Application Report

SPRACM1–September 2019

AM65x DDR ECC Initialization and Testing

Sitara Processors

ABSTRACT

The error correcting code (ECC) feature of double data rate (DDR) memories improves system reliability by protecting against random bit errors that occur with dynamic random access memory (DRAM) technology. This document presents the ECC initialization, usage, methods of priming, and error injection tests for the DDR controller on the Texas Instruments AM65x class of processors. Firmware developers and system designers should apply this information in the development of robust systems that require off-chip DDR memory.

Contents

1 Introduction 2
2 ECC Support 2
3 Requirements for ECC 2
4 New Features / Differences 2
5 ECC Function 3
6 How to Enable ECC 3
7 ECC Poison Feature 4
8 Methods for Priming ECC 7
9 Testing ECC 11
10 ECC Registers 15
11 References 15

List of Tables

1 DDRCTL_ECCCFG0 Register Field Descriptions 4
2 DDRCTL_ECCCFG1 Register Field Descriptions 4
3 DDRCTL_ADDRMAP2 Register Field Descriptions 5
4 DDRCTL_ECCPOISONADDR0 Register Field Descriptions 6
5 DDRCTL_ECCPOISONADDR1 Register Field Descriptions 6
6 ECC Registers 15

Trademarks

All trademarks are the property of their respective owners.

Copyright © 2019, Texas Instruments Incorporated

1 Introduction

This document discusses the operation and testability of the error correcting code (ECC) feature of the DDR memory controller on the Texas Instruments AM65x class of processors. The ECC discussed is applicable to DDR3(L) and DDR4 memories. This document does not cover LPDDR4 ECC, in-band ECC, Command/ Address parity (DDR4), or Cyclical Redundancy Checks (CRC, DDR4).

2 ECC Support

The AM65x DDR controller provides support for ECC to protect against bit errors that can occur in DRAM memories. The ECC performs Single-Error Correction and Double-Error Detection (SEC/DED) by storing parity bits in a separate memory through a dedicated byte lane reserved for ECC (traditional sideband ECC).

ECC Features

- SEC/DED for 32-bit interface (39 bits with ECC) - supported only for DDR3 and DDR4
  - Example: 2 16-bit DRAMs for 32-bit data + 1 16-bit DRAM for ECC (7 of 16 bits utilized)
  - Example: 4 8-bit DRAMs for 32-bit data + 1 8-bit DRAM for ECC (7 of 8 bits utilized)
- SEC/DED for 16-bit interface (22 bits with ECC) - supported for DDR3 and DDR4
- SEC/DED for 8-bit interface (13 bits with ECC) - supported only for DDR3 and DDR4
- Read-modify-write ECC for sub-word writes
- ECC address error logging
- Statistical counters for counting ECC errors
- Injecting ECC errors during normal operation for validation
- Automatic ECC scrub operation inside the DDR controller for any read command received which has led to a single-bit error.

NOTE: Errata i2009 DDRSS: DDR Controller ECC Scrubbing feature can cause DRAM data corruption - keep disabled at all times (DIS_SCRUB = 1). For more information, see the AM65x/DRA80xM Processors Errata.

For more information, see the *DDRSS Overview* chapter in the *AM65x/DRA80xM Processors Technical Reference Manual*.

3 Requirements for ECC

The basic hardware and software requirements to utilize ECC are shown below:

- Dedicated DRAM on ECC byte lane to store ECC bits (in addition to DRAMs for data)
- Layout routing rules apply to ECC byte lane similar to other byte lanes
- DDR PHY must enable and train ECC byte lane during PHY leveling/training sequence
- DDR controller must enable ECC during initialization.

For more information, see *AM65x/DRA80xM EMIF Tools*.

4 New Features / Differences

The controller must enable ECC during initialization. ECC cannot be deactivated unless the controller is first reset. Therefore, ECC cannot be tested by disabling ECC, writing data without ECC, then re-enabling ECC to detect that introduced error. Attempting to disable ECC after initialization has no effect.

Instead the ECC supports a poisoning feature at a specified address. At that address, the ECC data is poisoned with either a 1-bit or a 2-bit error. For more information, see Section 7. Most register bit fields are of Programming Mode Static, which means they can be written only when the controller is in reset. For more information, see the *DDR Controller Register Specifics* section in the *AM65x/DRA80xM Processors Technical Reference Manual*.

Copyright © 2019, Texas Instruments Incorporated

5 ECC Function

To use ECC, the ECC byte lane needs to be enabled during DDR initialization and trainng and the ecc_mode bitfield needs to be set to 0b100 - ECC enabled in the ECCCFG0 register.

If ECCCFG0.ecc_mode = 0b100, SEC/DED ECC is enabled. In this mode, the controller performs the following functions:

On writes, the ECC is calculated across each word of data (of width equal to the SDRAM width) and the resulting ECC code is written to a separate ECC byte lane. This additional ECC byte is always written to the uppermost byte of SDRAM. For DDR4 and DDR3, the ECC byte lane is byte lane 4 for x32 bus width and byte lane 2 for x16.

On reads, all byte lanes including the ECC byte are read from SDRAM. The ECC code is then “decoded”. A check is performed to verify that the ECC code is as expected, based on the data word read from the non-ECC bytes. If it is correct, the data is sent to the SoC core as normal.

The Read-Modify-Write (RMW) feature allows ECC to be used for sub quanta writes, or writes to an address that is not aligned with the DDR bus-width quanta. If the RMW module detects write data that does not form a complete quanta, a read for that burst (typically 32 bytes) is then issued by the controller. This return data burst is then merged with the sub-quanta write data and the modified burst is then written to the SDRAM.

When the controller detects a correctable ECC error (1-bit), it performs the following:

- Sends the corrected data to the SoC core as part of the read data
- Stores the ECC error information in the registers

When the controller detects an uncorrectable error, it does the following:

- Sends the data with error to the SoC core as part of the read data
- Stores the ECC error information in the registers
- Generates an error, which can trigger an interrupt to the core through the ECC aggregator. For more information, see the *ECC Aggregator Registers* section and the *DDRSS0 Hardware Requests* table in the *AM65x/DRA80xM Processors Technical Reference Manual*.

6 How to Enable ECC

During the DDR PHY initialization sequence, keep the ECC byte enabled during the training sequence. By default it will be enabled with register DX4GCRn (32-bit mode, byte 4) or DX2GCRn (16-bit mode/NARROW MODE, byte 2). Perform all PHY training steps with the ECC byte enabled.

During the DDR controller initialization, write 0x00000014 to ECC Configuration Register 0 (ECCCFG0). This enables ECC with default settings (ecc_mode = 0b100 - ECC enabled - SEC/DED over 1 beat) and disables scrubbing (for SR 1.0 errata, dis_scrub = 1). A beat represents the position of the data in the SDRAM burst.

These ECC register writes are handled by the EMIF Register Configuration Tool. In the Step1-System Details tab, set "Enable ECC" to "Yes". The resulting GEL file and uBoot .dtsi file outputs will enable ECC by setting DDRSS_DDRCTL_ECCCFG0 to 0x00000014 and enabling the ECC byte lane in the respective DATX8 n General Configuration Registers.

Copyright © 2019, Texas Instruments Incorporated

Table 1. DDRCTL_ECCCFG0 Register Field Descriptions

Bit	Field	Description
31-5	RESERVED	
4	DIS_SCRUB	Disable ECC scrubs. Valid only when .ecc_mode = 3'b100 or 3'b101. Programming Mode: Static
3	TEST_MODE	If this bit is set to 1, no ECC is performed, and the ECC byte is accessed directly from co_wu_rxdata_ecc and ra_co_resp_ecc_data. This test mode is only supported in full bus width mode. In other words, if .data_bus_width is non-zero, this test_mode field must be set to 0. If test_mode is set to 1, the ecc_mode field is ignored. Programming Mode: Static
2-0	ECC_MODE	ECC mode indicator 000 - ECC disabled 100 - ECC enabled - SEC/DED over 1 beat 101 - ECC enabled - Advanced ECC (not supported) - all other settings are reserved for future use Programming Mode: Static

Software must write into the ECC protected DDR memory before reading from it to prime each ECC byte with valid codes.

Reading from unprimed addresses will generate false ECC errors because the ECC DRAM has not been primed (or written to) with valid ECC codes. You must write to DDR before reading from it. During a read, if a DDR prefetch accesses an unprimed memory region, ECC errors will occur if the prefetched data is not masked and and it propagates into the ECC engine through the memory controller. For methods on priming, see Section 8.

7 ECC Poison Feature

For testing purposes, the ECC Poison feature can be used to test ECC software by introducing errors to the code read from the ECC byte, falsely generating 1-bit or 2-bit errors.

The poisoning must be configured and enabled at initialization time, and cannot be enabled after initialization. During initialization, write to ECCCFG1 to enable poisoning, and select either 1-bit or 2-bit poisoning.

Table 2. DDRCTL_ECCCFG1 Register Field Descriptions

Bit	Field	Description
31-2	RESERVED	
1	DATA_POISON_BIT	Selects whether to poison 1 or 2 bits if 0 -> 2-bit (uncorrectable) data poisoning, if 1 -> 1-bit (correctable) data poisoning, if .data_poison_en=1. Programming Mode: Quasi-dynamic Group 3
0	DATA_POISON_EN	Enable ECC data poisoning - introduces ECC errors on writes to address specified by the /1 registers Programming Mode: Quasi-dynamic Group 3

- When ECCCFG1 = 0x00000003, poisoning is enabled with 1-bit error
- When ECCCFG1 = 0x00000001, poisoning is enabled with 2-bit error

Poisoning uses the physical address which is the DDR memory's RANK, BANK, COL, and ROW addressees.

The logical to physical address mapping is configurable through the DDRCTL_ADDRMAPn registers. These registers map the physical address to the Host Interface (HIF) address. The logical address can then be calculated by left shifting the HIF address by 2 binary places and adding it to the DDR base address.

Copyright © 2019, Texas Instruments Incorporated

For example, lets convert the physical address with RANK = 0, Bank = 0, ROW = 0, COL = 0x04 (0b00000100). COL1 and COL0 are fixed and cannot be moved from the least significant two bits of the HIF address. The DDRCTL_ADDRMAP2 determines which binary place in the HIF address the COL address bits 5 through 2 map to. The second bit is set high in the COL address (counting from zero). The default value of zero in the ADDRMAP_COL_B2 bitfield of the DDRCTL_ADDRMAP2 register maps COL[2] to HIF[2]. Writing a larger value into this bitfield moves the mapping of COL[2] to a more significant bit position. A value of 1 maps COL[2] to HIF[3], and so forth. With the default value of zero in ADDRMAP_COL_B2 and the only physical address bit set being COL[2], the HIF address is 0x04 (0b00000100). Converting from HIF to logical address by left shifting by two binary spaces and adding the DDR base address (0x80000000) results in a logical address of 0x80000010.This calculation is valid when configured for a 32-bit data bus width. If DDR is configured for a 16-bit data bus width, the logical address is equal to the HIF address left shifted by one binary place. In summary, a physical address of RANK 0, BANK 0, ROW 0, COL 0x04 maps to 0x80000010 with the default value in bit field ADDRMAP_COL_B2. The EMIF Register Configuration Tool supports configuration of the address map and generates the corresponding register values to be written during initialization.

As another example, to convert the logical address 0x80000020 to its physical address perform the above procedure in reverse. First subtract the base address of DDR (0x80000000) and right shift by two binary places resulting in HIF address 0x00000008 (0b00100000 right shifted by 2 places becomes 0b00001000 = 0x08). to identify which physical address component is mapped to HIF address bit position 3 (counting from zero, 0x08 has only binary bit 3 set high), see the *EMIF Register Configuration Tool Address Mapping* tab. Alternately, read the DDRCTL_ADDRMAPn registers to identify which physical address component is mapped to HIF address bit position 3. These registers map the physical address components to a limited range of HIF address bit positions. In its reset state, register DDRCTL_ADDRMAP2 bits [11:8] ADDRMAP_COL_B3 map COL[3] to HIF address bit 3. Since all other bits of the logical address are zero, all other bits of the physical address components are set low when logical address 0x08000020 is accessed: RANK 0, BANK 0, ROW 0, COL 0x08.

The length of each physical address component depends the DRAM used and is configured. Flexibility in the HIF to physical address mapping can be leveraged to maximize bandwidth (bank interleaving).

Table 3. DDRCTL_ADDRMAP2 Register Field Descriptions

Bit	Field	Description
31-28	RESERVED	
27-24	ADDRMAP_COL_B5	- Full bus width mode: Selects the HIF address bit used as column address bit 5. Valid Range: 0 to 7, and 15 Internal Base: 5 The selected HIF address bit is determined by adding the internal base to the value of this field. If unused, set to 15 and then this column address bit is set to 0. Programming Mode: Static
23-20	RESERVED	
19-16	ADDRMAP_COL_B4	- Full bus width mode: Selects the HIF address bit used as column address bit 4. Valid Range: 0 to 7, and 15 Internal Base: 4 The selected HIF address bit is determined by adding the internal base to the value of this field. If unused, set to 15 and then this column address bit is set to 0. Programming Mode: Static
15-12	RESERVED	
11-8	ADDRMAP_COL_B3	- Full bus width mode: Selects the HIF address bit used as column address bit 3. Valid Range: 0 to 7, and 15 Internal Base: 3 The selected HIF address bit is determined by adding the internal base to the value of this field. If unused, set to 15 and then this column address bit is set to 0. Programming Mode: Static
7-4	RESERVED	
3-0	ADDRMAP_COL_B2	- Full bus width mode: Selects the HIF address bit used as column address bit 2. Valid Range: 0 to 7, and 15 Internal Base: 2 The selected HIF address bit is determined by adding the internal base to the value of this field. If unused, set to 15 and then this column address bit is set to 0. Programming Mode: Static

Copyright © 2019, Texas Instruments Incorporated

Injection of artificial ECC errors at a specific address, for testing purposes, is also highly configurable through ECC poison registers. The poisoned address is defined by programming the DDRCTL_ECCPOISONADDR0 and/or DDRCTL_ECCPOISONADDR1 registers with the physical address parameters.

Table 4. DDRCTL_ECCPOISONADDR0 Register Field Descriptions

Bit	Field	Description
31-25	RESERVED	
24	ECC_POISON_RANK	Rank address for ECC poisoning Programming Mode: Static
23-12	RESERVED	
11-0	ECC_POISON_COL	Column address for ECC poisoning. Note that this column address must be burst aligned: - In full bus width mode, ecc_poison_col [2:0] must be set to 0 - In half bus width mode, ecc_poison_col [3:0] must be set to 0 - In quarter bus width mode, ecc_poison_col [4:0] must be set to 0 Programming Mode: Static

Table 5. DDRCTL_ECCPOISONADDR1 Register Field Descriptions

Bit	Field	Description
31-30	RESERVED	
29-28	ECC_POISON_BG	Bank Group address for ECC poisoning Programming Mode: Static
27	RESERVED	
26-24	ECC_POISON_BANK	Bank address for ECC poisoning Programming Mode: Static
23-18	RESERVED	
17-0	ECC_POISON_ROW	Row address for ECC poisoning. This is 18-bits wide in configurations with DDR4 support and 16-bits in all other configurations. Programming Mode: Static

Since the COL address bits reside in the LSBs of the HIF address (with the default values in the DDRCTL_ADDRMAPn registers), it is easy to inject errors into the COL addresses 0x40 and 0x50, which map to logical addresses 0x80000100 and 0x80000140.

- ECCPOISONADDR0 = 0x0000040 injects errors at logical address 0x80000100
- ECCPOISONADDR0 = 0x0000050 injects errors at logical address 0x80000140

Example: col address 0x50

- Physical address all zeros except COL address = 0x50
- HIF Address = 0x50 (0b01010000) with default DDRCTL_ADDRMAPn regsisters (COL address bits are least significant).
- Logical address = 0x80000140 (0b0101000000) HIF shifted left by 2 (32-bit data width) added to the DDR base address 0x80000000 (32-bit DDR addressing enabled).

Copyright © 2019, Texas Instruments Incorporated

The below snippet from a DDR config GEL (modified to support ECC and poisoning) can be used to enable ECC poisoning during initialization:

```
... (GEL file before controller configuration)
//---------------------------------------------------------------------------------
 //Perform DDR controller configuration
//---------------------------------------------------------------------------------
 Write_MMR(DDRSS_DDRCTL_MSTR,0x41040010); Write_MMR(DDRSS_DDRCTL_RFSHCTL0,0x00210070);
// Write_MMR(DDRSS_DDRCTL_ECCCFG0,0x00000000); //replace with if/else ECC_ENABLE
if(!ECC_ENABLE) {
    Write_MMR(DDRSS_DDRCTL_ECCCFG0,0x00000000);
} else {
    // Write_MMR(DDRSS_DDRCTL_ECCCFG0,0x00000014 ); //enable ECC
    Write_MMR(DDRSS_DDRCTL_ECCCFG0,0x044FFFD4 ); //enable ECC with defaults
if(ECC_POISON)     {
        Write_MMR(DDRSS_DDRCTL_ECCCFG1,0x00000001 );
        //bit 1 data_poison_bit
        //if 0 -> 2-bit (uncorrectable),
        //if 1 -> 1-bit (correctable)
        //MM bit 0 data_poison_en - Enables ECC data poisoning
        Write_MMR(DDRSS_DDRCTL_ECCPOISONADDR0,0x0000040 );
        //col[4] --> HIF addr[4] --> logical addr[6]
        //ECCPOISONADDR0 = 0x0000040 poisons logical address 0x80000100
        }
    }
Write_MMR(DDRSS_DDRCTL_CRCPARCTL0,0x00008000);
... (rest of GEL file)
```

8 Methods for Priming ECC

To utilize ECC, the memory space must first be primed, or written to before being read from. Priming ensures valid ECC codes are written to the ECC byte corresponding to the data written to the data bytes. If DDR memory is read from before being written to, then ECC errors will occur. There are multiple methods of priming the ECC:

- DMA - the DMA does not use the memory management unit (MMU), simplifying the priming process.
- CPU writes (typically with a loop) - recommended with the MMU disabled.
- uBoot (for operating systems) - runs from R5 and A53. Recommend priming immediately after initialization since uBoot uses DDR. The R5 does not support the MMU.
- Code composer memory window - use the "Fill Memory" feature with base address and length. The memory window must not read from any unprimed DDR addresses before priming or ECC errors will occur. Point the memory window to non-DDR space before initializing DDR and using the Fill Memory feature to prime the DDR space.

For its efficiency, it is recommended to use the DMA to prime as soon as DDR has been initialized. Prime all DDR regions that may be accessed (if not the entire DDR space). Each priming method is further detailed below.

Copyright © 2019, Texas Instruments Incorporated

Priming With the DMA

Using the DMA is the easiest and fastest way to prime the ECC by writing to the DDR space before reading from it. For example, a buffer of 0x2000 bytes can be repeatedly copied by the DMA to DDR until the entire 2GB of DDR is filled. After executing this operation, the whole 2GB DDR space is primed and can be written to and read from with ECC protection. For more information, see the below code snippet from the udma_memcpy_test example in PROCESSOR-SDK-AM65X (http://www.ti.com/tool/PROCESSOR-SDK-AM65X). Path: pdk_am65xx_1_0_3\packages\ti\drv\udma\examples\udma_memcpy_test\udma_memcpy_test.c.

```
static int32_t App_udmaMemcpy(Udma_ChHandle chHandle,
                              void *destBuf,
                              void *srcBuf,
                              uint32_t length)
{
    int32_t     retVal = UDMA_SOK;
    uint32_t   *pTrResp, trRespStatus;
    uint64_t    pDesc = 0;
    uint8_t    *tprdMem = &gUdmaTprdMem[0U];

    /* Update TR packet descriptor */
    App_udmaTrpdInit(chHandle, tprdMem, destBuf, srcBuf, length);

    /* Submit TRPD to channel */
    retVal = Udma_ringQueueRaw(
                 Udma_chGetFqRingHandle(chHandle), (uint64_t) tprdMem);
    if(UDMA_SOK != retVal)
    {
        App_print("[Error] Channel queue failed!!\n");
    }
...
    return;
}
```

Priming With CPU Writes

A simple for loop can be used to prime the ECC. When using the A53 core to write to DDR, it is recommended to disable the MMU. Priming with the MMU enabled can lead to false ECC errors due to boundary alignment requirements.

MMU is not used during DMA accesses, and it is not used by the R5 core (during boot). The MMU can be disabled by modifying the entry portion of the project.

To disable the MMU, comment out the below lines in the entry file (k3m4_aarch64_entry_AVV.S). The lines being commented re-enable MMU after being disabled to load new MMU table address:

```
//File: k3m4_aarch64_entry_AVV.S
...
//comment these lines to keep MMU disabled
    // Re-enable MMU
    // mrs x0, SCTLR_EL3
    // orr x0, x0, #1
    // msr SCTLR_EL3, x0
    // dsb sy
    // isb
```

A for loop like the one below can be used to prime the DDR memory ECC. ECC errors will not be falsely generated during the priming process with the MMU disabled.

```
uint32_t * pDstBuff = (uint32_t *)0x80000000;
for(int i = 0; i<DATA_BYTES/4; i++)
{
    pDstBuff[i] = 0xABCD1234; // prime the DDR memory ECC with 32-bit data type
}
```


Copyright © 2019, Texas Instruments Incorporated

Priming With uBoot

The DDR initialization with uBoot executes from the R5 core, which does not support MMU. A simple for loop or DMA copy can be used immediately after initializing the DDR to prime the ECC. uBoot and Linux must not be permitted to read from unprimed DDR or else ECC errors will be generated. Inside the uBoot file u-boot/drivers/ram/k3-am654-ddrss, add a similar am654_prime_ecc() function as shown below.

```
static int am654_ddrss_probe(struct udevice *dev)
{
    struct am654_ddrss_desc *ddrss = dev_get_priv(dev);
    int ret;
    debug("%s(dev=%p)\n", __func__, dev);
    ret = am654_ddrss_ofdata_to_priv(dev);
    if (ret)
        return ret;
    ddrss->dev = dev;
    ret = am654_ddrss_power_on(ddrss);
    if (ret)
        return ret;
    ret = am654_ddrss_init(ddrss);
    ret = am654_prime_ecc();
    return ret;
}
static int am654_prime_ecc(void)
{
  u32 *ddrPtr;
  u32 index;
  ddrPtr = (u32 *) 0x80000000;
  for(index=0;index<0x80000000;index++)
    ddrPtr[index] = 0x00000000;  //Prime 1GB DDR ECC by writing to DDR space
return 0;
}
```


Copyright © 2019, Texas Instruments Incorporated

Priming With Code Composer Memory Window or GEL File

Code Composer Studio (CCS) can be used to prime a small amount of DDR memory. The time required to write large chunks of memory is much greater than using the DMA or CPU.

The CCS Memory Browser has a Fill Memory feature that can write to a range of memory with a specific value. Make sure not to read from the DDR region until it has been primed to avoid generating ECC errors. Point the Memory Browser to some other address like 0x0 before initializing DDR and using the Mem Fill feature to write to DDR memory.

A GEL file can be written to write to the DDR space before reading from it. GEL execution is also slow and impractical to prime the entire DDR space in reasonable time.

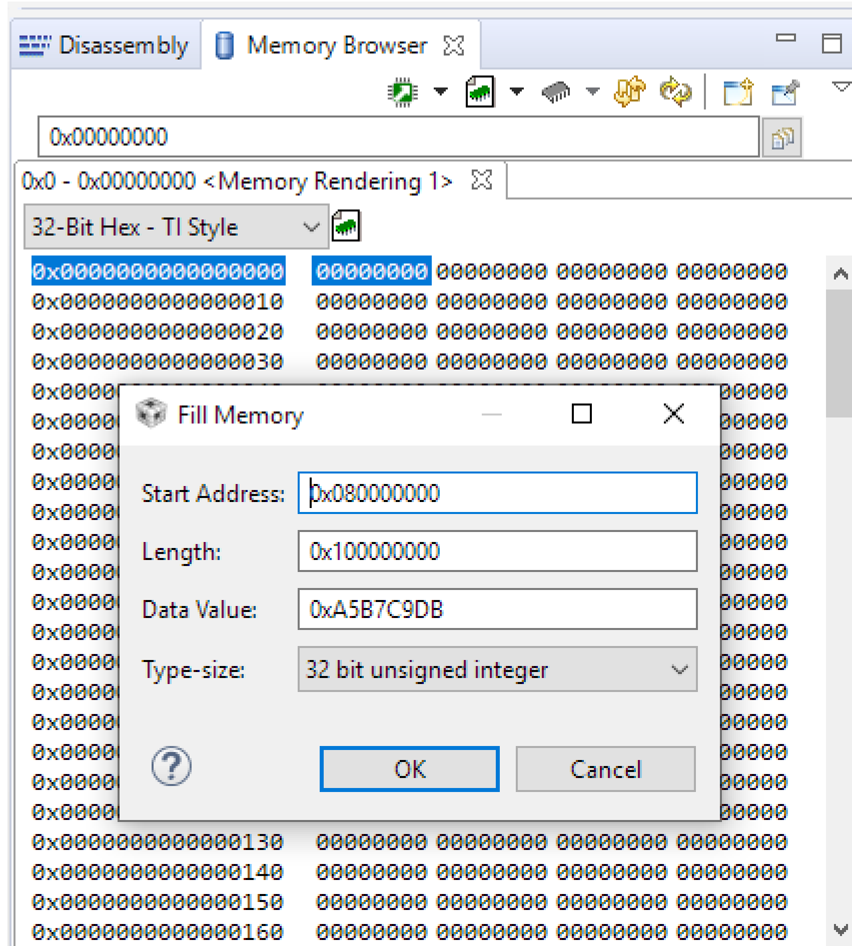

Figure 1. CCS Memory Window Fill Memory

Copyright © 2019, Texas Instruments Incorporated

9 Testing ECC

Once the DDR ECC is primed, it can be tested in a few ways:

- With ECC enabled and poisoning disabled, ensure no ECC errors occur during priming. After priming, write and read from the DDR space and ensure no ECC errors occur and no error counters increment. Try reading from unprimed memory locations to observe expected ECC errors.
- With ECC enabled, enable 1-bit poisoning to ensure that these introduced errors are detected and automatically corrected. Verify that the 1-bit (correctable) error counters increment and the appropriate error address registers correspond to the address that was poisoned.
- With ECC enabled, enable 2-bit poisoning to ensure that these introduced errors are detected (2-bit errors cannot be corrected). Verify that the 2-bit (uncorrectable) error counters increment and the appropriate error address registers correspond to the address that was poisoned.

NOTE: The test may hang upon detecting 2-bit errors, especially if interrupts are not configured to handle these errors.

- DDR stress tests can be utilized to validate the configuration of the DDR memory (including ECC byte lane), to prove that the DDR timings have margin, and that the layout is not impacted by crosstalk, signal integrity issues, and so forth.

GEL file functions can be used to dump the ECC registers and check for error status, error counts, error addresses, and so forth. For more information, see the GEL files that are installed with CCS when AM65x support is chosen during installation:
<CCS_PATH>\ccs_base\emulation\boards\am65x\gel\M4_DDR39SS\M4_DDR_Init.gel.

- ECC_RegDump_Decode() - a GEL hotmenu function to dump and decode ECC registers
- Data_WrRd_test() - a GEL hotmenu function to write, read, and verify DDR memory

Enable ECC Without Poisoning First

- Verify ECC byte lane trains without errors (same routing rules apply to ECC byte lane as the data byte lanes)
- Verify no errors occurred before priming step
- Prime DDR space (subset or all) and verify no ECC errors occurred during priming
- Write and read from DDR (using the Memory window, WrRd test provided in DDR GEL file, or some custom test)
- DDR memory in nominal conditions should not have any ECC errors

Reinitialize DDR With Poisoning Enabled

- It is recommended to power cycle the processor and DDR memory first
- Calculate logical address that will be corrupted by poison registers (physical address RANK, BANK, ROW, COL to logical address without MMU)
- Prime with one of the above methods (Recommend using the DMA or CPU with MMU disabled)
- Utilize write/read test to prove ECC detects and corrects 1-bit errors, and detects 2-bit errors (uncorrectable)

ECC registers expected to show errors: ERR CNT shows 1-bit and 2-bit error counts. "Corr" registers show correctable errors - address, bit, and so forth. "Uncorr" registers show uncorrectable 2-bit errors - address, bit, and so forth.

Solving for logical address when ECCPOISONADDR0 = 0x40

0b01000000 HIF << 2 = 0b0100000000 = 0x100, Logical address (0x80000100)

The poisoned ECCPOISONADDR0 address should match the address in DDRCTL_ECCCADDR0/1 (correctable) or DDRCTL_ECCUADDR0/1 (uncorrectable), depending on 1-bit or 2-bit poisoning.

Another example: ECCPOISONADDR0 = 0x50

0b01010000 HIF << 2 = 0b0101000000 = 0x140 Logical address (0x80000140)

Copyright © 2019, Texas Instruments Incorporated

Expected register dump after priming and ECCPOISONADDR0 = 0x00000050 (no errors are expected):

```
CortexA53_0_0: GEL Output: DDR ECC RegDump & Decode...
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCFG0 = 0x00000014
CortexA53_0_0: GEL Output: - uncorrected_err_threshold = 0
CortexA53_0_0: GEL Output: - eccap_en = 0
CortexA53_0_0: GEL Output: - inline_ecc_en = 0
CortexA53_0_0: GEL Output: - dis_scrub = 1
CortexA53_0_0: GEL Output: - test_mode = 0
CortexA53_0_0: GEL Output: - ecc_mode = 4 (4 = ECC enabled - SEC/DED over 1 beat)
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCFG1 = 0x00000003
CortexA53_0_0: GEL Output: - poison_chip_en = 0
CortexA53_0_0: GEL Output: - data_poison_bit = 1 (0 -> 2-bit (uncorrectable), 1 -> 1-
bit (correctable)
CortexA53_0_0: GEL Output: - data_poison_en = 1
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCSTAT = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorrected_err = 0
CortexA53_0_0: GEL Output: - ecc_corrected_err = 0
CortexA53_0_0: GEL Output: - ecc_corrected_bit_num = 0
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCLR = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCERRCNT = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_err_cnt = 0
CortexA53_0_0: GEL Output: - ecc_corr_err_cnt = 0
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCADDR0 = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_rank = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_row = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCADDR1 = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_cid = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_bg = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_bank = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_col = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCSYN0 (Corrected) = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCSYN2 (Corrected) = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCBITMASK0 = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCBITMASK2 = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCUADDR0 = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_rank = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_row = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCUADDR1 = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_cid = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_bg = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_bank = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_col = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCUSYN0 (Uncorrected) = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCUSYN2 (Uncorrected)= 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCPOISONADDR0 = 0x00000050
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCPOISONADDR1 = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ADVECCINDEX = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCPOISONPAT0 = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCPOISONPAT2 = 0x00000000
```


Copyright © 2019, Texas Instruments Incorporated

Expected register dump after reading once from the poisoned address (1-bit error) and ECCPOISONADDR0 = 0x00000040 :

- Prime 0x200 words
- Read only poisoned addresss 0x80000100 with ccs expressions window
- Observe a single 1-bit error detected, at expected ECC address (0x40)
- ecc_corr_err_cnt = 1

```
CortexA53_0_0: GEL Output:
CortexA53_0_0: GEL Output: DDR ECC RegDump & Decode...
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCFG0 = 0x00000014
CortexA53_0_0: GEL Output: - uncorrected_err_threshold = 0
CortexA53_0_0: GEL Output: - eccap_en = 0
CortexA53_0_0: GEL Output: - inline_ecc_en = 0
CortexA53_0_0: GEL Output: - dis_scrub = 1
CortexA53_0_0: GEL Output: - test_mode = 0
CortexA53_0_0: GEL Output: - ecc_mode = 4 (4 = ECC enabled - SEC/DED over 1 beat)
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCFG1 = 0x00000003
CortexA53_0_0: GEL Output: - poison_chip_en = 0
CortexA53_0_0: GEL Output: - data_poison_bit = 1 (0 -> 2-bit (uncorrectable), 1 -> 1-
bit (correctable)
CortexA53_0_0: GEL Output: - data_poison_en = 1
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCSTAT = 0x00000100
CortexA53_0_0: GEL Output: - ecc_uncorrected_err = 0
CortexA53_0_0: GEL Output: - ecc_corrected_err = 1
CortexA53_0_0: GEL Output: - ecc_corrected_bit_num = 0
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCLR = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCERRCNT = 0x00000001
CortexA53_0_0: GEL Output: - ecc_uncorr_err_cnt = 0
CortexA53_0_0: GEL Output: - ecc_corr_err_cnt = 1
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCADDR0 = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_rank = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_row = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCADDR1 = 0x00000040
CortexA53_0_0: GEL Output: - ecc_corr_cid = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_bg = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_bank = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_col = 0x00000040
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCSYN0 (Corrected) = 0xA5B7C9DB
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCSYN2 (Corrected) = 0x0000002B
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCBITMASK0 = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCBITMASK2 = 0x00000001
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCUADDR0 = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_rank = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_row = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCUADDR1 = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_cid = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_bg = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_bank = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_col = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCUSYN0 (Uncorrected) = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCUSYN2 (Uncorrected)= 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCPOISONADDR0 = 0x00000040
CortexA53_0_0: GEL Output: - ecc_poison_rank = 0x00000000
CortexA53_0_0: GEL Output: - ecc_poison_cid = 0x00000000
CortexA53_0_0: GEL Output: - ecc_poison_col = 0x00000040
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCPOISONADDR1 = 0x00000000
CortexA53_0_0: GEL Output: - ecc_poison_bg = 0x00000000
CortexA53_0_0: GEL Output: - ecc_poison_bank = 0x00000000
CortexA53_0_0: GEL Output: - ecc_poison_row = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ADVECCINDEX = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCPOISONPAT0 = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCPOISONPAT2 = 0x00000000
```


Copyright © 2019, Texas Instruments Incorporated

Expected register dump after reading once from the poisoned address (2-bit error) and ECCPOISONADDR0 = 0x00000050 :

- Prime 0x200 words
- Read only poisoned address 0x80000140 with expressions window
- Observe a single 2-bit error detected, at expected ECC address (0x50)
- ecc_uncorr_err_cnt = 1

```
CortexA53_0_0: GEL Output: DDR ECC RegDump & Decode...
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCFG0 = 0x00000014
CortexA53_0_0: GEL Output: - uncorrected_err_threshold = 0
CortexA53_0_0: GEL Output: - eccap_en = 0
CortexA53_0_0: GEL Output: - inline_ecc_en = 0
CortexA53_0_0: GEL Output: - dis_scrub = 1
CortexA53_0_0: GEL Output: - test_mode = 0
CortexA53_0_0: GEL Output: - ecc_mode = 4 (4 = ECC enabled - SEC/DED over 1 beat)
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCFG1 = 0x00000001
CortexA53_0_0: GEL Output: - poison_chip_en = 0
CortexA53_0_0: GEL Output: - data_poison_bit = 0 (0 -> 2-bit (uncorrectable), 1 -> 1-
bit (correctable)
CortexA53_0_0: GEL Output: - data_poison_en = 1
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCSTAT = 0x00010000
CortexA53_0_0: GEL Output: - ecc_uncorrected_err = 1
CortexA53_0_0: GEL Output: - ecc_corrected_err = 0
CortexA53_0_0: GEL Output: - ecc_corrected_bit_num = 0
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCLR = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCERRCNT = 0x00010000
CortexA53_0_0: GEL Output: - ecc_uncorr_err_cnt = 1
CortexA53_0_0: GEL Output: - ecc_corr_err_cnt = 0
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCADDR0 = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_rank = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_row = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCADDR1 = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_cid = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_bg = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_bank = 0x00000000
CortexA53_0_0: GEL Output: - ecc_corr_col = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCSYN0 (Corrected) = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCCSYN2 (Corrected) = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCBITMASK0 = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCBITMASK2 = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCUADDR0 = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_rank = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_row = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCUADDR1 = 0x00000050
CortexA53_0_0: GEL Output: - ecc_uncorr_cid = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_bg = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_bank = 0x00000000
CortexA53_0_0: GEL Output: - ecc_uncorr_col = 0x00000050
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCUSYN0 (Uncorrected) = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCUSYN2 (Uncorrected)= 0x0000007C
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCPOISONADDR0 = 0x00000050
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCPOISONADDR1 = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ADVECCINDEX = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCPOISONPAT0 = 0x00000000
CortexA53_0_0: GEL Output: DDRSS_DDRCTL_ECCPOISONPAT2 = 0x00000000
```


Copyright © 2019, Texas Instruments Incorporated

10 ECC Registers

A wide range of information about the detected errors can be obtained by reading the ECC error reporting registers. There are also two interrupts, ecc_corrected_err_intr and ecc_uncorrected_err_intr, which are asserted when corrected or uncorrected errors are detected. The register fields ECCSTAT.ecc_corrected_bit_num, ECCCADDR0.*, ECCCADDR1.*, ECCUADDR0.* and ECCUADDR1.* are not be cleared by the clear operation (see ECCCLR). The ECC logging registers capture only the first ECC correctable error and first ECC uncorrectable error detected. They store the log information related to those errors (even if additional errors arrive), until the associated interrupt is cleared by writing to the respective bits in the ECCCLR register.

Table 6. ECC Registers

Register Name	Address	Description
DDRSS_DDRCTL_ECCCFG0	0x02980070	ECC Configuration Register 0
DDRSS_DDRCTL_ECCCFG1	0x02980074	ECC Configuration Register 1
DDRSS_DDRCTL_ECCSTAT	0x02980078	SEC/DED ECC Status Register (Valid only in MEMC_ECC_SUPPORT==1 (SEC/DED ECC mode))
DDRSS_DDRCTL_ECCCLR	0x0298007C	ECC Clear Register
DDRSS_DDRCTL_ECCERRCNT	0x02980080	ECC Error Counter Register
DDRSS_DDRCTL_ECCCADDR0	0x02980084	ECC Corrected Error Address Register 0
DDRSS_DDRCTL_ECCCADDR1	0x02980088	ECC Corrected Error Address Register 1
DDRSS_DDRCTL_ECCCSYN0	0x0298008C	ECC Corrected Syndrome Register 0
DDRSS_DDRCTL_ECCCSYN2	0x02980094	ECC Corrected Syndrome Register 2
DDRSS_DDRCTL_ECCBITMASK0	0x02980098	ECC Corrected Data Bit Mask Register 0
DDRSS_DDRCTL_ECCBITMASK2	0x029800A0	ECC Corrected Data Bit Mask Register 2
DDRSS_DDRCTL_ECCUADDR0	0x029800A4	ECC Uncorrected Error Address Register 0
DDRSS_DDRCTL_ECCUADDR1	0x029800A8	ECC Uncorrected Error Address Register 1
DDRSS_DDRCTL_ECCUSYN0	0x029800AC	ECC Uncorrected Syndrome Register 0
DDRSS_DDRCTL_ECCUSYN2	0x029800B4	ECC Uncorrected Syndrome Register 2
DDRSS_DDRCTL_ECCPOISONADDR0	0x029800B8	ECC Data Poisoning Address Register 0. If a HIF write data beat matches the address specified in this register, an ECC error will be introduced on that transaction (write/RMW), if ECCCFG1.data_poison_en=1
DDRSS_DDRCTL_ECCPOISONADDR1	0x029800BC	ECC Data Poisoning Address Register 1. If a HIF write data beat matches the address specified in this register, an ECC error will be introduced on that transaction (write/RMW), if ECCCFG1.data_poison_en=1
DDRSS_DDRCTL_ADVECCINDEX	0x02980374	Advanced ECC Index Register
DDRSS_DDRCTL_ECCPOISONPAT0	0x0298037C	ECC Poison Pattern 0 Register
DDRSS_DDRCTL_ECCPOISONPAT2	0x02980384	ECC Poison Pattern 2 Register

11 References

- Texas Instruments: *AM65x/DRA80xM Processors Technical Reference Manual*
- Texas Instruments: *AM65x/DRA80xM EMIF Tools*
- Texas Instruments: *AM65x/DRA80xM Processors Errata*

Copyright © 2019, Texas Instruments Incorporated

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2019, Texas Instruments Incorporated