVA7
8	IRC 2-bit ECC err detection	IRC0_NMIPL2 : NMIPL8	IRC0_NMIVA8
9 to 10	Reserved	-	-

Note that 1-bit errors are always corrected and sent to the master who accessed the SRAM holding the vector addresses. If you enable the 1-bit ECC error interrupt, the interrupt handler (IRQ-16) would be called. To correct the error in SRAM, you can write back the vector address into the SRAM to force a recalculation of the ECC by hardware. IRCn_EAN holds the relative address of the SRAM where the error occurred.

If a double-bit ECC error in the SRAM is detected by reading IRCn_IRQVAr by the CPU, a correct vector address must be written again in the relevant IRCn_IRQVAr register with the corresponding NMI handler. Then, the error bit must be cleared by writing to IRCn_EE1:EENC.

If a double-bit ECC error in the SRAM is detected by reading IRCn_IRQVAr by the IRQ processing unit, the correct vector address must be rewritten to IRCn_IRQVAr register corresponding to the IRQEEVA handler.

More details about these registers are covered in the subsequent sections.

2.1 ECC Test Function

The interrupt controller also has a function for generating pseudo ECC errors by corrupting the data read from SRAM. Using this function, you can check the operation of the ECC error handler while developing applications. By setting the ECC error bit registers (IRCn_EEB0 to IRCn_EEB2), you can invert the data read from the SRAM by using the bitwise operation. This test function can be enabled/disabled using the ECC test register (IRCn_ET).

2.1.1 Registers needed for ECC Tests for IRC Vector Address RAM

Table 2. IRC Vector Address RAM ECC Registers

Abbreviated Register Name	Register Name	Register Details
IRCN_ET	IRC ECC test register	This register sets enable/disable of the test mode for the ECC protection function of the SRAM in the interrupt controller.
IRCN_UNLOCK	IRC Unlock Register	This register controls the write lock for the interrupt controller to each register.
IRCN_EEB0 to IRCn_EEB1	IRC ECC error bit register	These registers are used in ECC test mode. In the data read from the SRAM, you can spuriously corrupt any of the bits in the data area.
IRCN_EEB2	IRC ECC error bit register	This register is used in ECC test mode. In the data read from the SRAM, you can spuriously corrupt any of the bits in the ECC parity area.
IRCN_IRQVAr	IRC IRQ vector address register	This register indicates the 32-bit vector address of individual IRQ channels. Before the corresponding IRQ channel is enabled, the vector address should be initialized by software. The same type of register is installed for each IRQ channel. The "r" in the abbreviated register name corresponds to the IRQ channel number, r (0 to 511). This register is on SRAM of the interrupt controller.
IRCN_IRQEEVA	IRC ECC error vector address register	This register is used to perform correct error handling when SRAM holding the IRQ vector is read by the IRQ processing block and a 2-bit ECC error is detected.
IRCN_EEI	IRC ECC error interrupt register	This register indicates the ECC error interrupt status. It also clears the interrupt. There are two types of ECC error interrupt: NMI and IRQ.
IRCN_EAN	IRC ECC address number register	When a single-bit or double-bit ECC error occurs, this register indicates the SRAM address of the error.

Note that IRCn_EEB0 to IRCn_EEB2 (n = 0) are the critical registers that need to be written to corrupt the data in the SRAM area of the interrupt controller.

The registers IRC0_EEB0 and IRC0_EEB2.EEBE correspond to the even vectors i.e., IRC0_IRQVA0, IRC0_IRQVA2, IRC0_IRQVA4, etc., and the registers IRC0_EEB1 and IRC0_EEB2.EEBO correspond to the odd vectors i.e., IRC0_IRQVA1, IRC0_IRQVA3, IRC0_IRQVA5, etc.

Also note that only the lower eight bits of IRCn_EAN registers are significant; they hold a relative address only. IRCn_EAN.EAN represents the 8-byte offset of the ECC error occurrence address from the SRAM base address (i.e., the address of IRCn_IRQVA0). Effectively, the absolute error occurrence address will be

$$= (\text{Addr of IRCn_IRQVA0}) + (8 \times \text{IRCN_EAN.EAN})$$



2.2 Sample Codes for ECC Injection Tests

2.2.1 2 Bit Error Injection and Handling NMI

Consider the following function:

```
void Test_ECC_2Bit(void)
{
    volatile uint32_t Read_Vector;

    /* Unlock Interrupt Controller */
    IRC0_UNLOCK = 0x17ACC911;

    /* Set Address to trigger ISR when read by Interrupt Controller */
    IRC0_IRQEEVA = (uint32_t)Test_ECC2Bit_Int_Controller;

    /* Enable the Self Test */
    IRC0_ET_ET = 1;

    /* Set Double Bit Error in Even Vector Address*/
    IRC0_EEB0_EEB = 0x0000000C;

    /* Read an Even Vector Address to trigger the IRQ */
    Read_Vector = (uint32_t)IRC0_IRQVA2;

    /* Lock Interrupt Controller */
    IRC0_UNLOCK = 0x17B10C11;
}
```



Because you are setting two bits as '1' in IRC0_EEB0_EEB, you are corrupting two bits of the SRAM memory. Now, a read to any of the even vector address by the CPU (for example, IRC0_IRQVA2) would trigger NMI8.

If the operating system has initialized/mapped this NMI, it can be serviced using a similar function as shown below:

```
FN_NMI_DEFINE_BEGIN(ECC_2BitError_NMI, INTERRUPTS_NMI_NUMBER_8)
{
    /* 2 Bit ECC Detected, triggered the NMI */
    /* Add code to trigger Software Reset or required user action */

    /* Clear Hold Bit for NMI8 */
    IRC0_NMIHC = 0x08;

    /* Clear Error Bit */
    IRC0_EEI_EENC = 0x1;
}
```

As shown in Figure 11, you can see "ECC_2BitError_NMI" being hit after a read of IRC0_IRQVA2.

Figure 11. IRC RAM 2 Bit Error and NMI Handler

```

void Test_ECC_2Bit(void)
{
    volatile uint32_t Read_Vector;

    /* Unlock Interrupt Controller */
    IRCO_UNLOCK = 0x17ACC911;

    /* Set Address to trigger ISR when read by Interrupt Controller */
    IRCO_IRQEEVA = (uint32_t)Test_ECC2Bit_Int_Controller;

    /* Enable the Self Test */
    IRCO_ET_ET = 1;

    /* Set Double Bit Error in Even Vector Address*/
    IRCO_EEBO_EEB = 0x0000000C;

    /* Read an Even Vector Address to trigger the IRQ */
    Read_Vector = (uint32_t)IRCO_IRQVA2;

    /* Lock Interrupt Controller */
    IRCO_UNLOCK = 0x17B10C11;
}

```

```

FN_NMI_DEFINE_BEGIN(ECC_2BitError_NMI, INTERRUPTS_NMI_NUMBER_8)
{
    /* 2 Bit ECC Detected, triggered the NMI */
    /* Add code to trigger Software Reset or required user action */

    /* Clear Hold Bit for NMI8 */
    IRCO_NMIHC = 0x08;

    /* Clear Error Bit */
    IRCO_EEI_EENC = 0x1;
}
FN_NMI_DEFINE_END()

```

2.2.2 1-Bit Error Injection and Handling IRQ

Consider the following function:

```
void Test_ECC_1Bit(void)
```




```
{  
    volatile uint32_t Read_Vector;  
  
    /* Unlock Interrupt Controller */  
    IRC0_UNLOCK = 0x17ACC911;  
  
    /* Enable the Self Test */  
    IRC0_ET_ET = 1;  
  
    /* Set Single Bit Error in Even Vector Address*/  
    IRC0_EEB0_EEB = 0x00000004;  
  
    /* Read an Even Vector Address to trigger the IRQ */  
    Read_Vector = (uint32_t)IRC0_IRQVA2;  
  
    /* Lock Interrupt Controller */  
    IRC0_UNLOCK = 0x17B10C11;  
  
}
```

Because you are setting one bit as [1' in IRC0_EEB0_EEB, you are corrupting one bit of the SRAM memory. Now, a read to any of the even vector address (for example, IRC0_IRQVA2) would trigger IRQ16.

If the operating system has initialized/mapped this IRQ, it can be serviced using a similar function as shown below:

```
FN_IRQ_DEFINE_BEGIN(ECC_1BitError_IRQ, INTERRUPTS_IRQ_NUMBER_16)  
{  
    /* 1 Bit ECC Detected, triggered the IRQ */
```



```
IRC0_EEI_EEIC = 0x1;

/* Clear Hold Bit for IRQ16 */
IRC0_IRQHC = 0x00000010;
}
FN_IRQ_DEFINE_END()
```

As shown in Figure 12, you can see “ECC_1BitError_IRQ” being hit after a read of IRC0_IRQVA2.

Figure 12. IRC RAM 1 Bit Error and IRQ Handler

```

void Test_ECC_1Bit(void)
{
    volatile uint32_t Read_Vector;

    /* Unlock Interrupt Controller */
    IRCO_UNLOCK = 0x17ACC911;

    /* Enable the Self Test */
    IRCO_ET_ET = 1;

    /* Set Single Bit Error in Even Vector Address*/
    IRCO_EEBO_EEB = 0x00000004;

    /* Read an Even Vector Address to trigger the IRQ */
    Read_Vector = (uint32_t)IRCO_IRQVA2;

    /* Lock Interrupt Controller */
    IRCO_UNLOCK = 0x17B10C11;
}

FN_IRQ_DEFINE_BEGIN(ECC_1BitError_IRQ, INTERRUPTS_IRQ_NUMBER_16)
{
    /* 1 Bit ECC Detected, triggered the IRQ */
    IRCO_EEI_EEIC = 0x1;

    /* Clear Hold Bit for IRQ16 */
    IRCO_IRQHC = 0x00000010;
}
FN_IRQ_DEFINE_END()

```

3. ECC for VRAM

Video RAM (VRAM) is a 2-MB embedded SRAM, which is part of the Graphics Subsystem. The VRAM also has built-in ECC that can be enabled for various VRAM sub-regions. Note that because this ECC region is shared with the user region, the memory size available for the user program shall reduce if ECC is enabled. You can define the ECC-enabled area and ECC-disabled area as part of the VRAM configurations.

Single-bit and double-bit error detection triggers respective NMI, which can be handled in the user application.

Figure 13 shows the NMIs for VRAM error detection:

Figure 13. NMI for VRAM ECC

2. NMI Map

This section shows NMI map.

NMI of not-implemented function is not supported. See the chapter of function list.

NMI No.	Detail	Priority Level	IRQ/NMI Vector Address
0	NMIX pin(Ext-IRC)	IRC0_NMIPL0 : NMIPL0	IRC0_NMIVA0
1 to 2	Reserved	-	-
3	Reserved	-	-
4	LVDs IRQ	IRC0_NMIPL1 : NMIPL4	IRC0_NMIVA4
5	CSV, Profile	IRC0_NMIPL1 : NMIPL5	IRC0_NMIVA5
6	HW-WDT	IRC0_NMIPL1 : NMIPL6	IRC0_NMIVA6
7	SW-WDT	IRC0_NMIPL1 : NMIPL7	IRC0_NMIVA7
8	IRC 2-bit ECC err detection	IRC0_NMIPL2 : NMIPL8	IRC0_NMIVA8
9 to 10	Reserved	-	-
11	Backup RAM 2-bit ECC error detection	IRC0_NMIPL2 : NMIPL11	IRC0_NMIVA11
12	M-CAN RAMs 2-bit ECC error detection	IRC0_NMIPL3 : NMIPL12	IRC0_NMIVA12
13	DMAC MPU #0 protection violation	IRC0_NMIPL3 : NMIPL13	IRC0_NMIVA13
14	Reserved	-	-
15	SHE MPU	IRC0_NMIPL3 : NMIPL15	IRC0_NMIVA15
16 to 17	Reserved	-	-
18	TPU protection violation	IRC0_NMIPL4 : NMIPL18	IRC0_NMIVA18
19	Reserved	-	-
20	Graphics subsystem Memory Protection	IRC0_NMIPL5 : NMIPL20	IRC0_NMIVA20
21	Graphics subsystem (VRAM) ECC Single Bit Error Detection	IRC0_NMIPL5 : NMIPL21	IRC0_NMIVA21
22	MLB_MPU_NMI	IRC0_NMIPL5 : NMIPL22	IRC0_NMIVA22
23	ETHERNET_MPU_NMI	IRC0_NMIPL5 : NMIPL23	IRC0_NMIVA23
24 to 31	Reserved	-	-

A single-bit error in the VRAM triggers NMI 21 when the erroneous memory address is read in the application. You can read the `vram_sberraddr_sn` register (where 'n' varies from 0 to 2) to know the VRAM address location that has experienced the bit flip.

A double-bit error in the VRAM triggers NMI 20 when the erroneous memory address is read in the application. NMI 20 is also triggered when the Graphics Subsystem Memory Protection Unit and Bus Monitoring detect any error.

NMIs should be handled in the user application to clear the interrupt and check the VRAM address where error has occurred. Single-bit errors are corrected by the ECC block; however, it is the user's responsibility to either reset the controller or rewrite the memory again in case of double-bit errors.

3.1 ECC Test Function

The VRAM also supports injecting pseudo ECC errors by corrupting the data read from the VRAM. The error can be injected using `vram_errinj_ecc_sn_lo` and `vram_errinj_ecc_sn_hi` registers. By setting a bit in these registers, the respective bit is inverted in the data read from the VRAM.

3.1.1 Registers Needed for ECC Tests for VRAM

Table 3. VRAM ECC Registers

Abbreviated Register Name	Register Name	Register Details
<code>vram_errinj_ecc_sn_lo</code>	<code>vram_errinj_ecc_sn_lo</code> (where 'n' varies from 0 to 2)	ECC error injection for Sn interface (check bits for lower 32 bits).
<code>vram_errinj_ecc_sn_hi</code>	<code>vram_errinj_ecc_sn_hi</code> (where n varies from 0 to 2)	ECC error injection for Sn interface (check bits for upper 32 bits).
<code>vram_LockUnlock</code>	<code>vram_LockUnlock</code>	Register to change the protection status of the address block 2.(Refer Table 2)
<code>vram_LockStatus</code>	<code>vram_LockStatus</code>	Updates the VRAM lock status, privilege status and freeze status.
<code>vram_sram_select</code>	<code>vram_sram_select</code>	Selects the ECC-protected region in units of 4 KB. A value of '0' means there is no protected region. This field accepts the value greater than 511. Such a value means that whole of 2 MB are protected region.
<code>VRamInterruptEnable</code>	<code>VRamInterruptEnable</code>	VRAM non-maskable interrupt enable register. Writing '1' to a bit in this register enables the VRAM interrupt. There are three LSb bits, each for VRAM Interrupt ECC 3DGC, VRAM Interrupt ECC Disp and VRAM Interrupt ECC 2DGC Sys
<code>VRamInterruptClear</code>	<code>VRamInterruptClear</code>	VRAM non-maskable interrupt clear register. Writing '1' to a bit in this register clears the respective interrupt.

Here is the flow for performing the VRAM ECC test:

1. Unlock VRAM registers by writing 0x691DB936 to the `vram_LockUnlock` register. Wait until the unlock is successful by monitoring the `vram_LockStatus` register.
2. Select the VRAM region for which ECC should be enabled, by using the `vram_sram_select` register.
3. Enable error injection by setting one/ two bits of `vram_errinj_ecc_sn_lo` and `vram_errinj_ecc_sn_hi`, for single/double bit error.

4. Enable the VRAM interrupt by setting the respective bits of the VRamInterruptEnable register.
5. Lock VRAM registers by writing 0x5651f763 to the vram_LockUnlock register.
6. Write a dummy byte to the VRAM location and read it back.
7. Handle NMI 21/NMI 20 for clearing the interrupt for single bit/double bit error.

3.2 Sample Codes for ECC Injection Tests

3.2.1 1-Bit Error Injection and Handling NMI

The following code snippet injects one-bit error in a VRAM location:

```
static void ECC_Test()  
{  
    /*Unlock the VRAM registers*/  
    GRPSUB_SUBC_VRAM_LOCKUNLOCK=0x691DB936;  
    while (GRPSUB_SUBC_VRAM_LOCKSTATUS == 1);  
    /*Write 512 to VRAM Select register to enable ECC for entire VRAM region*/  
    GRPSUB_SUBC_VRAM_SRAM_SELECT = 512;  
    /*Inject 1 bit error by setting one bit in error injection register. */  
    GRPSUB_SUBC_VRAM_ERRINJ_ECC_S0_LO_VRAM_ERRINJ_ECC_S0_LO = 0x000000001ul;  
    GRPSUB_SUBC_VRAM_ERRINJ_ECC_S0_HI_VRAM_ERRINJ_ECC_S0_HI = 0x00;  
    /*Enable VRAM interrupts for error detection*/  
    GRPSUB_SUBC_VRAMINTERRUPTENABLE = 1;  
    /*Lock the VRAM registers*/  
    GRPSUB_SUBC_VRAM_LOCKUNLOCK = 0x5651f763;  
    while (GRPSUB_SUBC_VRAM_LOCKSTATUS == 1);  
}
```



```
/*Write a dummy data to a VRAM address (0x50000000) */
*((volatile uint32_t*)0x50000000) = 0x25;

DSB ();

/*Read the VRAM address*/
Read_vram= *((volatile uint32_t*)0x50000000)-Read_vram;
}
```

As soon as the VRAM location is read, NMI 21 is triggered, which shall be handled in the user application. A sample code for handling this NMI is shown below:

```
FN_NMI_DEFINE_BEGIN(test_ECC, INTERRUPTS_NMI_NUMBER_21)
{
    /*unlock the key register for VRAM*/
    GRPSUB_SUBC_VRAM_LOCKUNLOCK=0x691DB936;
    while (GRPSUB_SUBC_VRAM_LOCKSTATUS == 1)
    {
    }
    /* clear the injected error*/
    GRPSUB_SUBC_VRamInterruptClear =0x7U;
    GRPSUB_SUBC_VRAM_ERRINJ_ECC_S0_LO_VRAM_ERRINJ_ECC_S0_LO = 0x00000000ul;
    GRPSUB_SUBC_VRAM_ERRINJ_ECC_S0_HI_VRAM_ERRINJ_ECC_S0_HI =0;
    /* lock configuration registers*/
    GRPSUB_SUBC_VRAM_LOCKUNLOCK =0x5651F763;
    while (GRPSUB_SUBC_VRAM_LOCKSTATUS == 1)
    {

```

```

}

/*Clear hold bit for NMI21*/
IRC0_NMIHC = 0x15U;

}

FN_NMI_DEFINE_END()

```

In the NMI handler, the error injection register is cleared to stop pseudo ECC errors and the NMI is cleared. For clearing error injection, follow the same flow as that of enabling error injection. Instead of setting the bit, clear the bit in the error injection register.

In Figure 14 and Figure 15, you can observe that the NMI is triggered after a read is done to the VRAM address.

Figure 14. 1-Bit-Error Injection for VRAM

```

static void ECC_Test()
{
    /*Unlock the VRAM registers*/
    GRPSUB_SUBC_VRAM_LOCKUNLOCK=0x691DB936;
    while(GRPSUB_SUBC_VRAM_LOCKSTATUS == 1);

    /*Write 512 to VRAM Select register to enable ECC for entire VRAM region*/
    GRPSUB_SUBC_VRAM_SRAM_SELECT = 512;

    /*Inject 1 bit error by setting one bit in error injection register.*/
    GRPSUB_SUBC_VRAM_ERRINJ_ECC_SO_LO_VRAM_ERRINJ_ECC_SO_LO = 0x000000001ul;
    GRPSUB_SUBC_VRAM_ERRINJ_ECC_SO_HI_VRAM_ERRINJ_ECC_SO_HI = 0x00;

    /*Enable VRAM interrupts for error detection*/
    GRPSUB_SUBC_VRAMINTERRUPTENABLE = 1;

    /*Lock the VRAM registers*/
    GRPSUB_SUBC_VRAM_LOCKUNLOCK = 0x5651f763;
    while(GRPSUB_SUBC_VRAM_LOCKSTATUS == 1);

    /*Write a dummy data to a VRAM address {0x50000000}*/
    *((volatile uint32_t*)0x50000000) = 0x25;
    DSE();
    /*Read the VRAM address*/
    Read_vram* *((volatile uint32_t*)0x50000000)-Read_vram;

}

```


Figure 15. NMI Handler for 1-bit VRAM Error

```

FN_NMI_DEFINE_BEGIN(test_ECC, INTERRUPTS_NMI_NUMBER_21)
{
    /*unlock the key register for VRAM*/
    GRPSUB_SUBC_VRAM_LOCKUNLOCK=0x691DB936;
    while (GRPSUB_SUBC_VRAM_LOCKSTATUS == 1)
    {
    }
    /* clear the injected error*/
    GRPSUB_SUBC_VramInterruptClear =0x7U;
    GRPSUB_SUBC_VRAM_ERRINJ_ECC_SO_LO_VRAM_ERRINJ_ECC_SO_LO = 0x00000000u1;
    GRPSUB_SUBC_VRAM_ERRINJ_ECC_SO_HI_VRAM_ERRINJ_ECC_SO_HI =0;

    /* lock configuration registers*/
    GRPSUB_SUBC_VRAM_LOCKUNLOCK =0x5651F763;
    while (GRPSUB_SUBC_VRAM_LOCKSTATUS == 1)
    {
    }

    /*Clear hold bit for NMI21*/
    IRCO_NMIHC = 0x15U;
}
FN_NMI_DEFINE_END()

```

Once error injection and NMI are cleared, NMI is not again triggered until there is a bit flip in the VRAM memory.

3.2.2 2-Bit Error Injection and Handling NMI

The following code snippet enables two-bit error injection in VRAM memory:

```

static void ECC_Test()
{
    /*Unlock the VRAM registers*/
    GRPSUB_SUBC_VRAM_LOCKUNLOCK=0x691DB936;
    while (GRPSUB_SUBC_VRAM_LOCKSTATUS == 1);
    /*Write 512 to VRAM Select register to enable ECC for entire VRAM region*/

```



```
GRPSUB_SUBC_VRAM_SRAM_SELECT = 512;

/*Inject 2-bit error by setting one bit in error injection register. */
GRPSUB_SUBC_VRAM_ERRINJ_ECC_S0_LO_VRAM_ERRINJ_ECC_S0_LO = 0x000000011u1;
GRPSUB_SUBC_VRAM_ERRINJ_ECC_S0_HI_VRAM_ERRINJ_ECC_S0_HI = 0x00;

/*Enable VRAM interrupts for error detection*/
GRPSUB_SUBC_VRAM_INTERRUPTENABLE = 1;

/*Lock the VRAM registers*/
GRPSUB_SUBC_VRAM_LOCKUNLOCK = 0x5651f763;
while (GRPSUB_SUBC_VRAM_LOCKSTATUS == 1);

/*Write a dummy data to a VRAM address (0x50000000) */
*((volatile uint32_t*)0x50000000) = 0x25;

DSB();

/*Read the VRAM address*/
Read_vram= *((volatile uint32_t*)0x50000000)-Read_vram;
}
```

As soon as the VRAM location is read, NMI 20 is triggered, which shall be handled in the user application. A sample code for handling this NMI is shown below:

```
FN_NMI_DEFINE_BEGIN (test_ECC2bit, INTERRUPTS_NMI_NUMBER_20)
{
    GRPSUB_SUBC_VRAM_LOCKUNLOCK=0x691DB936;
    while (GRPSUB_SUBC_VRAM_LOCKSTATUS == 1)
    {
    }
}
```



```
/* clear the injected error*/
GRPSUB_SUBC_VRamInterruptClear =0x7U;
GRPSUB_SUBC_VRAM_ERRINJ_ECC_S0_LO_VRAM_ERRINJ_ECC_S0_LO = 0x00000000u1;
GRPSUB_SUBC_VRAM_ERRINJ_ECC_S0_HI_VRAM_ERRINJ_ECC_S0_HI =0;

/* lock configuration registers*/
GRPSUB_SUBC_VRAM_LOCKUNLOCK =0x5651F763;
while (GRPSUB_SUBC_VRAM_LOCKSTATUS == 1)
{
}
/*Clear hold bit for NMI20*/
IRC0_NMIHC = 0x14U;
}
FN_NMI_DEFINE_END()
```

In the NMI handler, the error injection register is cleared to stop pseudo ECC errors and the NMI is cleared. For clearing the error injection register, follow the same flow as that of enabling error injection. Instead of setting the bit, clear the bit in the error injection register.

In Figure 16 and Figure 17, you can observe that the NMI is triggered after a read is done to the VRAM address.

Figure 16. 2-Bit Error Injection for VRAM

```
static void ECC_Test()
{
    /*Unlock the VRAM registers*/
    GRPSUB_SUBC_VRAM_LOCKUNLOCK=0x691DB936;
    while (GRPSUB_SUBC_VRAM_LOCKSTATUS == 1);

    /*Write 512 to VRAM Select register to enable ECC for entire VRAM region*/
    GRPSUB_SUBC_VRAM_SRAM_SELECT = 512;

    /*Inject 2 bit error by setting one bit in error injection register.*/
    GRPSUB_SUBC_VRAM_ERRINJ_ECC_SO_LO_VRAM_ERRINJ_ECC_SO_LO = 0x000000011u1;
    GRPSUB_SUBC_VRAM_ERRINJ_ECC_SO_HI_VRAM_ERRINJ_ECC_SO_HI = 0x00;

    /*Enable VRAM interrupts for error detection*/
    GRPSUB_SUBC_VRAMINTERRUPTENABLE = 1;

    /*Lock the VRAM registers*/
    GRPSUB_SUBC_VRAM_LOCKUNLOCK = 0x5651f763;
    while (GRPSUB_SUBC_VRAM_LOCKSTATUS == 1);

    /*Write a dummy data to a VRAM address (0x50000000)*/
    *((volatile uint32_t*)0x50000000) = 0x25;
    DSB();
    /*Read the VRAM address*/
    Read_vram= *((volatile uint32_t*)0x50000000)-Read_vram;

}
```

Figure 17. NMI Handler for 2-bit VRAM Error

```

1  FN_NMI_DEFINE_BEGIN (test_ECC2bit, INTERRUPTS_NMI_NUMBER_20)
2  {
3
4
5  GRPSUB_SUBC_VRAM_LOCKUNLOCK=0x691DB936;
6  *   while (GRPSUB_SUBC_VRAM_LOCKSTATUS == 1)
7      {
8      }
9      /* clear the injected error*/
10 *   GRPSUB_SUBC_VramInterruptClear =0x70;
11 *   GRPSUB_SUBC_VRAM_ERRINJ_ECC_SO_LO_VRAM_ERRINJ_ECC_SO_LO = 0x00000000ul;
12 *   GRPSUB_SUBC_VRAM_ERRINJ_ECC_SO_HI_VRAM_ERRINJ_ECC_SO_HI =0;
13
14
15      /* lock configuration registers*/
16 *   GRPSUB_SUBC_VRAM_LOCKUNLOCK =0x5651F763;
17 *   while (GRPSUB_SUBC_VRAM_LOCKSTATUS == 1)
18     {
19     }
20
21      /*Clear hold bit for NMI0*/
22 *   IRCO_NMIHC = 0x140;
23 }
24 FN_NMI_DEFINE_END()

```

4. ECC for Work Flash

Work Flash adds an 8-bit error check code for every 64 bits, so that it can detect and correct 1-bit errors and detect 2-bit errors. The ECC feature can be turned ON/OFF by configuring the ECCOFF bit in the WFCFG_ECR register.

1-bit error is correctable and triggers the interrupt IRQ 20. 2-bit error results in a bus error leading to a data abort.

Figure 18. IRQ for 1-Bit Work Flash Error

1. IRQ Map

This section shows list of interrupt vector.

This list shows the assignment of interrupt vectors/interrupt control registers.

Vector of not-implemented function is not supported. See the chapter of function list.

IRQ No.	Detail	IRQ Priority Register	Vector Address Register
0	Reserved	-	-
1	System Control Status	IRC0_IRQPL0 : IRQPL1	IRC0_IRQVA1
2	HW-WDT Pre-warning	IRC0_IRQPL0 : IRQPL2	IRC0_IRQVA2
3	SW-WDT Pre-warning	IRC0_IRQPL0 : IRQPL3	IRC0_IRQVA3
4 to 7	Reserved	-	-
8	TCFLASH RDY, Hang up, Single Bit Error	IRC0_IRQPL2 : IRQPL8	IRC0_IRQVA8
9	Reserved	-	-
10	Work FLASH Hang up	IRC0_IRQPL2 : IRQPL10	IRC0_IRQVA10
11 to 13	Reserved	-	-
14	System RAM Single Bit Error	IRC0_IRQPL3 : IRQPL14	IRC0_IRQVA14
15	Backup RAM / CAN FD RAM(ch.0,1,5,6) Single Bit Error	IRC0_IRQPL3 : IRQPL15	IRC0_IRQVA15
16	IRC Vector Address RAM Single Bit Error	IRC0_IRQPL4 : IRQPL16	IRC0_IRQVA16
17 to 19	Reserved	-	-
20	Work FLASH RDY, Write Enable Release, Single Bit Error	IRC0_IRQPL5 : IRQPL20	IRC0_IRQVA20

4.1 ECC Test Function

To test the ECC functionality, the ECC logic has a test function to insert an error into the data and ECC read from the flash memory.

4.2 ECC Error Injection Related Register Configuration

Table 4. Work Flash ECC Registers

Abbreviated Register Name	Register Name	Register Details
WFCFG_CPR	Work Flash Configuration Protection Key Register	Used to protect the following registers from unintended writing: (WFCFG_CR),(WFCFG_ECR),(WFCFG_DBEIR), (WFCFG_EEIR).
WFCFG_ECR	Work Flash ECC Control Register	This register is used to control the operation of the ECC logic.
WFCFG_DBEIR	Work Flash Data Bit Error Injection Register	This register is used to perform an ECC logic operation test by injecting errors into the data bits read from Flash memory.
WFCFG_EEIR	Work Flash ECC Bit Error Injection Register	This register is used to perform an ECC logic operation test by injecting errors into the ECC bits read from Flash memory.
WFCFG_SECIR	Work Flash SEC Interrupt Register	This register contains the status flags, enable bits, and clear bits related to 1-bit error correction interrupts.
WFCFG_EEAR	Work Flash ECC Error Address Register	This register retains the address at which a 1-bit error was detected during reading. If the 1-bit error was detected multiple times, the register retains the address at which the error was last detected.
WFCFG_UCEAR	Work Flash Uncorrectable Error Address Register	This register retains the address at which an uncorrectable error was detected during reading. If the uncorrectable error was detected multiple times, the register retains the address at which the error was last detected.

For performing the ECC Error injection tests, the following procedure is performed:

- a) Unlock WFLASH ECC configuration interface.
- b) Perform the required configuration.
- c) Repeat a) and b) for the required registers.
- d) Read the Work Flash memory.

Notice the error response corresponding to the configuration.



4.3 1-Bit Error Injection and Handling IRQ

Consider the following lines of code:

```
void Test_WFLASH_ERRdata_1bit(void)
{
    if (WFCFG_ECR_ECCOFF != 0x0) //Enable ECC for Mirror 1 and Mirror 4 enabled if not enabled
    {

        WFCFG_CPR = WFLASH_KEY_UNLOCK; // Unlock WFLASH configuration registers

        while (WFCFG_CPR != 0xFFFFFFFF)
        {
        }

        WFCFG_ECR_ECCOFF = 0x0;

    }

    WFCFG_SECIR_SECIE =1; // ECC interrupt enable

    WFCFG_CPR = WFLASH_KEY_UNLOCK; // Unlock WFLASH configuration registers
    while (WFCFG_CPR != 0xFFFFFFFF)
    {
    }

    WFCFG_DBEIR_L=0x0001;
```




```
data_wflash_read_data = *((volatile uint32_t *)0x0E000000); // reading workflash location  
  
}
```

In this example, Bit 0 is configured for data flipping. This will cause 1-bit error detection and correction and trigger the corresponding interrupt (IRQ 20). The following is a sample interrupt handler:

```
/* ISR for the 1-bit ECC detection and correction by WFLASH */  
FN_IRQ_DEFINE_BEGIN(WFLASH_ECC_Test, INTERRUPTS_IRQ_NUMBER_20)  
{  
  
    WFLASH_Error_Address = WFCFG_EEAR; // read error address  
  
    if(WFCFG_SECIR_SECINT == 1) // read error flag.  
        WFCFG_SECIR_SECIC = 1; // Clear error flag.  
  
}  
FN_IRQ_DEFINE_END()
```

Figure 19 and Figure 20 show that the interrupt gets triggered after a Work Flash read.

Figure 19. 1-Bit Work Flash Error Injection

```

void Test_WFLASH_ERRdata_1bit(void)
{
    • if(WFCFG_ECR_ECCOFF != 0x0) //Enable ECC for Mirror 1 and Mirror 4 enabled if not enabled
      {
    •     WFCFG_CPR = WFLASH_KEY_UNLOCK; // Unlock WFLASH configuration registers
    •     while (WFCFG_CPR != 0xFFFFFFFF)
      {
    •     }
    •     WFCFG_ECR_ECCOFF = 0x0;
    •     }
    •     WFCFG_SECIR_SECIE =1; // ECC interrupt enable
    •     WFCFG_CPR = WFLASH_KEY_UNLOCK; // Unlock WFLASH configuration registers
    •     while (WFCFG_CPR != 0xFFFFFFFF)
      {
    •     }
    •     WFCFG_DBEIR_L=0x0001;
    •     data_wflash_read_data = *((volatile uint32_t *)0x0E000000); // reading workflash location
    •     }
}

```

Figure 20. IRQ Handler for 1-Bit Work Flash Error

```

/* ISR for the 1-bit ECC detection and correction by WFLASH */
FN_IRQ_DEFINE_BEGIN(WFLASH_ECC_Test, INTERRUPTS_IRQ_NUMBER_20)
{
    • WFLASH_Error_Address = WFCFG_EEAR; // read error address
    • if( WFCFG_SECIR_SECINT == 1)// read error flag.
    •     WFCFG_SECIR_SECIC = 1;// Clear error flag.
    • }
    • FN_IRQ_DEFINE_END()
}

```

Here is an example for 2-bit ECC data error for Work Flash:



```
void Test_WFLASH_ERRdata_2bit(void)
{
    if(WFCFG_ECR_ECCOFF != 0x0) //Enable ECC for Mirror 1 and Mirror 4 enabled if not enabled
    {

        WFCFG_CPR = WFLASH_KEY_UNLOCK; // Unlock WFLASH configuration registers.

        while (WFCFG_CPR != 0xFFFFFFFF)
        {
        }

        WFCFG_ECR_ECCOFF = 0x0;

    }

    // Configure Double-Bit Error

    WFCFG_CPR = WFLASH_KEY_UNLOCK; // Unlock WFLASH configuration registers
    while (WFCFG_CPR != 0xFFFFFFFF)
    {
    }

    WFCFG_DBEIR_L=0x0001; // Configure lower 32 bit of error injection register

    WFCFG_CPR = WFLASH_KEY_UNLOCK; // Unlock WFLASH configuration registers
```



```
while (WFCFG_CPR != 0xFFFFFFFF)
{
}

WFCFG_DBEIR_H=0x0001;    // Configure upper 32 bit of error injection register

data_wflash_read_data = *((volatile uint32_t *)0x0E000000); // reading workflash location.
}
```

In this example, Bit 0 and Bit 32 are configured for data flipping. This will cause 2-bit error detection resulting in bus error leading to data abort.

A sample for reading the error status inside the data abort handler follows:

```
if (WFCFG_BERR_DED ==1) // Check for 2 bit ECC error.
{
    WFLASH_Error_Address_2bit= WFCFG_UCEAR; // read error register.
    WFCFG_BERRCLR_DEDCLR =1; // clear error status.
}
```

Figure 21 and Figure 22 show that the interrupt gets triggered after a Work Flash read.

Figure 21. 2-Bit Work Flash Error Injection

```

void Test_WFLASH_ERRdata_2bit(void)
{
    if(WFCFG_ECR_ECCOFF != 0x0) //Enable ECC for Mirror 1 and Mirror 4 enabled if not enabled
    {
        WFCFG_CPR = WFLASH_KEY_UNLOCK; // Unlock WFLASH configuration registers.
        while (WFCFG_CPR != 0xFFFFFFFF)
        {
        }
        WFCFG_ECR_ECCOFF = 0x0;
    }
    // Configure Double-Bit Error
    WFCFG_CPR = WFLASH_KEY_UNLOCK; // Unlock WFLASH configuration registers
    while (WFCFG_CPR != 0xFFFFFFFF)
    {
    }
    WFCFG_DBEIR_L=0x0001; // Configure lower 32 bit of error injection register
    WFCFG_CPR = WFLASH_KEY_UNLOCK; // Unlock WFLASH configuration registers
    while (WFCFG_CPR != 0xFFFFFFFF)
    {
    }
    WFCFG_DBEIR_H=0x0001; // Configure upper 32 bit of error injection register
    data_wflash_read_data = *((volatile uint32_t *)0x0E000000); // reading workflash location.
}

```

Figure 22. Data Abort for 2-Bit Work Flash Error

```

static void DefaultDataAbortExceptionHandler(void)
{
    // IRQs are disabled now.
    // Please refer to the ARM documentation on how to handle this exception.

    volatile uint32_t WFLASH_Error_Address_2bit;

    // For debugging purpose only
    #ifdef DEBUG
    // Collect some information about the cause of the data abort
    volatile uint32_t u32DataFaultStatus = MRC( 15, 0, 5, 0, 0 );
    volatile uint32_t u32AuxDataFaultStatus = MRC( 15, 0, 5, 1, 0 );
    volatile uint32_t u32DataFaultAddress = MRC( 15, 0, 6, 0, 0 );
    #endif // DEBUG

    if(WFCFG_BERR_DED ==1) // Check for 2 bit ECC error.
    {
        WFLASH_Error_Address_2bit= WFCFG_UCEAR; // read error register.
        WFCFG_BERRCLR_DEDCLR =1; // clear error status.
    }

    while (1)
    {
        NOP();
    }
}

```

Because there is a data abort, you can observe the values in DFSR and DFAR registers for further analysis.

Figure 23. DFSR Register Values During Data Abort

DFSR	0x00001008
Ext	0x1
WnR	0x0
FS[4]	0x0
FS[3:0]	0x8
IFSR	0x00000000
ADFSR	0x00000000
AIFSR	0x00000000
DFAR	0x0e000000

Table 5. DFSR Fault Status During Abort

Bits	Description	Value	Inference
FS[4,3:0]	DFSR - Fault Status	0b01000	Synchronous External Abort.
[31:0]	DFAR	0xe0000000	Address where Work Flash is read.

An error can also be injected by the destruction of the ECC area. The following example codes can be used for that.

4.3.1 1-bit ECC Area Error in Work Flash

```
void Test_WFLASH_ERRECC_1bit(void)
{
    if(WFCFG_ECR_ECCOFF != 0x0) //Enable ECC for Mirror 1 and Mirror 4 enabled if not enabled
    {

        WFCFG_CPR = WFLASH_KEY_UNLOCK; // Unlock WFLASH configuration registers

        while (WFCFG_CPR != 0xFFFFFFFF)
        {
        }
    }
}
```



```
WFCFG_ECR_ECCOFF = 0x0;

}

WFCFG_SECIR_SECIE =1; // ECC interrupt enable

WFCFG_CPR = WFLASH_KEY_UNLOCK; // Unlock WFLASH configuration registers
while (WFCFG_CPR != 0xFFFFFFFF)
{
}

WFCFG_EEIR=0x0008;

    data_wflash_read_data = *((volatile uint32_t *)0x0E000000); // reading workflash location
}
```

2-bit ECC area Error in Work Flash:

```
void Test_WFLASH_ERRECC_2bit(void)
{
    if(WFCFG_ECR_ECCOFF != 0x0) //Enable ECC for Mirror 1 and Mirror 4 enabled if not enabled
    {

        WFCFG_CPR = WFLASH_KEY_UNLOCK; // Unlock WFLASH configuration registers.
    }
}
```




```
while (WFCFG_CPR != 0xFFFFFFFF)
{
}

WFCFG_ECR_ECCOFF = 0x0;

}

// Configure Double-Bit Error

WFCFG_CPR = WFLASH_KEY_UNLOCK; // Unlock WFLASH configuration registers
while (WFCFG_CPR != 0xFFFFFFFF)
{
}

WFCFG_EEIR=0x0003; // Configure lower 32 bit of error injection register

data_wflash_read_data = *((volatile uint32_t *)0x0E000000); // reading workflash location.
}
```

5. ECC for TCMRAM

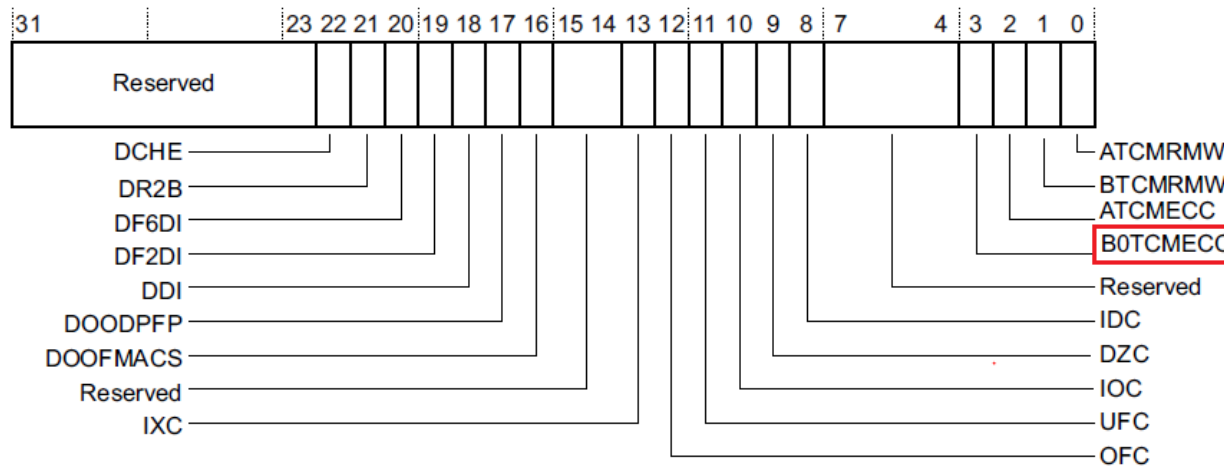
- Cortex-R5F - the CPU core in Traveo MCU has a provision for an ECC check for the TCM RAM. It supports 1-bit error correction and 2-bit error detection.
- Because the following access to uninitialized area of the TCRAM causes an ECC error, 32-bit initialization is necessary before using it.
 - a. 8- or 16-bit write access.
 - b. Read access.

- The Cortex-R5F performs an ECC check for the TCRAM in units of 32 bits.
- For 32- or 64-bit write operation, it adds ECC to data and then writes it to the TCRAM.
- For 8- or 16-bit write operation, it performs Read-Modify-Write access so that the correct ECC can be generated. An ECC check is also performed during the read operation in case of Read-Modify-Write access.
- TCRAM has provision to insert an error in the data read from the TCRAM so that the 32-bit ECC function of the Cortex-R5F can be tested. ECC error generation can be either by data area destruction or ECC area destruction.

5.1 Configuration of Error Correction

TCRAM 1-bit error correction can be enabled or disabled by using c15, Secondary Auxiliary Control Register.

Figure 24. c15 Auxiliary Control Register bit arrangements



BTCMECC Correction for internal ECC logic on BTCM ports

0 = Enabled. This is the reset value.

1 = Disabled.

When ECC error correction is disabled, an abort is generated.

5.2 TCMRAM ECC Behavior

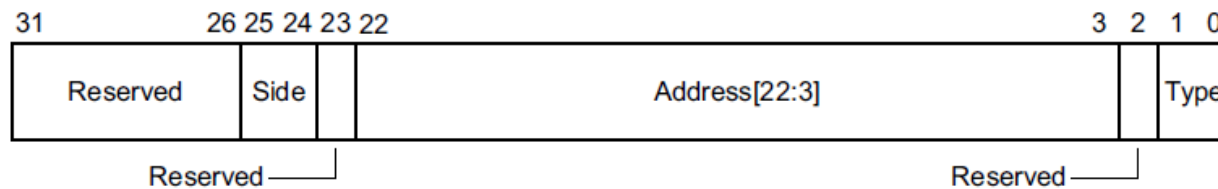
The following table indicates the behavior on ECC error detection for different configurations and errors.

Table 6. TCMRAM ECC Error Detection for Different Configurations and Errors

S.No	C15, Secondary Auxiliary Control register- bit BTCMECC	1-bit Error	2-bit Error	Access TCM/ AXI	Error path	Error Status Register	Error Address Register
1	Enabled	Yes		TCM	Corrected	CFLR	CFLR
2	Enabled		Yes	TCM	Data Abort	DFSR - Synchronous ECC error	DFAR
3	Disabled	Yes		TCM	Data Abort	CFLR DFSR - Synchronous ECC error	CFLR DFAR
4	Disabled		Yes	TCM	Data Abort	DFSR - Synchronous ECC error	DFAR
5	Enabled	Yes		AXI	Corrected	CFLR	CFLR
6	Enabled		Yes	AXI	Data Abort	DFSR - Synchronous external abort	DFAR
7	Disabled	Yes		AXI	Corrected	CFLR	CFLR
8	Disabled		Yes	AXI	Data Abort	DFSR - Synchronous external abort	DFAR

5.3 Correctable Fault Location Register - TCM, Bit Assignments

Figure 25. CFLR Register Bit Arrangements



[25:24] Side Indicates the source of the error:

0b01 = ATCM

0b10 = BTCM

[22:3] Address

Indicates the bits [22:3] of the address in the TCM where the error occurred. Address [22:3].

[1:0] Type Indicates the type of access that caused the error:

0b00 = Instruction

0b01 = Data

0b10 = AXI Slave

Whenever a correctable error occurs in TCM RAM region, the following values are read from CFLR.

Side – It will be always 2 (indicating BTCM where TCRAM is connected)

Type –

- “0b01” – On access via TCM interface
- “0b10” – On access via AXI interface

5.4 ECC Error Injection Related Register Configuration

Table 7. TCMRAM ECC Registers

Abbreviated Register Name	Register Name	Register Details
TRCFGn_TCMCFG0	TCRAM IF Configuration Register 0	This register has seven bits for ERRECC data for the BTCM port, two bits for setting the number of waits, and the LOCKSTATUS bit that indicates the locked state or unlocked state of the TCRAM IF configuration register.
TRCFGn_TCMCFG1	TCRAM IF Configuration Register 1	This register is used to insert an error in data read from the TCRAM for testing the ECC function of the Cortex-R5F BTCM port.
TRCFGn_TCMUNLOCK	TCRAM IF Unlock Register	This register locks or unlocks write access to the TCRAM interface registers. Writing the correct unlock value (0xACC55ECC) to this register enables write access to the registers. After setting registers, writing the correct lock value (0x5ECCB10C) to this register disables write access to the registers.

Do the following to perform the ECC Error injection tests:

1. Unlock the TCRAM interface.
2. Perform the required configuration.
3. Lock the TCRAM interface.
4. Read the TCRAM memory.
5. Observe the error response corresponding to the configuration.

5.5 Sample Codes for ECC Injection Tests

Note the following important points before doing the TCMRAM ECC Injection tests:

- Stack and debugger memory might have been allocated in TCRAM. In such cases, when the error injection feature is configured for TCRAM, reads from stack or debugger memory can affect the address read from CFLR because CFLR retains the address where the last correctable error was detected.
- If cache is enabled during tests through the AXI interface, it can also affect the reported error address.

5.5.1 1-bit ECC Data Error

Consider the following function:

```
void Test_TCRAM_ERRdata_1bit(void)
{

    uint32_t test_read_data =0;

    data_tcram_read_data[10] = 10; // writing to an address in TCRAM.

    TRCFG0_TCMUNLOCK = TCMRAM_KEY_UNLOCK; // Unlock TCMRAM configuration registers

    while (TRCFG0_TCMCFG0_LOCKSTATUS == 1)
    {
```



```
}  
  
TRCFG0_TCMCFG1_ERRBIT=0x0008;    // Configure Single-Bit Data Error  
  
TRCFG0_TCMUNLOCK = TCMRAM_KEY_LOCK;    // Lock TCMRAM configuration registers  
  
while (TRCFG0_TCMCFG0_LOCKSTATUS == 0)  
{  
}  
  
test_read_data = data_tcram_read_data[10]; // read TCRAM data  
  
}
```

In this example, Bit 3 is configured for data flipping.

`data_tcram_read_data` array is placed in the TCRAM TCM region.

The expected result is that the single-bit error is corrected and the corresponding address is reported in the CFLR register.

Figure 26 and Figure 27 show the value of the CFLR register after the TCRAM is read:

Figure 26. TCMRAM 1-Bit Error Injection

```
void Test_TCRAM_ERRdata_1bit(void)
{
    • uint32_t test_read_data =0;
    • data_tcram_read_data[10] = 10; // writing to an address in TCRAM.
    • TRCFG0_TCMUNLOCK = TCMRAM_KEY_UNLOCK; // Unlock TCMRAM configuration registers
    • while (TRCFG0_TCMCFG0_LOCKSTATUS == 1)
      {
      }
    • TRCFG0_TCMCFG1_ERRBIT=0x0008; // Configure Single-Bit Data Error
    • TRCFG0_TCMUNLOCK = TCMRAM_KEY_LOCK; // Lock TCMRAM configuration registers
    • while (TRCFG0_TCMCFG0_LOCKSTATUS == 0)
      {
      }
    • test_read_data = data_tcram_read_data[10]; // read TCRAM data
    STOPPED }

```

Figure 27. CFLR Register Value

<input type="checkbox"/> CP15 C15	
<input checked="" type="checkbox"/> Secondary_Auxiliary_Control	0x00400000
<input checked="" type="checkbox"/> Normal_AXI_Peripheral_Interface_Region	0xb0000045
<input checked="" type="checkbox"/> Virtual_AXI_Peripheral_Interface_Region	0x00000000
<input checked="" type="checkbox"/> AHB_Peripheral_Interface_Region	0xb4000045
<input checked="" type="checkbox"/> nVAL_IRQ_Enable_Set	0x00000000
<input checked="" type="checkbox"/> nVAL_FIQ_Enable_Set	0x00000000
<input checked="" type="checkbox"/> nVAL_Reset_Enable_Set	0x00000000
<input checked="" type="checkbox"/> nVAL_Debug_Request_Enable_Set	0x00000000
<input checked="" type="checkbox"/> nVAL_IRQ_Enable_Clear	0x00000000
<input checked="" type="checkbox"/> nVAL_FIQ_Enable_Clear	0x00000000
<input checked="" type="checkbox"/> nVAL_Reset_Enable_Clear	0x00000000
<input checked="" type="checkbox"/> nVAL_Debug_Request_Enable_Clear	0x00000000
<input checked="" type="checkbox"/> Build_Options_1	0x00800000
<input checked="" type="checkbox"/> Build_Options_2	0x0ebfc1c2
<input checked="" type="checkbox"/> Pin_Options	0x00000001
<input checked="" type="checkbox"/> Correctable_Fault_Location	0x02000029
<input type="checkbox"/> Invalidate_All_Data_Cache	0x00000000
<input type="checkbox"/> Cache_Size_Override	0x00000000

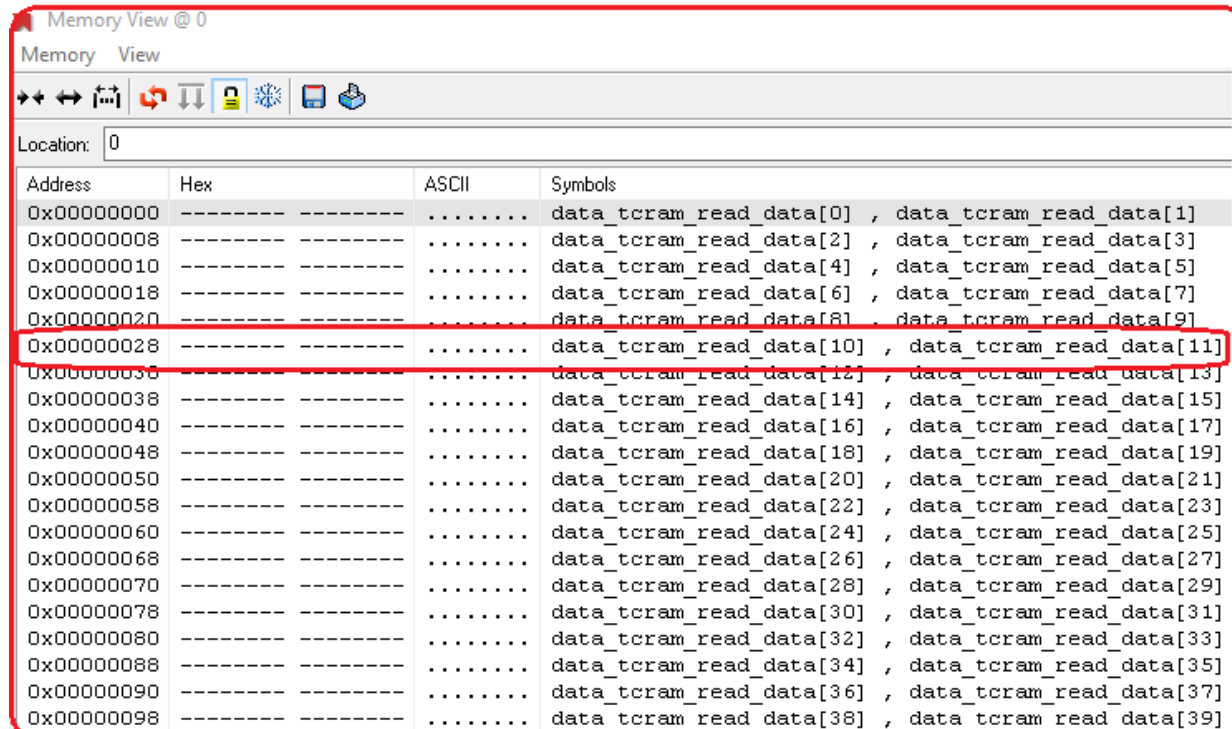
The value of CFLR is 0x2000029.

Table 8. Decoding of CFLR

Bits	Description	Value	Inference
[31:0]	CFLR	0x2000029	
[25:24]	Side	0b10	BTM - source of error
[22:3]	Address [22:3]	0x05	Address [20:0] - 0x28. Bits {2:0} is considered as 0.
[1:0]	Type	0b01	Data

The address of the data_tcram_read_data[10] register is 0x00000028. This matches with the CFLR address field.

Figure 28. TCMRAM Data in Memory View of Debugger

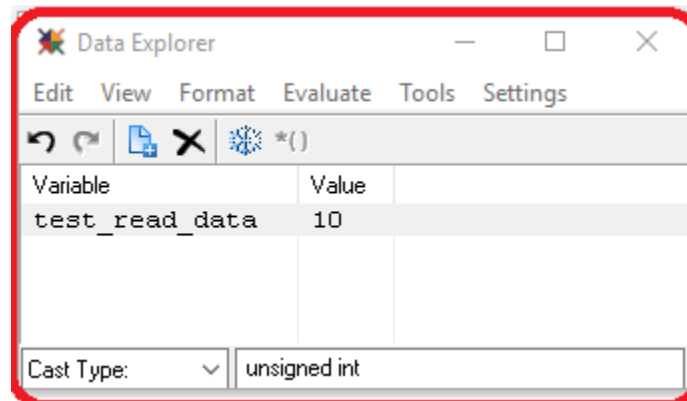


Address	Hex	ASCII	Symbols
0x00000000	-----	data_tcram_read_data[0] , data_tcram_read_data[1]
0x00000008	-----	data_tcram_read_data[2] , data_tcram_read_data[3]
0x00000010	-----	data_tcram_read_data[4] , data_tcram_read_data[5]
0x00000018	-----	data_tcram_read_data[6] , data_tcram_read_data[7]
0x00000020	-----	data_tcram_read_data[8] , data_tcram_read_data[9]
0x00000028	-----	data_tcram_read_data[10] , data_tcram_read_data[11]
0x00000030	-----	data_tcram_read_data[12] , data_tcram_read_data[13]
0x00000038	-----	data_tcram_read_data[14] , data_tcram_read_data[15]
0x00000040	-----	data_tcram_read_data[16] , data_tcram_read_data[17]
0x00000048	-----	data_tcram_read_data[18] , data_tcram_read_data[19]
0x00000050	-----	data_tcram_read_data[20] , data_tcram_read_data[21]
0x00000058	-----	data_tcram_read_data[22] , data_tcram_read_data[23]
0x00000060	-----	data_tcram_read_data[24] , data_tcram_read_data[25]
0x00000068	-----	data_tcram_read_data[26] , data_tcram_read_data[27]
0x00000070	-----	data_tcram_read_data[28] , data_tcram_read_data[29]
0x00000078	-----	data_tcram_read_data[30] , data_tcram_read_data[31]
0x00000080	-----	data_tcram_read_data[32] , data_tcram_read_data[33]
0x00000088	-----	data_tcram_read_data[34] , data_tcram_read_data[35]
0x00000090	-----	data_tcram_read_data[36] , data_tcram_read_data[37]
0x00000098	-----	data_tcram_read_data[38] , data_tcram_read_data[39]

Figure 29 indicates the data read from the memory. The corrected data is the same as what was written before is read back.

```
data_tcram_read_data[10] = 10; // writing to an address in TCRAM.
```

Figure 29. Correct TCMRAM Data



5.5.2 2-bit ECC Data Error

Consider the following function:

```
void Test_TCRAM_ERRdata_2bit(void)
{
    uint32_t test_read_data =0;

    TRCFG0_TCMUNLOCK = TCMRAM_KEY_UNLOCK; // Unlock TCMRAM configuration registers

    while (TRCFG0_TCMCFG0_LOCKSTATUS == 1)
    {
    }

    TRCFG0_TCMCFG1_ERRBIT=0x0003; // Configure Double-Bit Data Error
```



```
TRCFG0_TCMUNLOCK = TCMRAM_KEY_LOCK;    // Lock TCMRAM configuration registers

while (TRCFG0_TCMCFG0_LOCKSTATUS == 0)
{
}

test_read_data = data_tcram_read_data[10]; // read TCRAM data

}
```

In this example, bits 0 and 1 are configured for data flipping, which corresponds to 2-bit error injection.

The `data_tcram_read_data` array is placed in the TCRAM TCM region. The expected result is that the double-bit error is detected and will result in a bus error leading to a data abort.

Following screenshots show program flow, DFSR and DFAR registers after TCRAM is read.

Figure 30. TCMRAM 2-bit Error Injection

```

void Test_TCRAM_ERRdata_2bit(void)
{
    • uint32_t test_read_data =0;
    • TRCFG0_TCMUNLOCK = TCMRAM_KEY_UNLOCK; // Unlock TCMRAM configuration registers
    • while (TRCFG0_TCMCFG0_LOCKSTATUS == 1)
      {
      }
    • TRCFG0_TCMCFG1_ERRBIT=0x0003; // Configure Double-Bit Data Error
    • TRCFG0_TCMUNLOCK = TCMRAM_KEY_LOCK; // Lock TCMRAM configuration registers
    • while (TRCFG0_TCMCFG0_LOCKSTATUS == 0)
      {
      }
    • test_read_data = data_tcram_read_data[10]; // read TCRAM data
    • }

```

Data execution has been caught by vector catch. The address 0xFFFF0010 indicates that data abort has occurred.

Figure 31. Data Abort During 2-bit TCMRAM Error

•	0xffff0000:	ea00045c	B	0x1170 (0xffff1178)
•	0xffff0004:	e51ff048	LDR	PC, [PC, -72] (0xfffeffc4)
•	0xffff0008:	e51ff048	LDR	PC, [PC, -72] (0xfffeffc8)
•	0xffff000c:	e51ff048	LDR	PC, [PC, -72] (0xfffeffc)
•	STOPPED 0xffff0010:	e51ff048	LDR	PC, [PC, -72] (0xfffeffd0)
•	0xffff0014:	e59fff74	LDR	PC, [PC, 0xf74] (0xffff0f90)
•	0xffff0018:	e51ff048	LDR	PC, [PC, -72] (0xfffeffd8)
•	0xffff001c:	e51ff428	LDR	PC, [PC, -0x428] (0xfffebfc)

You can observe the values in DFSR and DFAR registers for additional analysis:

Figure 32. DFSR and DFAR During Abort

[-] DFSR	0x00000409
ExT	0x0
WnR	0x0
FS[4]	0x1
FS[3:0]	0x9
[+] IFSR	0x00000000
[-] ADFSAR	0x00800000
CacheWay	0x0
Side	0x2
Recoverable_error	0x0
SideExt	0x0
Index	0x000
[+] AIFSR	0x00000000
DFAR	0x00000028

Table 9 DFSR and DFAR Inferences

Bits	Description	Value	Inference
FS[4,3:0]	DFSR - Fault Status	0b11001	Synchronous ECC Error
[31:0]	DFAR	0x00000028	Address where TCRAM is read
[20] [23:22]	ADFSR – SideExt [20] Side [23:22]	0b10	BTCM - source of error

An error can also be injected by the destruction of ECC area. The following example codes can be used for that.

5.5.3 1-bit ECC Area Error

```
void Test_TCRAM_ERRECC_1bit(void)
{
    uint32_t test_read_data =0;
```



```
data_tcram_read_data[10] = 10; // writing to an address in TCAM.

TRCFG0_TCMUNLOCK = TCMRAM_KEY_UNLOCK; // Unlock TCMRAM configuration registers

while (TRCFG0_TCMCFG0_LOCKSTATUS == 1)
{
}

TRCFG0_TCMCFG0_ERRECC=0x0008; // Configure Single-bit ECC area Error

TRCFG0_TCMUNLOCK = TCMRAM_KEY_LOCK; // Lock TCMRAM configuration registers

while (TRCFG0_TCMCFG0_LOCKSTATUS == 0)
{
}

test_read_data = data_tcram_read_data[10]; // read TCAM data

}
```

5.5.4 2-bit ECC Area Error

```
void Test_TCAM_ERRECC_2bit(void)
{

    uint32_t test_read_data =0;
```



```
TRCFG0_TCMUNLOCK = TCMRAM_KEY_UNLOCK; // Unlock TCMRAM configuration registers

while (TRCFG0_TCMCFG0_LOCKSTATUS == 1)
{
}

TRCFG0_TCMCFG0_ERRECC=0x0003; // Configure Double-bit ECC area Error

TRCFG0_TCMUNLOCK = TCMRAM_KEY_LOCK; // Lock TCMRAM configuration registers

while (TRCFG0_TCMCFG0_LOCKSTATUS == 0)
{
}

test_read_data = data_tcram_read_data[10]; // read TCRAM data
}
```

6. ECC for SYSRAM and Back-Up RAM

Like the ECC features discussed so far, System RAM (SYSRAM) and Backup RAM also have ECC support and ECC error injection features. The following examples illustrate these.

6.1 Sample Codes for ECC Injection Tests in SYSRAM

Ensure that the variable `data_sysram_read_data` is placed in the System RAM area.



6.1.1 1-bit ECC Data Error

```
void Test_SYSRAM_ERRdata_1bit(void)
{
    uint32_t test_read_data =0;

    SRCFG_KEY_UNLOCK = SYSRAM_KEY_UNLOCK; // Unlock SYRAM configuration registers

    while (SRCFG_CFG0_LOCK_STATUS == 1)
    {

    }

    if(SRCFG_ECCE_ECCEN != 1) //Enable ECC if not enabled. This bit can be modified only once in software
    {
        SRCFG_ECCE_ECCEN = 1;
    }

    SRCFG_INTE_SEC_INT_EN =1; // Enable ECC interrupt

    SRCFG_CFG1_ERRBIT=0x0001; // Configure Single-Bit Error

    SRCFG_KEY_UNLOCK = SYSRAM_KEY_LOCK; // Lock configuration registers

    while (SRCFG_CFG0_LOCK_STATUS == 0)
    {

    }
```




```
test_read_data = data_sysram_read_data[10]; // read SYSRAM data
```

```
}
```

In this example, Bit 0 is configured for data flipping. This will cause 1-bit error detection and correction and will trigger the corresponding interrupt. The following is a sample interrupt handler for the same.

```
/* ISR for the 1-bit ECC detection and correction by SYSRAM */
FN_IRQ_DEFINE_BEGIN (SYSRAM_ECC_Test, INTERRUPTS_IRQ_NUMBER_14)
{

    SRCFG_KEY_UNLOCK = SYSRAM_KEY_UNLOCK;
    while (SRCFG_CFG0_LOCK_STATUS == 1)
    {
    }

    SYSRAM_Error_Address = SRCFG_ERRADR; // read Single bit error address

    if(SRCFG_ERRFLG_SECFLG == 1) // read error flag.
    SRCFG_ERRFLG_SECCLR = 1; // Clear error flag.

    SRCFG_KEY_UNLOCK = SYSRAM_KEY_LOCK; // Lock configuration registers

    while (SRCFG_CFG0_LOCK_STATUS == 0)
    {
    }
}
```

```
}  
FN_IRQ_DEFINE_END ()
```

6.1.2 2-bit ECC Data Error

```
void Test_SYSRAM_ERRdata_2bit(void)  
{  
    uint32_t test_read_data =0;  
  
    SRCFG_KEY_UNLOCK = SYSRAM_KEY_UNLOCK; // Unlock SYSRAM configuration registers  
  
    while (SRCFG_CFG0_LOCK_STATUS == 1)  
    {  
  
        if(SRCFG_ECCE_ECCEN != 1) //Enable ECC if not enabled. This bit can be modified only once in software  
        {  
            SRCFG_ECCE_ECCEN = 1;  
        }  
  
        SRCFG_CFG1_ERRBIT=0x0003; // Configure Double-Bit Error  
  
        SRCFG_KEY_UNLOCK = SYSRAM_KEY_LOCK; // Lock configuration registers  
  
        while (SRCFG_CFG0_LOCK_STATUS == 0)  
        {
```

```
}  
  
test_read_data = data_sysram_read_data[10]; // read SYSRAM data  
  
}
```

In this example, Bit 0 and Bit 1 are configured for data flipping. This will cause 2-bit error detection resulting in a bus error leading to a data abort.

An error can also be injected by the destruction of ECC area.

6.2 Backup Ram (BURAM)

6.2.1 1-bit ECC Error

```
void Test_BURAM_ERR_1bit(void)  
{  
  
    BURIF_UNLOCK = BURAM_KEY_UNLOCK; // Unlock BURAM configuration registers  
  
    while (BURIF_STATUS_LOCKSTATUS == 1)  
    {  
  
    }  
  
    BURIF_EECSR_SEIE=0x1; //Enable Single-Bit Error Interrupt  
  
    BURIF_EFEAR_ERR_ADDR = 0x20; // Configure the address offset for ECC error
```



```
BURIF_EFECR_0_EI = 0x1;    // Configure Single-Bit Error

BURIF_EFECR_1_EY = 0x1; // Configure byte where error is to be introduced

BURIF_EFECR_2_FERR =1; // ECC psuedo error generation enable. Error gets inserted during the enabling of this
configuration

BURIF_UNLOCK = BURAM_KEY_LOCK; // Lock configuration registers

while (BURIF_STATUS_LOCKSTATUS == 0)
{
}

}
```

In this example, Bit 0 and Byte 0 are configured for pseudo ECC error. This will cause 1-bit error detection and correction and will trigger the corresponding interrupt. The following is a sample interrupt handler for the same.

```
/* ISR for the 1-bit ECC detection and correction by BURAM */
FN_IRQ_DEFINE_BEGIN(BURAM_ECC_Test, INTERRUPTS_IRQ_NUMBER_15)
{

    BURIF_UNLOCK = BURAM_KEY_UNLOCK;
    while (BURIF_STATUS_LOCKSTATUS == 1)
    {
    }
}
```



```
BURAM_Error_Address = BURIF_SEEAR_ERR_ADDR; // read error address

if( BURIF_EECSR_SEI == 1)// read error flag.
BURIF_EECSR_SEI = 0;// Clear error flag.

BURIF_UNLOCK = BURAM_KEY_LOCK; // Lock configuration registers

while (BURIF_STATUS_LOCKSTATUS == 0)
{
}

}
FN_IRQ_DEFINE_END()
```

6.2.2 2-bit ECC Error

```
void Test_BURAM_ERR_2bit(void)
{

BURIF_UNLOCK = BURAM_KEY_UNLOCK;// Unlock BURAM configuration registers

while (BURIF_STATUS_LOCKSTATUS == 1)
{
}
```



```
BURIF_EECSR_DEIE=0x1;    //Enable Configure Double-Bit Error

BURIF_EFEAR_ERR_ADDR = 0x20; // Configure the address offset for ECC error

BURIF_EFECSR_0_EI = 0x3;  // Configure Double-Bit Error

BURIF_EFECSR_1_EY = 0x1; // Configure byte where error is to be introduced

BURIF_EFECSR_2_FERR =1; // ECC psuedo error generation enable. Error gets inserted during the enabling of this
configuration

BURIF_UNLOCK = BURAM_KEY_LOCK; // Lock configuration registers

while (BURIF_STATUS_LOCKSTATUS == 0)
{
}

}
```

In this example, Bit 0, Bit 1, and Byte 0 are configured for pseudo ECC error. This will cause 2-bit error detection and will trigger the corresponding NMI. The following is a sample NMI handler for the same.

```
/* NMI for the 2-bit ECC detection by BURAM */
FN_NMI_DEFINE_BEGIN(BURAM_ECC_Test_2bit, INTERRUPTS_NMI_NUMBER_11)
{
```



```
BURIF_UNLOCK = BURAM_KEY_UNLOCK;
while (BURIF_STATUS_LOCKSTATUS == 1)
{
}

BURAM_Error_Address_2bit = BURIF_DEEAR; // read error address

if( BURIF_EECSR_DEI == 1)// read error flag.
BURIF_EECSR_DEI = 0;// Clear error flag.

BURIF_UNLOCK = BURAM_KEY_LOCK; // Lock configuration registers

while (BURIF_STATUS_LOCKSTATUS == 0)
{
}

}
FN_NMI_DEFINE_END()
```

7. Summary

This guide covered the ECC injection tests available in various Traveo memories. The hardware ECC logic itself can be enabled or disabled for these memories. Similarly, there are registers that would control the Interrupt generation in case of errors on these memories. Table 10 and Table 11 provide a summary of this.

Table 10. Summary of ECC for Various Memories in Traveo

Memory	ECC Logic (Default Control)	1-bit error interrupt default (When ECC logic is enabled, 1-bit errors are automatically corrected. Interrupt generation is optional setting for the user)	2-bit error interrupt/abort default
TCFLASH - AXI	Enabled (TCFCFGn_FECCCTRL)	Disabled (TCFCFG_FSECIR)	Enabled (abort) (always enabled when ECC logic is enabled)
TCFLASH - TCM	Enabled (Cortex-R5 ACTLR register)	Disabled (no specific interrupt flag, Cortex-R5 measurement counters can be used to trigger an event)	Enabled (abort) (always enabled when ECC logic is enabled)
WORKFLASH	Enabled (WFCFG_ECR)	Disabled (WFCFG_SECIR)	Enabled (abort) (always enabled when ECC logic is enabled)
TCMRAM	Enabled (Cortex-R5 ACTLR register)	Disabled (no specific interrupt flag, Cortex-R5 measurement counters can be used to trigger an event)	Enabled (abort) (always enabled when ECC logic is enabled)
SYSRAM	Enabled (SRCFG_ECCE)	Disabled (SRCFG_INTE)	Enabled (abort) (always enabled when ECC logic is enabled)
Back-Up RAM	Enabled (BURIF_EDPCR)	Disabled (BURIF_EECSR)	Disabled (interrupt) (BURIF_EECSR)
IRC Vector Address RAM	Enabled (always)	Enabled (Proper mapping of IRQ handler is up-to user)	Enabled (Proper mapping of NMI handler is up-to user)
VRAM	Disabled (Specify the sub-sections of the VRAM using VRAM_SRAM_SELECT)	Disabled (VRAMINTERRUPTENABLE)	Disabled (VRAMINTERRUPTENABLE)

Table 11. Summary of ECC for various memories in Traveo

Memory	ECC Scheme	Enabled by Default	Configurable	Failure Handling Single-Bit Error	Failure Handling Double-Bit Error
TCRAM (through BTCM)	32-bit ECC	Yes	Yes	ABORT (configurable)	ABORT

TCRAM (through BTCM)	32-bit ECC	Yes	Yes	CFLR	Bus error response
System RAM	32-bit ECC	Yes	Yes	IRQ	Bus error response
Backup RAM	8-bit ECC	Yes	Yes	IRQ	NMI
IRC vector RAM	32-bit ECC	Yes	No	IRQ	NMI
CAN RAM	32-bit ECC	Yes	Yes	IRQ	NMI and Bus error response
Video RAM	8-bit ECC	No	Yes	NMI	Bus error response
TCFLASH (through ATCM)	64-bit ECC	Yes	Yes	ABORT (configurable)	ABORT
TCFLASH (through AXI)	64-bit ECC	Yes	Yes	IRQ	Bus error response
WORKFLASH	32-bit ECC	Yes	Yes	IRQ	Bus error response

8. References

- [S6J3200 Series 32-bit Microcontroller Traveo™ Family](#)
- [32-bit Microcontroller Traveo™ Family S6J3200 Series Hardware Manual](#)
- [Traveo™ Family 32-Bit Microcontroller Platform Part Hardware Manual](#)
- [Cortex™ -R5 and Cortex-R5F Revision: r1p1 Technical Reference Manual](#)