Application Report
SPNA126–February 2011

ECC Handling in TMSx70-Based Microcontrollers

Frank Noha

ABSTRACT

This application report describes the Flash and RAM ECC handling methods for TMSx70-based microcontrollers in general.

To use the Flash/RAM ECC, the single error correction and double error detection (SECDED) module in the TMSx70 has to be configured accordingly. This application report covers typical software configurations of the SECDED modules and briefly explains some essential basics.

NOTE: To see if Flash ECC or RAM ECC is supported for your microcontroller and the respective Flash wrapper specification for technical details, see the TMSx70 device-specific data sheet.

Contents

List of Figures

List of Tables

1 Error Detection and Correction Techniques

In microcontrollers, the data stored in the memory could get corrupted over a period of time due to several reasons. This creates a necessity for the system to verify the correctness of the data before or while executing the software in the memory. The Error detection techniques help the system to check the integrity of the data in the memory.

Parity - This technique requires additional bits to be added corresponding to the data. Parity only detects single bit failures in a memory. This is a simpler well known error detection technique.

ECC - Adding Error Correcting Code (ECC) bits along with the original data is a technique that helps in detecting errors as well as correcting errors. The ECC scheme used on the TMSx70 devices is able to detect up to 2-bit failures and correct single bit failures.

The implementation and behavior of these error detection schemes makes it necessary for you to understand certain restrictions when using them in the application or while debugging the system. This application report covers only the ECC technique for TMSx70-based microcontrollers.

NOTE: Refer to the TMSx70 microcontroller-specific data sheet to know the kind of error detection technique supported.

1.1 Single Error Correction and Double Error Detection (SECDED)

SECDED is the hardware module embedded in the Flash/RAM wrapper that generates ECC, compares, detects the error bits and corrects if it is a single bit error.

- SECDED is capable of correcting single bit errors and detecting multiple bit errors.
- SECDED requires a total of eight ECC check bits associated with each 64-bit wide data word and its corresponding address.
- SECDED is adapted based on Flash/RAM technology design of the controller. Some designs have two SECDED modules that operate in parallel the results of which are compared and accepted only if both are same.

The SECDED logic in the respective memory wrapper calculates and checks the ECC bits in the TMSx70 microcontrollers. The logic is different for different memory types and CPUs.

1.1.1 TMSx70 Memory Types

- **Flash – F021, F035**
  - In case of Flash, the corresponding ECC values have to be programmed along with the data (program). The logic is adapted according to the corresponding flash technology of the microcontroller. Refer to the microcontroller specific datasheet to know more technical details of the SECDED module and ECC space mapping. (1)
- **RAM – TCRAM, ESRAM**
  - In case of RAM, the hardware generates the corresponding ECC check bits for the data during a write operation and stores them in the respective ECC memory location. During a memory read, the ECC bits in the ECC space are read along with the data to detect or correct any error.

    The RAM wrapper implementation is based on the CPU type.

1.1.2 TMSx70 CPU Types

The TMSx70 microcontrollers may be of any of the following CPU types from ARM®.

- ARM7TDMI®
- Cortex™-M3
- Cortex-R4

(1) nowECC™ is a TI proprietary tool.

In case of Cortex-R4 based CPU, the ECC encoding/decoding is done by the CPU (in-built), whereas, in case of Cortex-M3 and ARM7TDMI-based CPUs, the ECC encoding/decoding is done by the RAM wrapper. The main advantage of having the encoding/decoding within the CPU is to speed up the accesses by removing the time consuming SECDED block.

1.2 Flash Wrapper With ECC

You have to generate the ECC for the data and program it into the ECC space before enabling the ECC.

During the read operation, the 64-bit data together with the 19 address bits (21:3) pass through the ECC generator to produce the 8 check bits. These eight calculated ECC check bits are then XOR'ed with the stored check bits associated with the address and the read data. The 8-bit output is the syndrome.

For more information on mathematical ECC check bit calculation, see Section 1.5.

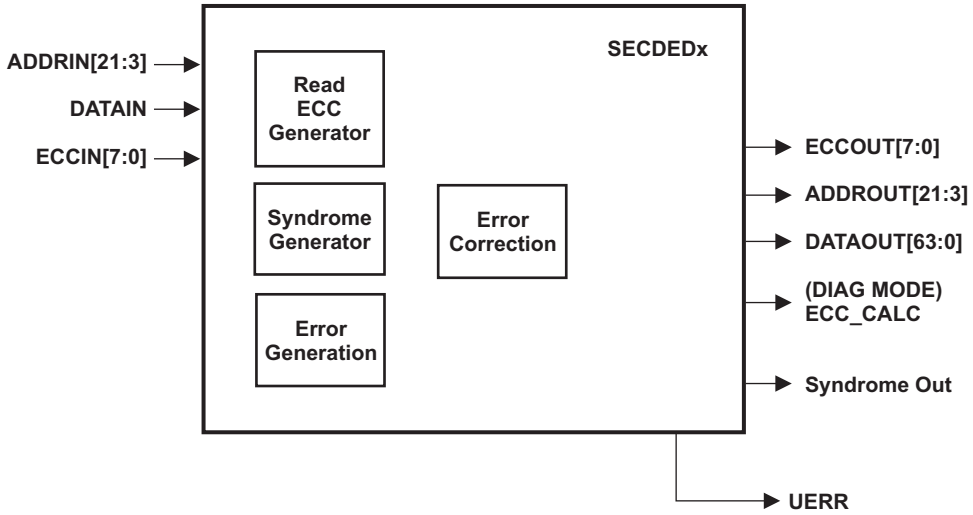

Figure 1. SECDED Block Diagram of Typical Flash Wrapper

The syndrome is decoded to determine one of three conditions:

- No error occurred.
- A correctable error occurred.
- A non-correctable error occurred.

NOTE: A single bit error in the address field is considered to be a non-correctable error.

1.3 RAM Wrapper With ECC

In case of RAM, the ECC calculation is done by the hardware and written into the corresponding ECC space.

Figure 2 describes the block diagram of a typical SECDED in a RAM wrapper.

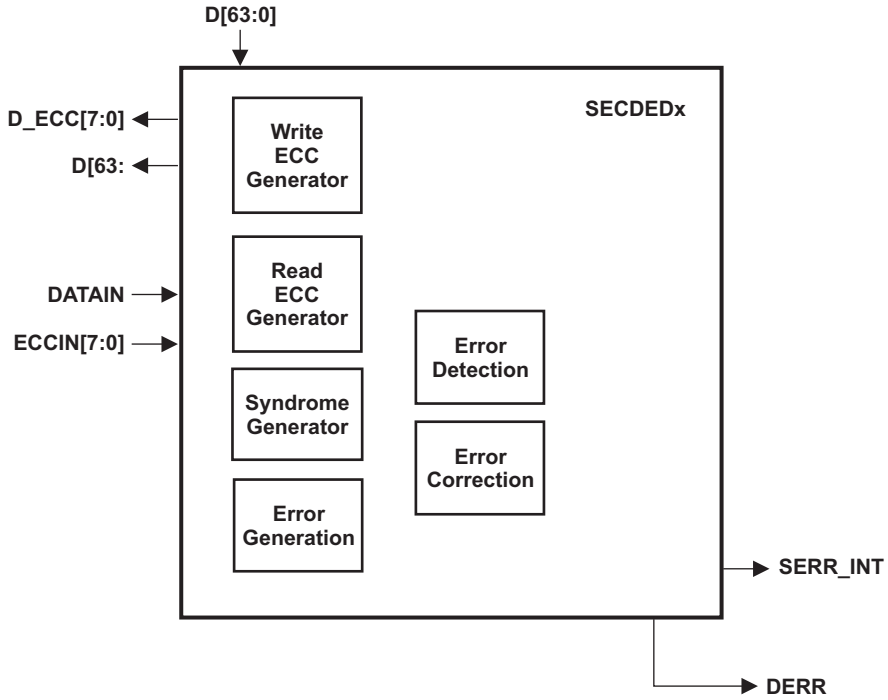

Figure 2. SECDED Block Diagram of a Typical RAM Wrapper

1.4 ECC Mapping in TMSx70 Microcontrollers

To be able to access the ECC bits associated with normal data words, a separate memory space has been defined in the TMSx70 based microcontrollers.

This memory space is offset by 4M bytes from the normal program or data memory area. Each 64-bit data word has corresponding 8-bit ECC check bits.

In case of Flash, the ECC memory mapping for F035 and F021 are illustrated in Section 2.

In case of RAM when ECC is enabled, these ECC bits are updated on every write access to the normal data word and the application can check the ECC bits by accessing the ECC space.

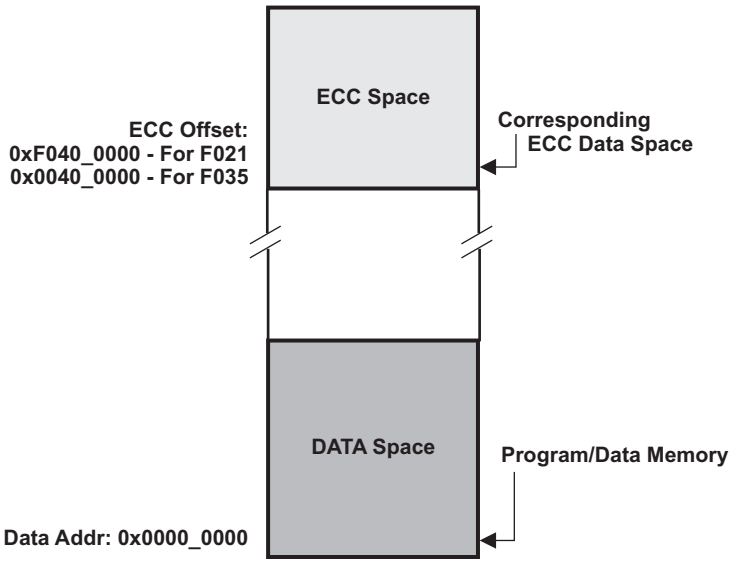

Figure 3. ECC Space Mapping

1.5 Error Correcting Code (ECC)

ECC is encoded based on modified Hamming code. Table 1 illustrates how the eight bit ECC is carried out for a 64 bit data [63:0] including 19 address bits [21:3].

Table 1. ECC Encoding

		82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
		Participating Address bits																		
ADDR_MSW_LSW	ECC Bit	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03
0007F_00FFFF00_FF0000FF	7													X	X	X	X	X	X	X
7FF80_FF0000FF_FF0000FF	6	X	X	X	X	X	X	X	X	X	X	X	X							
	5						X	X	X	X	X	X	X							
19F83_C0FCC0FC_C0FCC0FC	4			X	X			X	X	X	X	X	X						X	X
6A78D_39E338E3_38E338E3	3	X	X		X		X			X	X	X	X				X	X		X
2A9B5_A699A699_A699A699	2		X		X		X		X			X	X		X	X		X		X
0BAD1_15571557_15571557	1				X		X	X	X		X		X	X		X				X
554EA_B4D1B4D1_4B2E4B2E	0	X												X	X					

Participating Data bits																																						
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27
								X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X								X	X	X	X	X
X	X	X	X	X	X	X	X																	X	X	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X										X	X	X	X	X	X	X										X	X	X	X	X
X	X							X	X	X	X	X	X			X	X							X	X	X	X	X	X			X	X			
		X	X	X				X	X	X				X	X			X	X	X				X	X	X				X	X			X	X	X
X		X			X	X		X			X	X				X	X				X	X		X			X	X			X	X			X		
			X		X		X		X		X		X	X	X				X		X		X		X		X		X	X	X				X	
			X						X							X			X						X											

Participating Data Bits																										CPU Parity[1]	Check Bits[2]
26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	
X	X	X																	X	X	X	X	X	X	X	X	Even	ECC(7)
X	X	X																	X	X	X	X	X	X	X	X	Even	ECC(6)
X	X	X										X	X	X	X	X	X	X										ECC(5)
			X	X	X	X	X	X			X	X							X	X	X	X	X	X			Even	ECC(4)
			X	X	X				X	X			X	X	X				X	X	X				X	X	Odd	ECC(3)
X	X		X			X	X			X	X		X			X	X					X	X			X	Odd	ECC(2)
X		X		X		X		X	X	X				X		X		X		X		X		X	X	X	Even	ECC(1)
		X						X	X									X						X	X			ECC(0)

(1) Each ECC[x] bit represents the XOR of all the data bits marked with X in the same row.

(2) For devices that do not use address in the ECC calculations, remove the address bits from the calculation

NOTE: For more information on whether or not the Address bits (ADDR_MSW_LSW) are to be included in the ECC calculation, see the microcontroller-specific data sheet. This depends on the CPU and memory type.

Table 2. Example of ECC Values With Address

ADDR 21:0	Data High 127:64	Data Low 63:0	ECC 7:0
2415D8	F126E546	9A03FA6F	7C
0952B8	21D94D7E	B18B4F04	3A
02C580	F70C3A2D	EC8835ED	60
117B40	0ED9FB58	3E03C60D	6B
3DDB80	02324C15	A80EFA23	20
35D008	C34B6BF3	8FBD9E0F	4F
3F7180	FC31972C	D3EB454F	E9
3EED68	7BAF4225	4DEE03BB	B3
263938	446F1271	8DA56AF6	F0
21A9B8	98A582BA	EF7C951D	E8

Table 3. Example of ECC Values Without Address

Data High 127:64	Data Low 63:0	ECC 7:0
954F6D2F	2992A9B6	AA
8F8342C3	E7DE1D53	14
554B0A86	A8F07BDB	41
19F2DA66	14780AF1	60
5D80C176	A04CFED0	01
2B54902B	C4E77D0F	84
9190D774	01AEA191	97
D072D410	BD4E690F	CF
8F7FF177	6D1AD8A0	4F
2F92B288	D3E1A7BD	DD

2 Flash With ECC

The two types of Flash available are F035 and F021. The ECC check bit computing method remains the same for both of them but the mapping differs and is illustrated below.

2.1 Flash ECC Mapping

There is a difference in ECC space organization for F035 and F021 technology.

2.1.1 F035 Flash ECC Mapping

The ECC space of F035 Flash wrapper has each ECC byte mirrored three times as shown in Figure 4. Each read accesses four data words (128 bit) and 2 ECC bytes (16 bit).

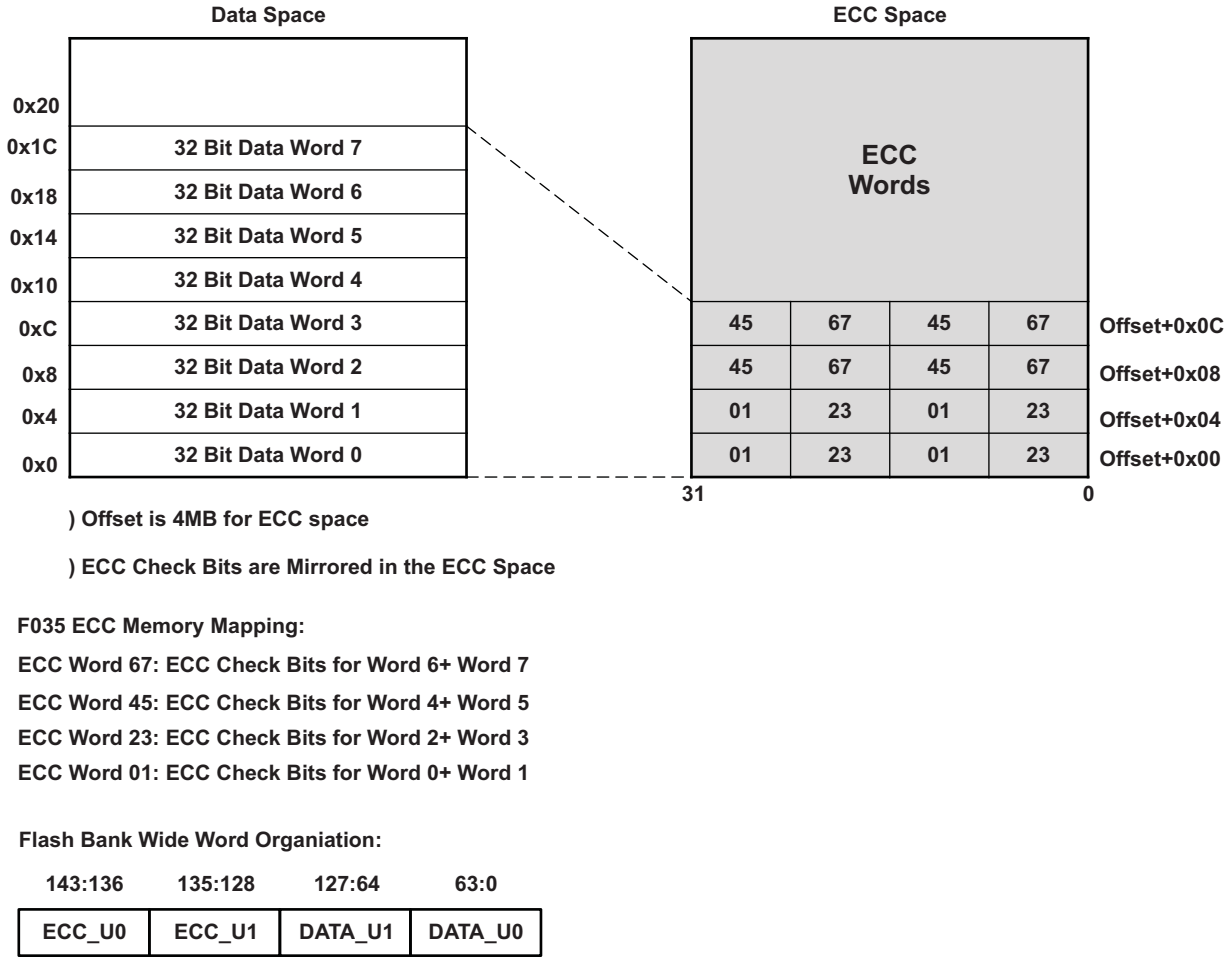

Figure 4. Data and ECC Space Mapping for F035 Flash in TMSx70

2.1.2 F021 Flash ECC Mapping

The F021 ECC space has no mirroring. The F021 Flash Bank is either 144 bit or 288 bit wide.

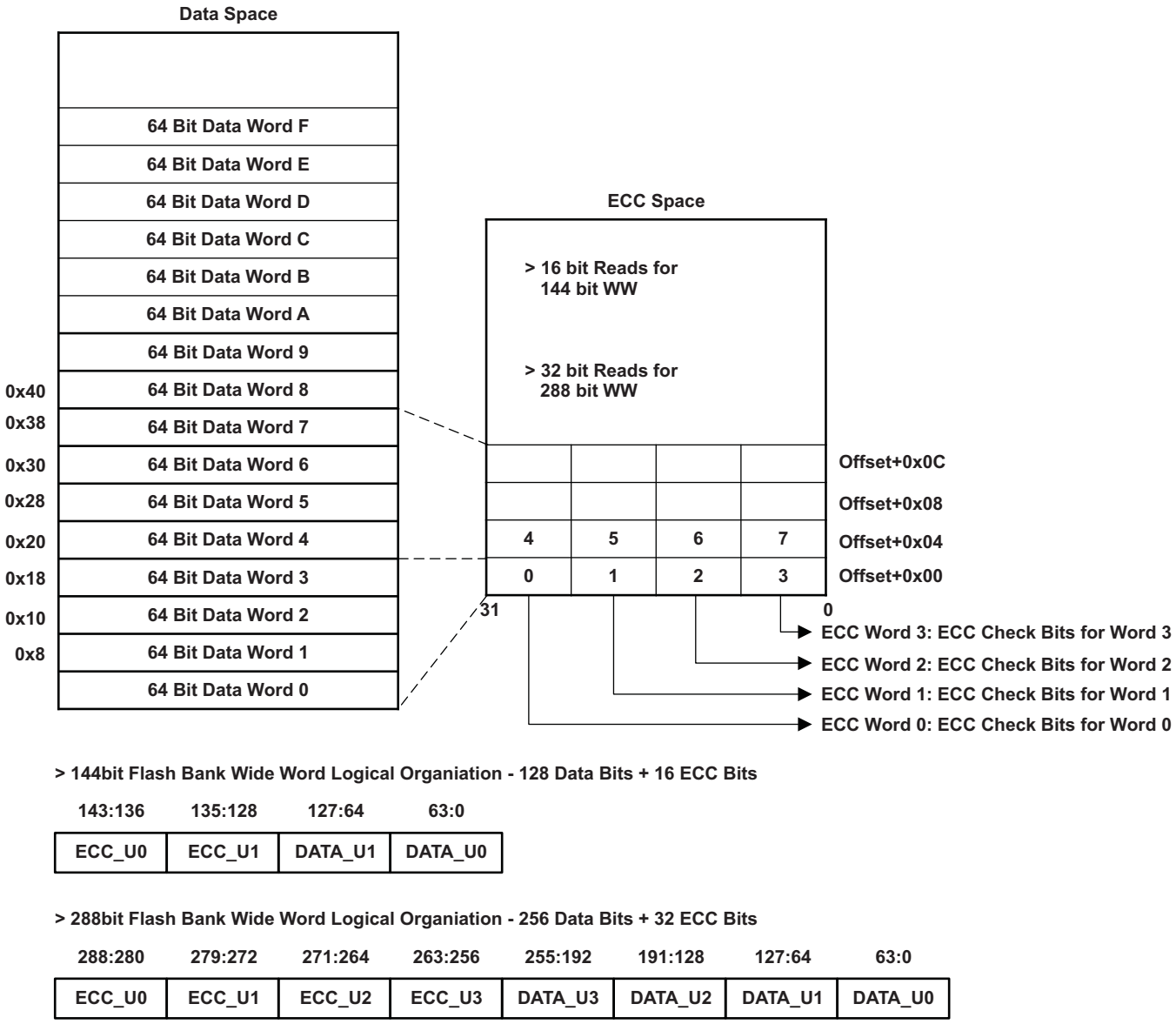

Figure 5. Data and ECC Space Mapping for F021 Flash in TMSx70

2.2 TI Flash ECC Generation Tool – nowECC™

To employ ECC for error checking the Flash ECC space has to be loaded with ECC check bits corresponding to data. We would need the following TI tools:

- nowECC - Tool to generate ECC file for a given program/data file.
- nowFlash™ - Tool to program the program/data and ECC file.

In the following illustration the program/data file is `prg.out`, the generated ECC file by the now ECC tool is `prg_ECC.out`.

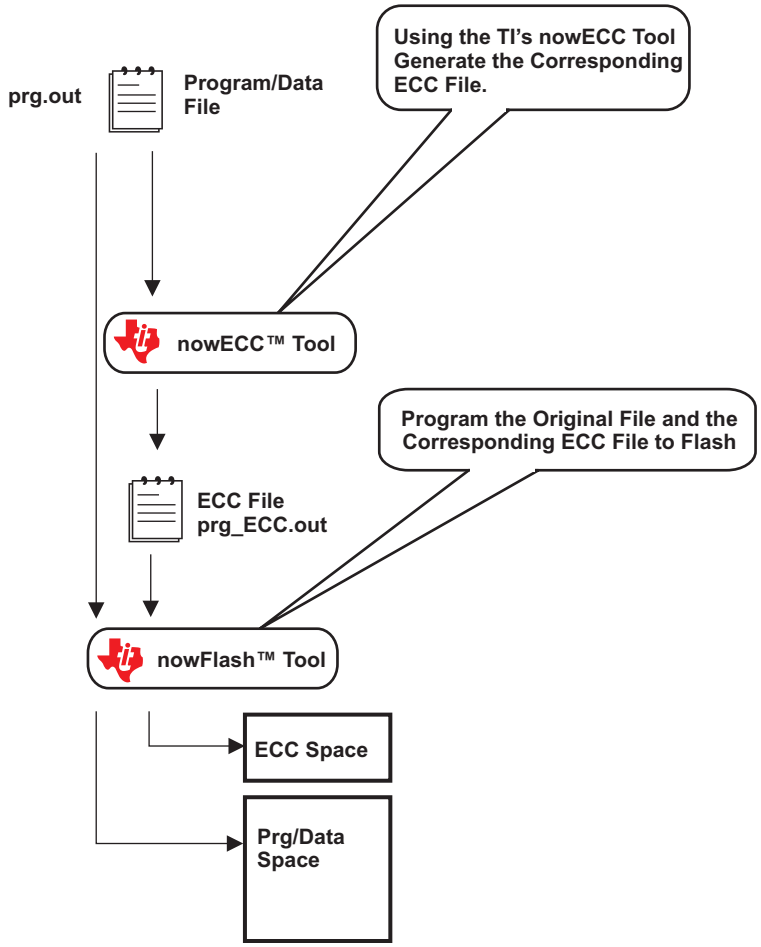

Figure 6. Programming Flash ECC

For programming the file/data:

1. Calculate ECC for the data that is to be programmed (Tool - nowECC).
2. Program the data along with the calculated ECC. (Tool - nowFLASH).

NOTE: For more information on nowECC tool usage, see the *nowECC Generation Tool User's Guide v2.14* (SPNU491).

During software execution:

1. Enable ECC in the respective memory wrapper, along with the interrupts as required.
2. You should handle the single bit/double bit errors in the ISR as required by the system.

Double bit ECC error interrupt is a high priority non-maskable hardware interrupt. This feature is to make sure that the software does not continue to execute when a double bit error is detected. It would be unsafe to continue any safety critical application in such a scenario, and the double bit ECC error service routine should handle it accordingly.

NOTE: In case of R4, the double bit ECC errors are mapped to ESM channel group3, which cause a data abort.

2.3 Enabling and Disabling Flash ECC

The following describes the general algorithm to be followed to enable or disable the ECC check:

1. Make sure that the ECC section is programmed before enabling the ECC.
2. Enable/disable single error correction/detection.
3. Enable/disable appropriate error interrupts.
4. Enable ECC/SECDED. (1)
5. In case of ECC error interrupts, handle it in the ISRs.

The ECC (SECDED) is globally enabled or disabled by a 4-bit EDACEN register. The ECC logic is disabled when the four bits are 0101. Any non-0101 combination value stored in the EDACEN register enables the ECC logic. It is advised to enable the ECC logic with *0xA* value stored in the EDACEN to prevent any soft error from disabling the logic.

Instructions to enable ECC for M3- and ARM7-based CPUs:

```
FLASH_Ptr->FEDACCTRL1_UN.FEDACCTRL1_ST.EDACEN_B4 = 0x0A; /*Enabling  ECC */
```

Instructions to enable ECC for R4-based CPUs:

In case of controllers with R4 CPU, the ECC check logic is within the CPU; this needs to be enabled after enabling the ECC in the Flash wrapper. Enabling the Flash wrapper ECC would keep the status registers in the Flash wrapper updated.

Enable Flash wrapper ECC:

```
FLASH_Ptr->FEDACCTRL1_UN.FEDACCTRL1_ST.EDACEN_B4 = 0x0A; /*Enabling ECC */
```

Enable ECC check within R4 CPU:

```
MRC   p15, #0, r1, c9, c12,     #0          ; Enabling Event monitor states
ORR   r1,   r1,   #0x00000010
MRC   p15, #0, r1, c9, c12,     #0          ; LDR R3,  = 0x00000010 ; (set 4th bit of PMNC
register)
MRC   p15, #0, r1, c9, c12,     #0

MRC   p15, #0, r1, c1, c0,      #1          ; Enabling ATCM
ORR   r1, r1,  #0x1, <<25
DMB
MRC   p15, #0, r1, c1, c0,      #1
ISB
```

Instructions to disable ECC for M3- or ARM7-based CPUs:

```
FLASH_Ptr->FEDACCTRL1_UN.FEDACCTRL1_ST.EDACEN_B4 = 0x05; /* Disable ECC check */
```

Instructions to disable ECC for R4-based CPUs:

In case of controllers with R4 CPU, the ECC within the CPU needs to be disabled. Following this the ECC in the Flash wrapper should be disabled.

(1) Only in case of R4, you must enable/disable the ECC within the R4 CPU following the wrapper configurations.

Disable ECC check within R4 CPU:

Continue with the following to disable the ECC within the R4 CPU:

```
MRC   p15, #0, r1, c1, c0,       #1         ; disable ATCM PARECCENRAM[0]
MVN   R0, #0x1 <<25
AND   R0, R1,  R0
DMB                                         ; Data memory Barrier to forcefully drain the store
buffer
MRC   p15, #0, r1, c1, c0,       #1
ISB                                         ; Instruction Synch. Barrier ensures previous
instruction has been executed

MRC   p15, #0, r1, c1, c0,       #1         ; disable ATCM ERRENRAM[0]
MVN   R0, #0x1, <<0
AND   R1, R1,  R0
DMB
MRC   p15, #0, r1, c1, c0,       #1
ISB
```

Disable Flash wrapper ECC:

```
FLASH_Ptr->FEDACCTRL1_UN.FEDACCTRL1_ST.EDACEN_B4 = 0x05; /* Disable ECC check */
```

NOTE: In case of Cortex-R4, it is recommended to flush the internal buffers using the DMB and ISB instructions after enabling/disabling the ECC and before reading/writing further data.

2.4 Debugging With ECC Enabled

The following ECC status registers should be monitored by the application software during an event of occurrence of an ECC error:

- Error Status Register [FEDACSTATUS]
  - One Fail Status Flag - Indicates error on *one fail*
  - Zero Fail Status Flag - Indicates error on *zero fail*
  - Multiple Bit Error Status Flag - Indicates multiple errors
- Correctable Error Address Register [FCOR_ERR_ADD] - Indicates corrected error bit address
- Uncorrectable Error Address Register FUNC_ERR_ADD - Indicates double bit error address
- Correctable Error Position Register - Indicates corrected error position

During error profiling the following registers are used.

- Single Error Correction Threshold Register - To program the threshold value.
- Error Correction Counter Register - Indicates number of times a correctable error occurred.

2.4.1 Single Stepping On Single/Multiple Bit Error

By single stepping with the debugger, you can simulate and verify single/multiple bit error scenarios.

With the Flash ECC and the ECC error interrupts enabled, when you try to single step using the debugger and encounters a single/multiple bit error, an error interrupt is generated by the hardware.

For single-bit error, the error bit will be corrected when read/executed by the CPU and you can continue with the code execution after clearing the relevant flags in the service routine.

For multiple-bit error, you may have to take appropriate steps depending on the application. Since the error cannot be corrected, continuing to execute the code would not be safe.

NOTE: In case of Cortex-R4, single stepping with ECC enabled will result in an abort.

2.4.2 On Flash ECC Single/Multiple Bit Error

Whenever a single bit error or double bit error occurs, you must clear the flags or the error event continues to be active and the error status is not updated.

For example:

If there is a *zero fail* at location 0x4000 and a *one fail* at location 0x4008, during the read with ECC enabled the *zero fail* is captured for the read at 0x4000, but the subsequent read at 0x4008 would not capture any error flag/error address/error position until the fail flag caused by the read at 0x4000 is cleared. The error flag can be cleared by writing a one into the corresponding flag bit.

Figure 7 illustrates the steps performed in the ECC error service routine on occurrence of a single bit error.

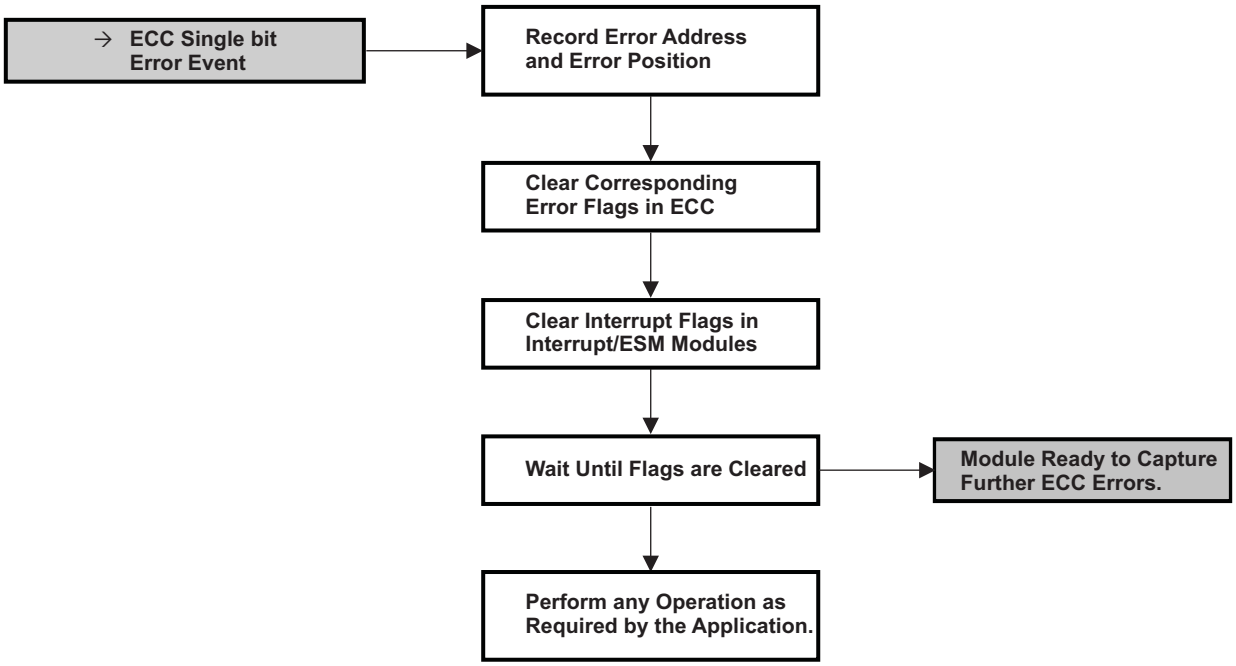

Figure 7. Service Routine Flow on an Occurrence of an ECC Error Event

2.5 *Diagnostic Mode*

The diagnostic mode of the SECDED module can be enabled to verify if the module is functioning as expected. The application software can verify the ECC functionality by switching to diagnostic mode at desired intervals or as when required.

Flash wrapper can be put in diagnostic mode to verify different supported logic. Six types of diagnostic modes are supported.

Diagnostic Control Register - DIAG_EN_KEY [3:0]

- DIAG_EN_KEY [3:0] = 0x5 - Enables diagnostic mode.
- DIAG_EN_KEY [3:0] = 0xA, or others - Disables diagnostic mode.

Inserting error bits in Flash

Below is an example to insert corrupt data bits in diagnostic mode and verify the generated error flags.

The following is the procedure to introduce corrupt data/address/ECC bits to verify the functionality:

1. Enable the SECDED.
2. Enable diagnostic mode.
3. Select SECDED 0 or 1.
4. Configure DIAG MODE1 - Diagnostic ECC data correction mode.
5. Write address bits 21:3 to FEMU_ADDR [21:3].
6. Write the original data with an error to FEMU_DLSW (lower 32 bit).
7. Write the original data with to FEMU_DMSW (Upper 32 bit).
8. Write desired ECC value to FEMU_ECC.
9. Now initiate SECDED action by triggering DIAG_TRIG =1.
   This calculates the ECC and write back the correct ECC to the FEMU_ECC register.
10. Check for relevant errors in Error Status Register.
    Check the address value in FCOR_ERR_ADD in case of correctable error.

Figure 8 illustrates the software flow for the above example on an F035 Flash microcontroller.

Figure 8. Introducing Error in Diagnostic Mode

3 RAM With ECC

The two types of RAM available are TCRAM wrapper and ESRAM wrapper. The TCRAM wrapper is specific to Cortex-R4 CPU type whereas ESRAM is for Cortex-M3 CPU type. The ECC check bit computation method remains the same for both.

3.1 Initializing RAM and ECC

In the case of RAM with ECC, when the device comes out of reset the data and ECC memory is in an unknown state. If ECC is enabled, any write access less than 64 bits forces a read-modify-write operation to calculate the correct ECC bits for the corresponding 64-bit data word. This read-modify-write most likely results in a false double bit error.

To avoid these false double bit errors, you must initialize RAM and its ECC space before enabling the ECC. You can do this by either of the following two methods.

3.1.1 Initialization Using Software

By enabling the Read-modify-write bypass (RMWCBYP) bit in the RAM Control register (RAMCTRL) the false double bit errors can be avoided. This bypasses the ECC checking while loading the data. With ECC enabled, the SECDED calculates and loads the corresponding ECC in the ECC space. After loading is complete, you should disable the RMWCBYP bit.

Steps to Initialize RAM:

1. Enable RMW (Read Modified Write) correction bypass.
2. Enable ECC.
3. Initialize RAM to known values (CPU write).
4. Disable RMW (Read Modified Write) correction bypass.

3.1.2 RAM Memory Initialization Using Hardware

In order to avoid false errors after power up and to efficiently initialize the RAM to a know state, the RAM wrapper has the capability to initialize the entire RAM array to a known state by writing all zeros to the data bits. If RAM ECC is enabled, the corresponding ECC check bits are also calculated and written to the ECC bits. After the initialization of RAM and RAM ECC, you need to make sure that the initialization is complete before making the first RAM write.

Steps to Initialize RAM:

1. Enable ECC.
2. Enable RAM memory initialization.
3. Wait until RAM memory initialization complete.

NOTE: You must make sure that the data into the RAM was written with ECC enabled if the data is to be read back with ECC.

3.2 Debugging With ECC Enabled

The following ECC status registers should be monitored by the application software during an event of occurrence of an ECC error:

- Memory Fault Detect Status Register – RAMERRSTATUS
  - Captures all error flags (Single error, multiple error, address decode failure, data compare logic fail).
- Single Error Address Register - RAMSERRADDR
  - Captures the error address when detecting a single bit error.
- RAM Error Position Register - RAMERRPOSITION
  - Records the binary encoded error position of which a single error is detected.

- Double Error Address Register - RAMDERRADDR
  - Captures the error address when detecting a double bit error.

3.2.1 Single Stepping On Single/Multiple Bit Error

With the RAM ECC enabled, when you try to single step using the debugger and encounter a single/multiple bit error, no error interrupts will be generated by the hardware even if the interrupts are enabled by the application software. Although the single error correction will happen when the word with the error bit is read by the CPU.

This is because when you execute from RAM and try to single step or add a software breakpoint, it would introduce unintended error interrupts, since the software breakpoints will change the opcodes at those addresses. These error interrupts are blocked by the hardware only while single stepping, so that the debug flow is not affected.

In case of Cortex-R4, single stepping with ECC on results in an abort.

3.2.2 On RAM ECC Single/Multiple Bit Error

Whenever a single bit error or a double bit error occurs, you must clear the flags or the error event continues to be active and the error status is not updated.

3.3 Disabling/Enabling ECC During Run Time

It is recommended to either use the ECC or not, throughout the application; however, you must be careful when disabling or enabling the ECC in between program executions. Also, there might be a scenario where the ECC may have to be disabled and then re-enabled for a specific task, or while accessing a specific location during run time.

While enabling or disabling the ECC you need to make sure that the ECC enable/disable is complete before any next write/read instructions are executed.

For example, if a specific section of data does not have its ECC programmed, you should disable ECC while accessing those locations.

3.3.1 Steps to Enable ECC

1. Enable ECC.
2. Wait until ECC enable instruction is complete.

Instructions to enable ECC for M3- and ARM7-based CPUs:

```
ECC_Ptr->RAMCTRL_UN.RAMCTRL_ST.ECC_ENABLE_B4 = 0xA;     /* enable ECC */
ECC_Ptr->RAMCTRL_UN.RAMCTRL_ST.ECC_ENABLE_B4;           /* wait for write to complete */
```

To enable ECC for R4-based CPUs:

In case of controllers with R4 CPU, the ECC generation logic is within the CPU and this needs to be enabled. The ECC in the RAM wrapper should also be enabled before enabling the ECC in R4, this would keep the status registers in the RAM wrapper updated.

```
ECC_Ptr->RAMCTRL_UN.RAMCTRL_ST.ECC_ENABLE_B4 = 0xA;     /* enable ECC */
ECC_Ptr->RAMCTRL_UN.RAMCTRL_ST.ECC_ENABLE_B4;           /* wait for write to complete */
```

Enable R4 CPU ECC:

```
ENABLE_ECC_R4_RAM

      MRC p15,#0,r1,c9,c12,#0          ; Reading secondary Aux secondary Reg.
      ORR r1, r1, #0x00000010
      DMB
      MCR p15,#0,r1,c9,c12,#0          ; Enable export of the events in PMNC
      ISB                              ; To ensure the write before proceeding


      MRC p15, #0, r1, c1, c0, #1
      ORR r1, r1, #0x1 <<26            ; B0TCM ECC Check Enable
      ORR r1, r1, #0x1 <<27            ; B1TCM ECC Check Enable
      DMB
      MCR p15, #0, r1, c1, c0, #1
      ISB                              ; To ensure the write before proceeding

      MOV PC, lr
```

3.3.2 Steps to Disable ECC

1. Disable ECC.
2. Wait until ECC disable instruction is complete.

Instructions to disable ECC for M3- and ARM7-based CPUs:

```
ECC_Ptr->RAMCTRL_UN.RAMCTRL_ST.ECC_ENABLE_B4 = 0x5;   /* disable ECC */
ECC_Ptr->RAMCTRL_UN.RAMCTRL_ST.ECC_ENABLE_B4;         /* wait for write to complete */
```

Instructions to disable ECC for R4-based CPUs:

In case of controllers with R4 CPU, the ECC within the CPU needs to be disabled. Following this, the ECC in the RAM wrapper should be disabled.

```
_DISABLE_ECC_R4_RAM


      MRC p15, #0, r1, c1, c0, #1
      MVN R0,#0x1 <<26                          ; B0TCM ECC check disable
      AND R1 ,R1, R0
      MVN R0,#0x1 <<27                          ; B1TCM ECC check disable
      AND R1 ,R1, R0
      DMB
      MCR p15, #0, r1, c1, c0, #1
      ISB                                       ; To ensure the write before proceeding

      MRC p15,#0,r1,c9,c12,#
      MVN R0,#0x00000010                        ; Disable export of the events in PMNC
      AND R1 ,R1, R0

      DMB
      MCR p15,#0,r1,c9,c12,#0
      ISB                                       ; To ensure the write before proceeding

      MOV PC, lr
```

Disable RAM wrapper ECC:

```
ECC_Ptr->RAMCTRL_UN.RAMCTRL_ST.ECC_ENABLE_B4 = 0x5; /* disable ECC */
ECC_Ptr->RAMCTRL_UN.RAMCTRL_ST.ECC_ENABLE_B4;       /* wait for write to complete */
```

3.4 Introducing An Error Into RAM ECC

To check the ECC functionality of the RAM ECC, error can be introduced by the application software at known locations and verified.

The following steps can be followed to introduce an ECC error into the RAM and to check if the RAM logic is working as expected.

Steps to introduce an error:

1. Initialize RAM – As mentioned in Section 3.1.
2. Enable ECC – This switches on the ECC generation logic.
3. Write data to a specific RAM location – This generates and loads the corresponding ECC check bits.
4. Disable ECC – This switches off the ECC generation logic so as to enable you to corrupt data.
5. Wait until ECC disabled – Although this step is not mandatory, make sure that it flushes the buffers.
6. Change only the data (Flip one bit) – This corrupts only the data part.
7. Enable ECC error interrupt – This enables you to handle the error in the exception routine.
8. Enable ECC – This switches on the ECC generation logic again and creates a scenario to check the logic.
9. Read the corrupted data. – The logic performs an ECC check when you read the same data.
10. Check for single bit error event – The ECC logic should generate a single bit error.

    This flow can be used by the application at required intervals to check that the RAM ECC logic works fine.

3.5 Switching Between ECC and EDC Modes

ECC Mode and EDC mode:

Table 4. Switching Between ECC and EDC Modes

ECC Mode	Error correcting code - mode	Detects and corrects single bit error , detects double bit error
EDC Mode	Error detecting code - mode	Only detects single and double bit error

NOTE: In case of Cortex-R4, the ECC encoding/decoding is done within the CPU, hence, the following description is applicable only for Cortex-M3 and ARM7TDMI.

In ECC mode, the corrections are done in the RAM wrapper, it limits the maximum possible access frequency of the RAM. It is possible for the CPU to access the RAM at a higher frequency, by switching to EDC mode (disabling the error correction - EDC mode).

Sample code to switch to ECC mode:

```
ECC_Ptr->RAMCTRL_UN.RAMCTRL_ST.ECC_ENABLE_B4 = 0x5;         /* Disable ECC */
ECC_Ptr->RAMCTRL2_UN.EDACCMODE_B4 = 0xA;                   /* Enable Error detection and
                                                              correction */
ECC_Ptr->RAMCTRL_UN.RAMCTRL_ST.ECC_ENABLE_B4 = 0xA;        /* Enable ECC */
ECC_Ptr->RAMCTRL_UN.RAMCTRL_ST.ECC_ENABLE_B4 ;             /* wait for write to complete */
```

Sample code to switch to EDC mode:

```
ECC_Ptr->RAMCTRL_UN.RAMCTRL_ST.ECC_ENABLE_B4 = 0x5;         /* Disable ECC */
ECC_Ptr->RAMCTRL2_UN.EDACCMODE_B4 = 0x5;                   /* Enable Error detection
ECC_Ptr->RAMCTRL_UN.RAMCTRL_ST.ECC_ENABLE_B4 = 0xA;        /* Enable ECC */
ECC_Ptr->RAMCTRL_UN.RAMCTRL_ST.ECC_ENABLE_B4 ;             /* wait for write to complete */
```

In EDC mode, the error bits are only detected and you may have to either read the data again by switching to ECC mode if a single bit error is detected, or do any appropriate actions as required by the application.

At higher frequencies, you can operate in ECC mode with additional RAM wait states.

Sample code to enable RAM data wait states:

```
SYS_Ptr->WAITST_UN.WAITST_ST/WST_DEN_B1=0x1;                /* Enable RAM data wait-state */
```

The requirement of data wait states for higher frequencies is device specific.

Also note that in the ARM7 or the Cortex M3 CPU based microcontrollers, the ECC correction is done in the RAM wrapper. In the Cortex-R4 CPU-based devices, the ECC correction is done by the CPU itself.

NOTE: For more information on the supported clock frequency to operate RAM with ECC and wait states required to operate at higher frequencies, see the device-specific data sheet .

4 References

- *nowECC Generation Tool User's Guide v2.14* (SPNU491)

Appendix A Terminology

A.1 Terminologies Used in This Document

Error detection is the detection of corrupted bits in the data.

Error correction is the reconstruction of the original error free data.

Single bit error is the error scenario where one bit of a word is read/found inverted (One as zero or zero as one).

Double/multiple bit error is the error scenario where more that one bit of a word are read/found inverted.

Error on one fail is the error when a “one” bit is read as a “zero” in a word.

Error on zero fail is the error when a “zero” bit is read as a “one” in a word.

EDC mode is the error detection mode wherein the error bits are only detected.

ECC mode is the error detection and correction mode wherein the single error bits are detected and corrected.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Audio	www.ti.com/audio	Communications and Telecom	www.ti.com/communications
Amplifiers	amplifier.ti.com	Computers and Peripherals	www.ti.com/computers
Data Converters	dataconverter.ti.com	Consumer Electronics	www.ti.com/consumer-apps
DLP® Products	www.dlp.com	Energy and Lighting	www.ti.com/energy
DSP	dsp.ti.com	Industrial	www.ti.com/industrial
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Security	www.ti.com/security
Logic	logic.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Power Mgmt	power.ti.com	Transportation and Automotive	www.ti.com/automotive
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com	Wireless	www.ti.com/wireless-apps
RF/IF and ZigBee® Solutions	www.ti.com/lprf		
	TI E2E Community Home Page		e2e.ti.com

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