Freescale Semiconductor
Application Note

Document Number: AN3538
Rev. 0, 12/2007

Enabling and Checking DDR ECC on PowerQUICC™ II Pro

1 Introduction

The memory controller of the PowerQUICC™ II Pro family has built-in Error Checking and Correction (ECC) capabilities. ECC allows single bit errors to be corrected and other bit errors to be detected increasing the reliability of high frequency operation as well as improving data accuracy and system uptime. This application note explains how to configure the memory controller to use ECC and how to use some of its test capabilities to validate correct ECC operation.

2 Hardware/Software Setup

The hardware platform used was a MPC8360 MDS board, Rev. 2 (MPC8360EA-MDS-PB) populated with Rev. 2.1 silicon. Code Warrior version 8.7 build 61218 was used to develop and run the code. Although the code was developed on the MPC8360 MDS it can be used on other PowerQUICC II Pro development boards by simply using the appropriate board initialization file. The program's output is sent to serial port 1 configured to baud rate 57600, 8 bit, no parity, 1 stop bit, and no flow control.

Contents

1. Introduction 1
2. Hardware/Software Setup 1
3. Setting Up and Initializing ECC 2
4. Verifying ECC Setup 3
5. Running the Program 7
6. References 7

© Freescale Semiconductor, Inc., 2007. All rights reserved.

freescale™ semiconductor

3 Setting Up and Initializing ECC

Prior to attempting to configure ECC it is recommended that memory operation is validated without ECC enabled. In particular, DDR timings should be calculated and verified.

This section describes the modifications that were made to the MPC8360 MDS (MPC8360EA-MDS-PB) default CodeWarrior initialization file (*CW_INSTALLATION_PATH\PowerPC_EABI_Support\Initialization_Files\PQ2\MPC8360_MDS_Rev2_init.cfg*) in order to activate the ECC functionality.

In summary, the changes to the initialization file are:

- Enable data initialization
  - Optionally define a pattern to initialize memory
- Disable single- and multi-bit error detection
- Enable ECC

The following sections detail each of the configuration steps listed above. Note that for normal ECC operation other steps could be added to the init file, but given the descriptive nature of this CW project, this configuration is performed inside the main() function to illustrate the program's logic.

3.1 Enable Data Initialization

On power up, the contents of the data and ECC memories are unknown. There is no guarantee that the ECC bytes are valid for the patterns in data memory. Enabling ECC in this state could cause the detection of single and multiple bit errors when initially accessing any data in memory. Normally software would have to initialize all data and ECC memory; however, the PowerQUICC II Pro memory controller can be programmed to perform this function automatically.

```
#DDR_SDRAM_CFG_2 [added D_INIT=1]
writemem.l 0xE0002114 0x00401010
```

When the memory controller is enabled it will set all configured memory either to zero (default) or a preprogrammed pattern (see below) and will set the corresponding ECC bits to valid values.

3.2 Defining a Pattern to Initialize Memory (Optional)

Memory can be initialized to any 32-bit pattern by programming the DDR_DATA_INIT register, allowing easy recognition of any uninitialized memory areas. This configuration is optional and, if not used, memory will be initialized to all-zeros—the register's default value.

```
#DDR_DATA_INIT
writemem.l 0xE0002128 0x11223344
```

3.3 Disable Single- and Multi-Bit Error Detection

As exceptions may be raised during the data initialization phase due to the random values in uninitialized memory, single- and multi-bit error detection must be disabled. This requires setting the `[MBED]` and `[SBED]` bits in the `ERR_DISABLE` register.

```
#ERR_DISABLE[MBED]=1 and ERR_DISABLE[SBED]=1
writemem.l 0xE0002e44 0x0000000C
```

Note that later these error detection mechanisms should be re-enabled—in the init file (after enabling the DDR controller) or in the program.

3.4 Enable ECC

To enable ECC generation and checking requires setting the [ECC_EN] bit in the DDR_SDRAM_CFG register.

```
#DDR_SDRAM_CFG [added ECC_EN=1]
writemem.l 0xE0002110 0x63000000
```

DDR_SDRAM_CFG[ECC_EN] must be set again, when the DDR controller's logic is activated:

```
#Enable: DDR_SDRAM_CFG (added ECC_EN=1)
writemem.l 0xE0002110 0xe3000000
```

With these settings in the initialization file, the board is initialized with Error Checking and Correction enabled. The next paragraph describes a simple program that can be used to confirm that ECC has been successfully enabled.

4 Verifying ECC Setup

This paragraph describes a simple program which checks if the ECC functionality is active. This program can be found as an attachment to this application note (AN3538SW). The program tests ECC by generating single-bit errors on a memory range and later reading from that range so that a certain number of single-bit errors are generated. When the number of single-bit errors detected exceeds a pre-configured threshold (see Section 4.1.5, "ERR_SBE: Single-Bit ECC Memory Error Management"), an exception is raised and the interrupt handler (written in C) is called. The interrupt handler clears the error source and prints a message on the serial port identifying which type of exception was raised and how many times it was raised. Figure 1 depicts the flow of the program.

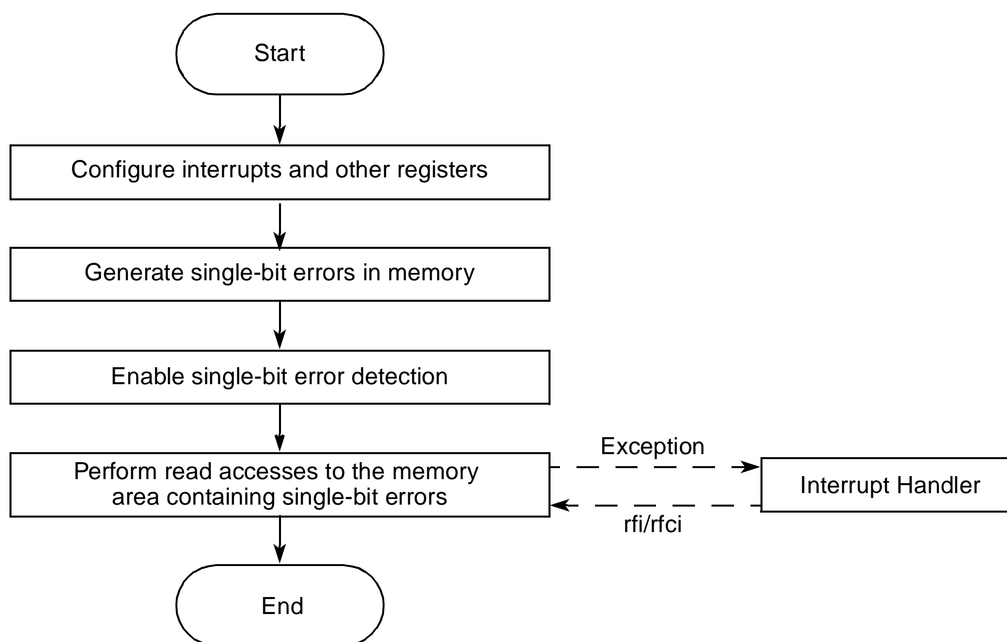

Figure 1. Program Flow

Note that any registers configured during the execution of the program could instead be set in the initialization file. However, for a better understanding of the logic behind the program, it was chosen to include their initialization in the program. Also, some of the configurations performed during program execution will deliberately cause errors and shouldn't be confused with the important configurations contained in the initialization file. The following sections detail each step of the program.

4.1 Configure Interrupts and Other Registers

In order to have interrupt-handling enabled several registers need to be configured, as described in the following sections.

4.1.1 SICFR: System Global Interrupt Configuration Register

Set DDR as the source for the highest priority interrupt (`SICFR[HPI]=0x4C`) and select critical interrupt (`cint`) as the output for the highest priority interrupt (`SICFR[HPIT]=0x2`). The highest priority interrupt can be redirected to other outputs, as Table 1 illustrates.

Table 1. Possible Outputs for the Highest Priority Interrupt

HPIT (Binary)	Interrupt Type	Interrupt Vector Address
00	`int` (external)	0x500
01	`smi` (system mgmt)	0x1400
10	`cint` (critical)	0xA00

By changing the value assigned to `SICFR[HPIT]` the output of the program, which is printed in the exception-handling routine, changes to one of the following:

```
InterruptHandler: 0x500 exception.
InterruptHandler: 0x1400 exception.
InterruptHandler: 0xa00 exception.
```

4.1.2 SIMSR_L: System Internal Interrupt Mask Register (Low)

The interrupts originated by DDR need to be unmasked. That is achieved by setting `SIMSR_L[12]=1`:

```
SIMSR_L = 0x00080000
```

4.1.3 MSR: Machine State Register

This register is used to enable the external and critical interrupts.

```
MSR[EE]=1
MSR[CE]=1
```

This register contains other settings that must not be overwritten. Therefore, the register is first read, OR'ed with 0x8080 (`EE|CE`) to set the desired bits, and written back.

4.1.4 ERR_INT_EN: Memory Error Interrupt Enable

The generation of interrupts when ECC single-bit errors are detected must be enabled. This register belongs to the DDR controller's register space.

```
ERR_INT_EN[SBEE]=1
```

4.1.5 ERR_SBE: Single-Bit ECC Memory Error Management

This register configures the number of single-bit errors to be detected before an exception is raised. An arbitrary value of 0xA0 was chosen for the test program.

```
ERR_SBE[SBET]=0xA0
```

4.2 Generate Single-Bit Errors in Memory

To generate single-bit errors in memory it is necessary to enable single-bit error injection and then perform write accesses to memory. At this point, error detection should not be enabled, therefore no single-bit errors will be detected or corrected.

4.2.1 ERR_INJECT: Memory Data Path Error Injection Mask ECC

Enable error injection when writing to memory and define an error injection mask:

```
ERR_INJECT[EIEN]=1
ERR_INJECT[EEIM]=0x01
```

To inject single-bit errors in memory the program performs write accesses to the memory area starting at the address pointed to by the unsigned char pointer `charPtr1`.

```
for (i = 0; i < NUM_CHARS; i++)
{
charPtr1[i] = 0xAB;
}
```

If copy-back caching is enabled, the value `NUM_CHARS` must be larger than the size of the data-cache to ensure that data is copied back to memory as a result of cast-outs due to reallocation of cache lines. If cache is not used, 1 byte in memory containing single-bit errors would be sufficient to cause any desired number single-bit errors—because all accesses access memory rather than hitting in data-cache.

4.2.2 ERR_INJECT: Memory Data Path Error Injection Mask ECC

Single-bit errors have already been generated and stored in memory. Single-bit error injection can now be disabled.

```
ERR_INJECT[EIEN]=0
ERR_INJECT[EEIM]=0
```

4.3 Enable Single-Bit Error Detection

4.3.1 ERR_DISABLE: Memory Error Disable

Enable single-bit error detection.

```
ERR_DISABLE[SBED] = 0 (enabled)
```

From this point, accesses to the memory region where single-bit errors have been stored will be detected and counted in `ERR_SBE[SBEC]`. When the value accumulated in `ERR_SBE[SBEC]` exceeds the value pre-configured in `ERR_SBE[SBET]`, the interrupt type that was previously configured will be generated.

4.3.2 Generate Single-Bit Errors In Memory

With single-bit error detection enabled, the following loop generates single-bit errors by performing read-accesses to the area of memory pointed to by `charPtr1`, where single-bit errors had been previously stored.

```
for (i = 0; i < NUM_CHARS; i++)
{
charPtr2[i] = charPtr1[i];
}
```

As the cycle progresses, the number of accumulated single-bit errors is stored in `ERR_SBE[SBEC]`. The following factors influence the counting of single-bit errors:

- The fact that data-cache is on (MPC8360: 32 kbytes)
- The number of bytes of memory read on each access (MPC8360: 8 bytes)

Only when the number of errors counted, `ERR_SBE[SBEC]`, reaches the preprogrammed threshold, `ERR_SBE[SBET]`, will the exception be raised and execution redirected to the interrupt handler.

4.4 Interrupt Handler

When an exception occurs the processor changes into supervisor mode, saves information about the state of the processor in specific registers (SRRx/CSRRx), and starts executing from the appropriate address of the interrupt vector. Different exceptions will cause execution to jump to different offsets in the exception table. The previous sections described how to configure the critical interrupt, which is the type of interrupt used in the test program. On a critical interrupt, execution jumps to address 0x0A00.

The first thing that is done when an exception occurs is to save all the registers into the stack so that later execution can be resumed from the point where it was interrupted by the exception. In this example, this register-saving macro is called `isr_prologue` (Interrupt Service Routine Prologue). Prior to exiting the exception state the interrupt handler restores the previously saved registers through the macro `isr_epilogue`. The format of the exception stack frame used is defined by the EABI (built on Power Architecture™ technology).

Both prologue and epilogue can be made common to a number of exceptions (External, Program, Floating Point, etc.), but the critical interrupt must have its own particular prologue and epilogue. This is because critical interrupts use `CSRR0` and `CSRR1` registers to save/restore the processor’s state while the other exceptions use `SRR0` and `SRR1`. Therefore, the critical interrupt’s prologue (named cisr_prologue) must save

CSRR0 and CSRR1 and the epilogue (named cisr_epilogue) must equivalently return from the interrupt handler with rfci (Return From Critical Interrupt) instead of the rfi (Return From Interrupt) used by other exceptions.

In the test program, the critical-interrupt handling is divided between two files containing the following elements:

- Source\eppc_exception.asm:
  - Interrupt vector (with, among others, the entry for the critical interrupt at address 0x0A00)
  - Prologue (cisr_prologue)
  - Call to external function InterruptHandler( )
  - Epilogue (cisr_epilogue)
- Source\interrupt.c:
  - Contains the InterruptHandler( ) function where a message is printed and the source of the interrupt is cleared

The interrupt-handler routine contained in the test program serves only to provide a very simple example of an interrupt-handling routine. As the interrupt to the core is level-sensitive it is essential that this interrupt handler clears the source of the interrupt. It is up to the user to add additional functionality that their application requires.

5 Running the Program

Using the hardware setup previously described, it is straightforward to run the test program and observe the printout on a serial terminal, where a line will be printed each time an exception is raised.

NOTE

When single-step debugging using the JTAG, the number of single-bit errors generated per step is higher than expected. Also, and more impacting, when single-stepping through the code and into the exception handler, the program is then unable to return from the interrupt-handling routine; therefore, to observe the complete execution of the program, the ‘run’ button should be used instead. Despite this limitation, the JTAG is still useful to observe the DDR controller’s settings changing when stepping through the code.

6 References

1. “MPC8360E PowerQUICC™ II Pro Integrated Communications Processor Family Reference Manual,” Freescale Document No. MPC8360ERM.
2. “e300 Power Architecture Core Family Reference Manual,” Freescale Document No. E300CORERM.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064
Japan
0120 191014 or
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
+1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc., 2007. Printed in the United States of America. All rights reserved.

Document Number: AN3538
Rev. 0
12/2007