NXP Semiconductors
Application Notes

Document Number: AN13109
Rev. 0, 12/2020

Enabling and Checking DDR ECC on S32V23x Device

by: NXP Semiconductors

Contents

1. Introduction

The DDR controller of the S32V23x family has built-in Error Checking and Correction (ECC) capabilities. ECC allows single bit errors to be corrected and two/three bits errors to be detected which increases the reliability of high frequency operation as well as improves data accuracy.

This application note provides an introduction to the ECC mechanism as well as guideline to configure and validate the ECC feature on DDR.

This document contains following sections:

- Error Correcting Code (ECC) on DDR: This section introduces features and calculation of the ECC on DDR.
- DDR ECC configuration: This section introduces how to configure the MEW registers to enable ECC on DDR.
- Fault injection and reaction: This section describes the basics of fault injection, how to inject and trigger ECC faults, how to verify ECC configuration on DDR and how to react to the fault.
- Configuration and verification examples: This section gives examples to configure and verify ECC mechanism on DDR.

The following abbreviations are used in the application note.

Table 1. **Acronyms and abbreviations**

Abbreviations	Description
MMDC	Multi-Mode DDR Controller
DDR	Double Data Rate
DRAM	Dynamic Random-Access Memory
MEW	MMDC ECC and Debug Watchpoint
ECC	Error Correcting Code
FCCU	Fault Collection and Control Unit
XOR	Exclusive Ored
XRDC	Extended Resource Domain Controller
SEC	Single error correction
DED	Double error detection
TED	Triple error detection

2. Error Correcting Code (ECC) on DDR

2.1. Features of ECC on DDR

As shown in *Figure 1* MMDC is a multi-mode DDR controller and MEW module supports ECC and debug watchpoint capabilities on MMDC transactions. All the ECC errors detected by MEW are reported to FCCU.

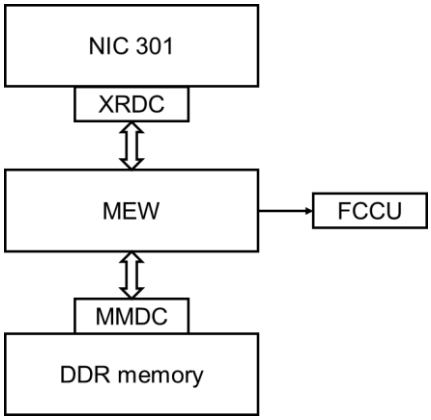

Figure 1. **MEW block diagram**

There are two instances of MEW and MMDC modules available on S32V23x. Each MEW internally uses four ECC modules in parallel. Each ECC module can generate eight ECC bits out of eight data bits and byte-aligned 32 address bits as shown in *Figure 2*.

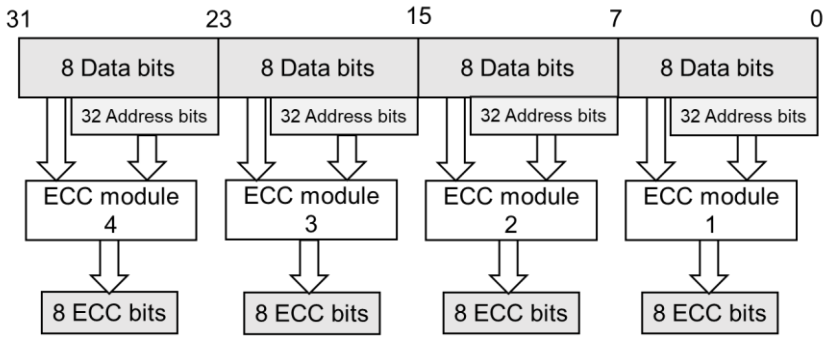

Figure 2. **Four ECC modules work in parallel**

Four ECC schemes are supported by MEW module of S32V23x as shown in *Table 2*:

1. SEC-DED-TED for 8 data bits and 8 ECC bits.
2. SED-DED for erroneous 23 address-bits.
3. SED for greater than 23 address bits.
4. Single error correction or Single error detection-Double error detection for combination of 8 data bits, 23 address bits and 8 ECC bits (SEC/SED-DED).

Table 2. **ECC Schemes on DDR**

Protected bits	SEC	SED	DED	TED
8 data bits	$X_{1,4}$		$X_{1,4}$	X_1
8 ECC bits	$X_{1,4}$		$X_{1,4}$	X_1
23 Address bits		$X_{2,4}$	X_2	
Address bits 23 to 31		$X_{3,4}$		

2.2. Impact on performance

When ECC feature is enabled, MEW module converts the incoming transactions to the new address if ECC region is accessed. For NON-ECC region, there is no conversion, but a default latency is added to match up the bus.

- The transfers which are smaller than x64 (like x8 / x16 / x32) are converted to double the current size: x8 => x16, x16 => x32, x32 => x64.
- For a transfer of 64-bit size and length < 9, the transaction will be doubled up to the maximum for a single burst. For burst transactions size x64 and length > 8, two bursts are generated.

The DRAM controller embeds ECC data within the normal data to the DRAM (inline ECC). While this reduces the available data rate and memory space for the payload, no additional DRAM device for ECC data is required.

In general, enabling ECC feature results in the following impacts:

- Reduce the available memory space, e.g. 1 MB protected normal data needs extra 1 MB space for its ECC data

- Reduce the DDR bus bandwidth at least in half
- Add some extra latency

2.3. Calculation of ECC on DDR

ECC algorithms that are also known as Hamming codes are mathematically determined. Hamming codes is based on the following concepts:

- In order to detect k-single bit error, minimum hamming distance (number of bit positions in which two code words differ) Dmin = k+1.
- In order to correct k-single bit error, minimum hamming distance Dmin = 2k +1.

In order to satisfy all the four ECC schemes, hamming distance-4 algorithm is used for development of the H-matrix which is the implementation of DDR ECC on S32V23x.

In this H-matrix, each data bit is exclusive Ored into four different ECC bits; Each ECC bit is implicitly exclusive Ored into a single ECC bits; 23-bit Address bits are exclusive Ored into 3, 6 or 8 ECC bits; Address bit 23 to 31 are exclusive Ored (XOR) with Address bit 0 to 8 to extend the address range to 32 bit.

The following figure shows the H-matrix [1]:

Table 3. **H-matrix**

		DDR bit 0	DDR bit 1	DDR bit 2	DDR bit 3	DDR bit 4	DDR bit 5	DDR bit 6	DDR bit 7
decimal code		15	51	85	106	150	172	216	225
ECC Bit	0	X	X	X					X
	1	X	X		X	X			
	2	X		X		X	X		
	3	X			X		X	X	
	4		X	X		X		X	
	5		X		X		X		X
	6			X	X			X	X
	7					X	X	X	X
binary code		00001111	00110011	01010101	01101010	10010110	10101100	11011000	11100001
		ECC bit 0	ECC bit 1	ECC bit 2	ECC bit 3	ECC bit 4	ECC bit 5	ECC bit 6	ECC bit 7
decimal code		1	2	4	8	16	32	64	128
ECC Bit	0	X							
	1		X						
	2			X					
	3				X				
	4					X			
	5						X		
	6							X	
	7								X
binary code		00000001	00000010	00000100	00001000	00010000	00100000	01000000	10000000

		Address bit 0	Address bit 1	Address bit 2	Address bit 3	Address bit 4	Address bit 5
decimal code		95	111	123	125	126	175
ECC Bit	0	X	X	X	X		X
	1	X	X	X		X	X
	2	X	X		X	X	X
	3	X	X	X	X	X	X
	4	X		X	X	X	
	5		X	X	X	X	X
	6	X	X	X	X	X	
	7						X
binary code		01011111	01101111	01111011	01111101	01111110	10101111
		Address bit 6	Address bit 7	Address bit 8	Address bit 9	Address bit 10	Address bit 11
decimal code		187	189	207	219	221	222
ECC Bit	0	X	X	X	X	X	
	1	X		X	X		X
	2		X	X		X	X
	3	X	X	X	X	X	X
	4	X	X		X	X	X
	5	X	X				
	6			X	X	X	X
	7	X	X	X	X	X	X
binary code		10111011	10111101	11001111	11011011	11011101	11011110

		Address bit 12	Address bit 13	Address bit 14	Address bit 15	Address bit 16	Address bit 17
decimal code		25	26	28	37	38	52
ECC Bit	0	X			X		
	1		X			X	
	2			X	X	X	X
	3	X	X	X			
	4	X	X	X			X
	5				X	X	X
	6						
	7						
binary code		00011001	00011010	00011100	00100101	00100110	00110100
		Address bit 18	Address bit 19	Address bit 20	Address bit 21	Address bit 22	
decimal code		82	131	133	145	255	
ECC Bit	0		X	X	X	X	
	1	X	X			X	
	2			X		X	
	3					X	
	4	X			X	X	
	5					X	
	6	X				X	
	7		X	X	X	X	
binary code		01010010	10000011	10000101	10010001	11111111	

Follow the steps below to calculate the ECC value:

First step, Address bit 23 to 31 are exclusive Ored (XOR) with Address bit 0 to 8 to extend the address range to 32-bit:

$$
\begin{aligned}
Addressbit_0 &= Addressbit_0 \oplus Addressbit_{23} \\
Addressbit_1 &= Addressbit_1 \oplus Addressbit_{24} \\
Addressbit_2 &= Addressbit_2 \oplus Addressbit_{25} \\
Addressbit_3 &= Addressbit_3 \oplus Addressbit_{26} \\
Addressbit_4 &= Addressbit_4 \oplus Addressbit_{27} \\
Addressbit_5 &= Addressbit_5 \oplus Addressbit_{28} \\
Addressbit_6 &= Addressbit_6 \oplus Addressbit_{29} \\
Addressbit_7 &= Addressbit_7 \oplus Addressbit_{30} \\
Addressbit_8 &= Addressbit_8 \oplus Addressbit_{31}
\end{aligned}
$$

Then, the ECC bits can be calculated according to the H-matrix, calculation formula is as follow, take ECC bit 0 as an example:

$$ECCbit_0 = Databit_0 \oplus Databit_1 \oplus Databit_2 \oplus Databit_7 \oplus Addressbit_0 \oplus Addressbit_1 \oplus Addressbit_2 \oplus Addressbit_3 \oplus Addressbit_5 \oplus Addressbit_6 \oplus Addressbit_7 \oplus Addressbit_8 \oplus Addressbit_9 \oplus Addressbit_{10} \oplus Addressbit_{12} \oplus Addressbit_{15} \oplus Addressbit_{19} \oplus Addressbit_{20} \oplus Addressbit_{21} \oplus Addressbit_{22}$$

To be more concise, the formula is simplified to:

$$ECCbit_0 = \text{^}(Databit_{0,1,2,7} \ \& Addressbit_{0,1,2,3,5,6,7,8,9,10,12,15,19,20,21,22})$$

Formulas for Remaining ECC bits:

$$ECCbit_1 = \text{^}(Databit_{0,1,3,4} \ \& Addressbit_{0,1,2,4,5,6,8,9,11,13,16,18,19,22})$$
$$ECCbit_2 = \text{^}(Databit_{0,2,4,5} \ \& Addressbit_{0,1,3,4,5,7,8,10,11,14,15,16,17,20,22})$$
$$ECCbit_3 = \text{^}(Databit_{0,3,5,6} \ \& Addressbit_{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,22})$$
$$ECCbit_4 = \text{^}(Databit_{1,2,4,6} \ \& Addressbit_{0,2,3,4,6,7,9,10,11,12,13,14,17,18,21,22})$$
$$ECCbit_5 = \text{^}(Databit_{1,3,5,7} \ \& Addressbit_{1,2,3,4,5,6,7,15,16,17,22})$$
$$ECCbit_6 = \text{^}(Databit_{2,3,6,7} \ \& Addressbit_{0,1,2,3,4,8,9,10,11,18,22})$$
$$ECCbit_7 = \text{^}(Databit_{4,5,6,7} \ \& Addressbit_{5,6,7,8,9,10,11,19,20,21,22})$$

To give an example, it is assumed the address is 0xC2008000 and data is 0x44.

- ECC bits is calculated as 0xA1 while data is written
- If the last bit of the data bits is accidentally reversed, the data changed from 0x44 to 0x45
- The ECC bits are calculated again while being read and the value is 0xAE.
- Then the syndrome can be calculated by following formula:
  $Syndrome = ECCbits_{original} \text{ ^ } ECCbits_{now}$

As in this example, the syndrome is 0xA1 ^ 0xAE = 0x0F, then by comparing syndrome to binary code in the H-matrix as shown in *Table 3* and it can be found that the reversed bit is DDR data bit0 as shown in the *Figure 3*.

		DDR bit 0
decimal code		15
ECC Bit	0	X
	1	X
	2	X
	3	X
	4	
	5	
	6	
	7	
binary code		00001111

Figure 3. **Binary code of DDR data bit0**

Because the H-matrix is based on Hamming distance-4 algorithm, double and triple bits error in ECC and data bits can also be detected but cannot be corrected.

For address bits, 23-bit address is included in the H-matrix and address bit 23 to 31 are XOR with address bit 0 to 8 to extend the address range to 32-bit. The ECC scheme of 32-bit address can only detect single bit error. Using address bit0 and bit23 to give an example:

- If address bit0 is accidentally flipped, the syndrome becomes 0xFE, but if address bit23 accidentally flipped will also obtained the same syndrome. This is the reason why address bits support SED but not SEC.
- If address bit0 and bit23 are accidentally reversed at the same time, the syndrome will stay as 0xA1, that's to say there is no error detected. This is the reason why DED can't be supported on address bits.

3. DDR ECC configuration

By default, ECC feature on DDR is disabled. To enable ECC, there are steps that the user should follow (Take DDR0 as an example, the configuration of DDR1 is similar). It is recommended to configure the ECC mechanism at boot stage before any application uses DDR.

Writing unlock PIN two times to unlock the MEW registers:

```
MEW_AXI_0->ECC_ULK_PTN = 0xAA55A5A5;
MEW_AXI_0->ECC_ULK_PTN = 0xAA55A5A5;
if( MEW_AXI_0->ECC_ULK_PTN == 0xFFFFFFFE )
{
    /* MEW_AXI_0 registers are unlocked */
}
```

After unlocking, the following registers can be written.

- MEW_AXI_ECC_GLBL_CTRL
- MEW_AXI_ECC_MX_EPA
- MEW_AXI_ECC_MN_EPA
- MEW_AXI_ECC_SHD_STAT_CTRL[SHD_RGN_SLT]
- MEW_AXI_ECC_DBG_CTRL[ADD_EN_ECC_DIS]

Setting maximum and minimum protected region address:

```
MEW_AXI_0->ECC_MX_EPA = ECC_MAX_ADDRESS;
MEW_AXI_0->ECC_MN_EPA = ECC_LOW_ADDRESS;
```

ECC protected region is from ECC_LOW_ADDRESS to ECC_MAX_ADDRESS - 1 address. Enough memory space should be reserved, because the actual physical memory space occupation is as twice as the size of protected region:

$$\begin{Bmatrix} ECC\_MAX\_ADDRESS > ECC\_LOW\_ADDRESS \\ (2 * ECC\_MAX\_ADDRESS - ECC\_LOW\_ADDRESS) < DDR\_MAX\_ADDRESS \end{Bmatrix}$$

Both maximum and minimum address should be multiple of 64KBytes.

Enable ECC and lock the registers:

Enable the module on both write and read data path by setting the corresponding bits in the control register ECC_GLBL_CTRL. Then lock MEW registers back after configuration is done by writing 0x55AAAA55 twice to register ECC_LK_PTN.

```
MEW_AXI_0->ECC_GLBL_CTRL = 0x00090009; // Enable the ECC path on write and read
```

```
MEW_AXI_0->ECC_LK_PTN = 0x55AAAA55;
MEW_AXI_0->ECC_LK_PTN = 0x55AAAA55;
if( MEW_AXI_0->ECC_LK_PTN == 0xFFFFFFFE )
{
    /* MEW_AXI_0 registers are locked */
}
```

4. Fault injection and reaction

4.1. Shadow region

If ECC is enabled for ECC protected region, any transaction from SOC to protected region will be converted by MEW module. In SOC address view, ECC protected region only includes data which are protected by ECC, but in DDR-RAM physical address view ECC data are embedded in the normal data and use double RAM space. For example, 32-bit data 0x11223344 writes to ECC protected region by SOC will be stored as 64-bit ECC-DATA combined at physical address view like 0x••33••44••11••22, symbol '•' stands for ECC bit, e.g. 0x7533A0448511DC22.

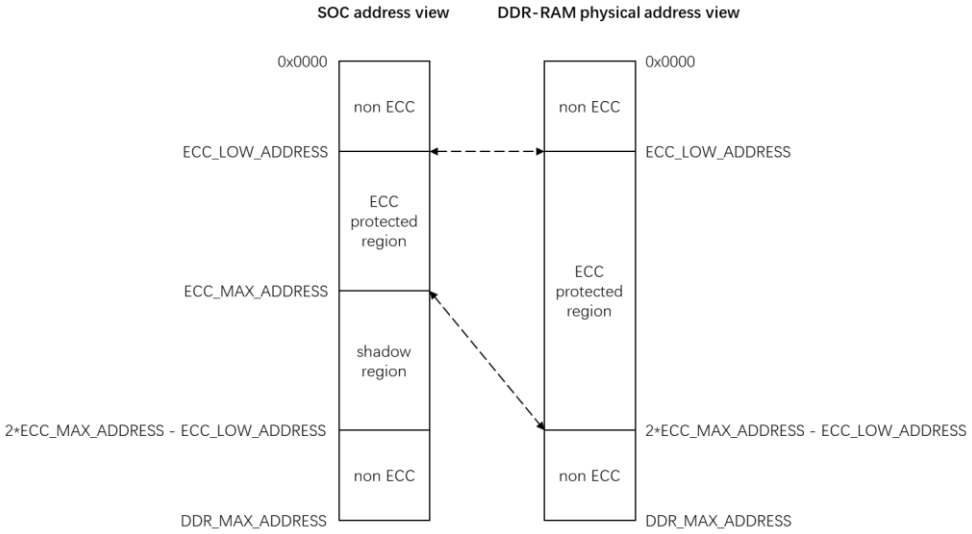

Figure 4. **Conversion between SOC and physical address views**

The address can be converted according to the following formula:

$$\text{address}_{\text{physical}} = 2 * address_{soc} - ECC\_LOW\_ADDRESS$$

In SOC address view as show in the *Figure 4*, shadow region is followed behind the protected region and is the same size with the protected region. The data bits can be modified without ECC bits recalculated and the ECC bits can be modified directly through shadow region. In normal application the shadow region should be protected by XRDC to avoid illegal access, as shown in the *Figure 1*. But in the fault injection application, shadow region can be used to inject fault. A single bit flipped through the shadow region can inject a single bit correctable ECC error and can be triggered by reading the corresponding data from the protected region. Two or more bits flipped through the shadow region will

inject an uncorrectable ECC error and can be triggered by reading the corresponding data from the protected region. Both single bit correctable and uncorrectable errors will be sent to the FCCU.

Because of size limitation, the shadow region can only be mapped to half of the protected region physical address, the register MEW_AXI_ECC_SHD_STAT_CTRL[SHD_RGN_SLT] is used to select first or second half that shadow region points to. Shadow region allows users to directly access the physical memory view that both data bits and ECC bits can be accessed.

By default, MEW_AXI_ECC_SHD_STAT_CTRL[SHD_RGN_SLT] = 1, shadow region points to second region of ECC protected region physical address as shown in the *Figure 5*. In this case, the start address of shadow region points to the data in protected region address (ECC_MAX_ADDRESS + ECC_LOW_ADDRESS)/2, the end address of shadow region points to the protected region address ECC_MAX_ADDRESS.

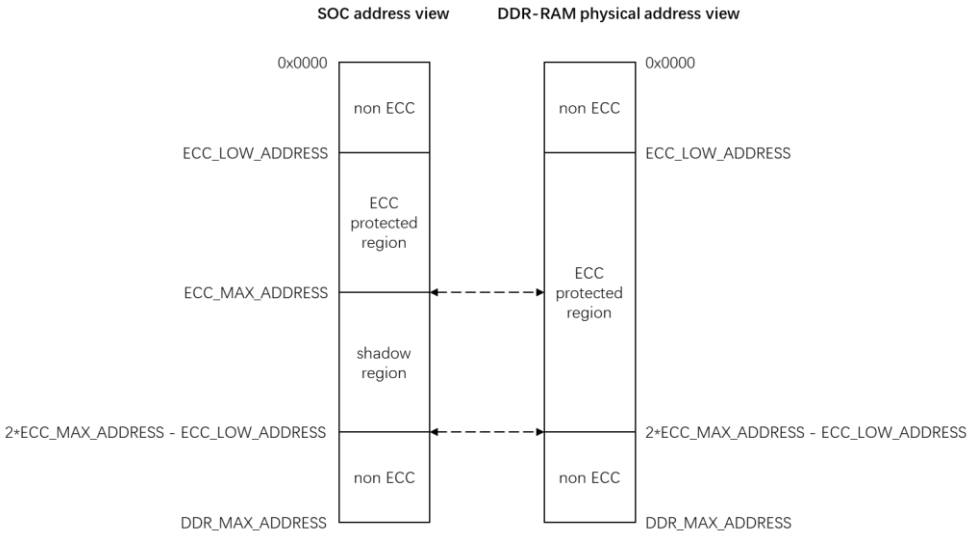

Figure 5. **Shadow region points to second region**

Shadow region can be selected to point to first region of ECC protected region physical address as shown in the *Figure 6* by setting MEW_AXI_ECC_SHD_STAT_CTRL[SHD_RGN_SLT] = 0. In this case, the start address of shadow region points to the data in protected region address ECC_LOW_ADDRESS, the end address of shadow region points to the protected region address (ECC_MAX_ADDRESS + ECC_LOW_ADDRESS)/2.

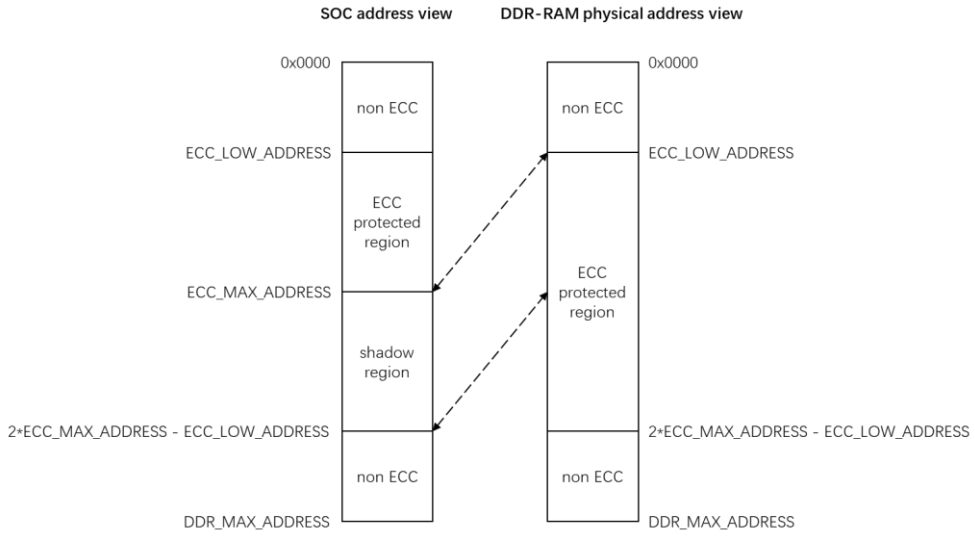

Figure 6. **Shadow region points to first region**

4.2. Steps to inject and trigger fault

Reading and writing operation on shadow region will not trigger ECC mechanism. Therefore, the data can be modified through shadow region without ECC bits recalculated and the ECC bits can be modified directly. Fault injection can be performed by this way.

Step 1. Configure and enable ECC

Configure and enable ECC by following the steps described in section 3.

Step 2. Initialize ECC bits

After ECC enabled, ECC bits are not automatically calculated and stored, the ECC bits need to be initialized by writing on the protected region.

Step 3. Inject fault

ECC mechanism on DDR supports single bit error correction and up to 3 bits error detection on data and ECC bits. So, fault can be injected by changing 1 to 3 bits of target data into the corresponding physical view address in the shadow region. Data bits and ECC bits in the shadow region can be changed in many ways, such as the user application on M4 and the u-boot command on A53, please refer to section 5 for more details.

Step 4. Trigger fault

After fault injection, a read operation at the target address in protected region will trigger an ECC fault. The corresponding error flag can be observed in the register MEW_AXI_ECC_ERR_IN_STCLR.

4.3. Fault reaction

When there is an ECC fault, there are some mechanisms to catch the fault information.

Inside the MEW, the following registers store information about the ECC fault:

- **MEW_AXI_ECC_ERR_IN_STCLR** register indicates the fault type and which ECC module has the fault.
- **MEW_AXI_ECC_EERAR** register gives 32-bit address which caused the first ECC fault.
- **MEW_AXI_ECC_EERDSRn** registers contain detailed information of the ECC fault, including data bits, ECC bits and syndrome.
- **MEW_AXI_ECC_CBL_UNCBL_BIT_EC** register records the count of correctable fault and uncorrectable fault. If the count exceeds the upper limit (15-bit unsigned int), the overflow flag bit will be set.

When interrupts are enabled in MEW_AXI_ECC_ERR_IE register, the corresponding ECC fault will be sent to FCCU module for user-defined handling.

There are two types of ECC faults: single bit correctable fault and uncorrectable fault. Reactions to the fault can be configured in the FCCU, including functional reset, alarm interrupt and active output pin. In this note, the alarm interrupt and reset are used as examples. The fault ID of single bit correctable error is 99 and the fault ID of uncorrectable error is 100. Fault ID is the fault source that caused FCCU to enter ALARM state. Refer to S32V234_NCF_List.xslx for Fault ID list (Reference Manual [1] attachment).

If alarm interrupt is configured as FCCU's reaction to the DDR ECC fault, an interrupt will be generated when ECC fault is detected by MEW. Then the fault should be handled in the FCCU interrupt handler to clear the fault source, fault flags in MEW and FCCU modules. Fault flags in MEW_AXI_ECC_ERR_IN_STCLR can be cleared by writing 1 to corresponding flag bit or setting MEW_AXI_ECC_SHD_STAT_CTRL[STAT_CLR] bit to clear all fault flags.

If reset is configured as FCCU's reaction to the DDR ECC fault, a reset will be generated when ECC fault is detected by MEW. A Long functional reset is recommended to react to the uncorrectable ECC error.

5. Configuration and verification examples

This section gives two examples, one is for M4 core and the other for A53.

5.1. Sample for M4

In this case, DDR1 memory region 0xC2000000 to 0xC2010000 is used as the protected region. The following function is used to configure MEW at initialization stage.

```
static void MEW_Init( void )
{
        /* Unlock MEW_AXI_ECC_ULK_PTK by writing unlock PIN 0xAA55A5A5 two times */
        REG_WRITE32( &(MEW_AXI_1->ECC_ULK_PTN), 0xAA55A5A5 );
        REG_WRITE32( &(MEW_AXI_1->ECC_ULK_PTN), 0xAA55A5A5 );
        /* High Address boundary for ECC protected memory region */
        REG_WRITE32( &(MEW_AXI_1->ECC_MN_EPA), 0xC2000000 );
        /* Low Address boundary for ECC protected memory region */
        REG_WRITE32( &(MEW_AXI_1->ECC_MX_EPA), 0xC2010000 );
```

```
        /* Enable the ECC path on write */
        REG_BIT_SET32( &(MEW_AXI_1->ECC_GLBL_CTRL),
        MEW_AXI_ECC_GLBL_CTRL_WR_EN_MASK );
        /* Enable the ECC path on read */
        REG_BIT_SET32( &(MEW_AXI_1->ECC_GLBL_CTRL),
        MEW_AXI_ECC_GLBL_CTRL_RD_EN_MASK );
        /* Lock MEW_AXI_ECC_ULK_PTK by writing unlock PIN 55AAAA55 two times */
        REG_WRITE32( &(MEW_AXI_1->ECC_LK_PTN), 0x55AAAA55 );
        REG_WRITE32( &(MEW_AXI_1->ECC_LK_PTN), 0x55AAAA55 );
}
```

After enabling ECC feature, the ECC protected region must be first initialized, otherwise ECC errors will come.

```
for ( uint32_t i = 0xC2000000; i < 0xC2010000; i += 4 )
{
        REG_WRITE32( (uint32_t * )(i), 0x00000000 );
}
```

Inject fault by modifying data and ECC bits through shadow region. Trigger a fault by reading the corresponding data from the protected region address.

```
/* Inject correctable error */
static void DDR_InjectCErr( void )
{
        /* Write data 0x11223344 in 0xC2008000 for generating a ECC code */
        REG_WRITE32( (uint32_t * )(0xC2008000), 0x11223344 );
        /* Read the ECC code of 0xC2008000 (ECC code located at 0XC2010000) */
        uint32_t test_DDR = REG_READ32( (uint32_t * )(0xC2010000) );
        /* Invert one bit of ECC code */
        test_DDR = (((~test_DDR) & 0x00000001) | (test_DDR & 0xFFFFFFFE));
        /* Write back inverted ECC code to 0xC2010000 */
        REG_WRITE32( (uint32_t * )(0xC2010000), test_DDR );
        /* Read data to trigger a ECC correctable error */
        test_DDR = REG_READ32( (uint32_t * )(0xC2008000) );
        (void)test_DDR;
}

/* Inject uncorrectable error */
static void DDR_InjectUErr( void )
{
        /* Write data 0x11223344 in 0xC2008000 for generating a ECC code */
        REG_WRITE32( (uint32_t * )(0xC2008000), 0x11223344 );
        /* Read the ECC code of 0xC2008000 (ECC code located at 0XC2010000) */
        uint32_t test_DDR = REG_READ32( (uint32_t * )(0xC2010000) );
        /* Invert two bits of ECC code */
        test_DDR = (((~test_DDR) & 0x00000003) | (test_DDR & 0xFFFFFFFC));
        /* Write back inverted ECC code to 0xC2010000 */
```

```
        REG_WRITE32( (uint32_t * )(0xC2010000), test_DDR );
        /* Read data to trigger a ECC uncorrectable error */
        test_DDR = REG_READ32( (uint32_t * )(0xC2008000) );
        (void)test_DDR;
}
```

5.2. Sample for A53 (u-boot)

In this case, DDR0 memory region 0x80000000 to 0xA0000000 is used as the protected region. The ECC feature on DDR can be configured in u-boot code after DRAM is initialized.

```
static void MEW_Init( void )
{
        /* unlock registers */
        writel(0xAA55A5A5, 0x40037010);
        writel(0xAA55A5A5, 0x40037010);
        /* set protected region address */
        writel(0x80000000, 0x40037008);
        writel(0xA0000000, 0x40037004);
        /* enable ECC */
        writel(0x00090009, 0x40037000);
        /* lock registers */
        writel(0x55AAAA55, 0x4003700C);
}
```

In this case, there is 512 MB memory which needs to be initialized. To optimize the initialization time, DMA module can be used to do this initialization.

```
static void ECC_protected_memory_Init( void )
{
        /* DMA CR */
        writel(0x00000100, 0x40002000);
        /* DMA EEI */
        writel(0x00000001, 0x40002014);
        /* DMA TCD */
        /* SOURCE ADDR */
        writel(0x00006780, 0x40003000);
        /* ATTR */
        writew(0x0505, 0x40003006);
        /* NBYTES */
        writel(0x20000000, 0x40003008);
        /* DEST ADDR */
        writel(0x80000000, 0x40003010);
        /* DEST OFFSET */
        writew(0x0020, 0x40003014);
        /* CITER ELINKNO */
        writew(0x0001, 0x40003016);
```

```
        /* CSR */
        writew(0x0000, 0x4000301C);
        /* BITER ELINKNO */
        writew(0x0001, 0x4000301E);
        /* Trigger */
        writeb(0x0000, 0x4000201D);
        while((readw(0x4000301C) & 0x0080)!=0x0080)
        {
            /* Wait until complete */
        }
}
```

Using DMA to initialize 512 MB of memory typically takes 4-5 seconds. For time-critical application scenarios there is a faster way to use ARM® NEON to initialize the DRAM, it takes around 276 us to initialize 512 MB memory:

```
void neon_memset_8uint(uint8_t *apDst, uint8_t aVal, int32_t aSize)
{
        /* Set up buffers and number of iterations */
        long lSIMDIterations = aSize / 16;
        uint8_t lSrcVals[16] = {aVal};
        memset(lSrcVals, aVal, 16);
        char* lpDstVals = (char*)apDst;
        char* lpSrcVals = (char*)lSrcVals;
        /* Run NEON */
        __asm volatile(
        "LD1 {V0.16B}, [%[lpSrcVals]]    \n\t"
        "1: \n\t"
        "ST1 {V0.16B}, [%[lpDstVals]], #16      \n\t"
        "subs %[lSIMDIterations],%[lSIMDIterations],#1 \n\t"
        "bne 1b \n\t"
        : [lpSrcVals] "+r"(lpSrcVals), [lpDstVals] "+r"(lpDstVals),
        [lSIMDIterations] "+r"(lSIMDIterations)
        :);
}

static void ECC_protected_memory_Init( void )
{
        neon_memset_8uint((uint8_t *)0x80000000, 0, 536870912);
}
```

An ECC fault can be injected and triggered by using the u-boot command line. In this way it can be verified if ECC feature on DDR works as expected.

```
=> mw.l 0x90000000 0x11223344 1          ← write 0x11223344 to ECC protected address 0x90000000
=> md.l 0xA0000000 2                     ← read the corresponding shadow region address of 0x90000000
a0000000: 3933ed44 c9119022
```

```
=> mw.l 0xA0000000 0x3933ed45 1        ← inject single bit error through shadow region
=> md.l 0x40037030 1                   ← the error flags show no errors
40037030: 00000000
=> md.l 0x90000000 1                   ← trigger ECC error by reading on injected address
90000000: 11223344
=> md.l 0x40037030 1                   ← the error flag shows there is a correctable error detected
40037030: 00010000
```

6. Summary

ECC feature on DDR can enhance data accuracy and stability, reduce single point failure and latent failure on DDR memory. The usage of ECC on DDR is closely related to application scenarios. The configuration and initialization of the ECC protected region should be done before DDRs are used. And the detected ECC faults should be properly handled. The sample codes in this note are for reference only. For more information, please refer to the reference manual [1] and the safety manual [2].

7. Reference

1. S32V234 Reference Manual, S32V234RM, Rev. 5, 11/2019
2. Safety Manual for S32V234, S32V234SM, Rev. 3, 10/2017

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN13109
Rev. 0
12/2020