Handling of External Memory EDAC Errors in LEON/GRLIB Systems

---

Application note

Doc. No GRLIB-AN-0004

Issue 1.1

2017-08-17

CHANGE RECORD

Issue	Date	Section / Page	Description
1.0	2015-10-28		First issue.
1.1	2017-08-17	4.1	Rephrased second bullet
		4.2	Described how a write_error trap can be generated on sub-word writes
		4.3	Updated data error trap handler example. It now uses pre-calculated DATA/TCB combinations instead of the EDAC diagnostic Read Bypass functionality.
		4.4	Added information on EDAC diagnostic Read Bypass

TABLE OF CONTENTS

1 INTRODUCTION....3
1.1 Scope of the Document....3
2 SYSTEM LEVEL BACKGROUND....3
3 SCRUBBING EXTERNAL MEMORY....4
3.1 Introduction....4
3.2 Baseline software approach....4
3.3 Hardware offloaded scrubbing....5
3.4 Additional considerations....5
4 HANDLING UNCORRECTABLE ERRORS....5
4.1 Background....5
4.2 LEON Processor Behaviour on Uncorrectable Errors....6
4.3 Data Trap Handler....7
4.4 Additional considerations....10

1 INTRODUCTION

1.1 Scope of the Document

This document contains general notes on handling correctable and uncorrectable errors in LEON/GRLIB system-on-chip devices.

2 SYSTEM LEVEL BACKGROUND

Note that system level design and radiation effects analysis is outside the scope of IP core and component support. This section is only intended to provide a background to clarify the design intent of the IP cores.

The approach that is assumed in the LEON3FT and LEON4FT designs and components for use in space applications is that the system designer designs the system so that correctable (single-bit in the case of BCH EDAC or single-nibble in case of Reed-Solomon codes) errors in memory happen at a controlled rate (probability per time unit) and uncorrectablet errors happen at a very low rate below the acceptable failure rate for the system so that the EDAC does not need to cover it.

If the system is designed to give such guarantees, the EDAC in the LEON3FT system's memory controller is then used to execute correctly despite the correctable errors that occur, and scrubbing in software or in hardware can be designed on top of that to prevent building up multiple single-bit errors over time to a double error.

There will always be a remaining failure rate caused by the risk of multi-bit errors and multiple correctable errors accumulating to uncorrectable errors faster than the scrubbing can repair them. It is the system designer's responsibility to do the necessary analyses to calculate the residual failure rate and decide what is acceptable for the mission.

In most cases, the system designer also adds a watchdog timer as a general safety net that resets the system in case of a crash. This can be due to multi bit error but also due to if there is a software bug that gets missed in validation or a hardware problem such as input clock glitch or power supply glitch.

In most designs it's preferred to fail immediately when getting double-bit errors since you can not guarantee that execution will continue correctly, you want to avoid incorrect behavior (for example sending bad messages to other systems) and you want to reboot to get to a known good state as fast as possible.

3 SCRUBBING EXTERNAL MEMORY

3.1 Introduction

The design used in LEON3FT systems such as UT699, UT699E, UT700, LEON3-RTAX and GR712RC is as follows:

- When a memory controller with EDAC enabled (such as FTMCTRL) detects a correctable error, the data will be temporarily corrected and delivered onto the on-chip bus. However a sideband signal will be asserted at the same time to indicate that a read access has been corrected.

- This sideband signal is connected to the AHB status register, and will cause it to copy in the address of the access into the address register, set the New error (NE) and correctable error (CE) bits, and raise an interrupt. The AHB status register is typically assigned to IRQ line #1 on systems designed by Cobham Gaisler, including the UT699 - UT700 and GR712.

3.2 Baseline software approach

A general software approach would then be to add a handler for IRQ 1, that checks the AHB status register and performs the appropriate action such as scrubbing the location by reading and writing it back, and then clear the AHB status register so it can latch in new errors.

To avoid cache effects on data load, one can use the load instruction with forced cache miss (LDA with ASI=1). The LEON3FT's cache is write through, therefore write accesses will not be affected by cache effects.

To make sure all data gets scrubbed regularly is is possible to have a background task with an address counter that reads through a bit of memory once in a while. This can just read and discard the data, since the IRQ handler above will take care of the actual scrubbing when needed.

Note that the AHB status register can only remember one error, so any correctable errors occurring before the current one has been handled (and the NE/CE bits have been cleared) will be temporarily corrected on the fly as usual but the correction will not be registered by the AHB status register and therefore not be corrected by the scrubbing scheme routine. Once the AHB status register is idle and someone reads again from this location, the correctable error will be discovered and handled.

DMA will complicate the issue, as the read/write sequence in the IRQ handler may get into a "race" with a DMA master writing to that same location. One will have to do some analysis to see if this can happen in the specific setup. Note that many times memory accessed via DMA will just be read/written once so that fact can be used in the analysis (for example you may find that areas accessed by DMA never need scrubbing).

3.3 Hardware offloaded scrubbing

Newer systems, such as the GR740 device or other devices that include the MEMSCRUB peripheral from Cobham Gaisler have a hardware memory scrubber. The MEMSCRUB IP cores provides periodic scrubbing of external memory. It uses a generic approach compatible with any memory controller with AHB interface and EDAC, designed in the way described in section 3.1. When the scrubber finds a correctable error, it will used a locked read/write cycle to perform the scrub operation, thus avoiding any possible race with DMA units or other masters.

The MEMSCRUB also includes all the AHB status register functionality and has a backward compatible software interface, therefore it can also be also be used with existing software scrubbing routines.

For more information on MEMSCRUB, please see the GRLIB IP Core User's Manual.

3.4 Additional considerations

In addition to the general notes in the previous section, the following items should be considered:

- In systems with a memory protection unit it may be required to check that the memory is writable.
- For uncorrectable errors (ERROR responses), software needs to check that the memory address is on the memory controller. The AHB (bus) controller in GRLIB replies with ERROR if an unmapped memory area is accessed.
- Always check that correctable error is in RAM before trying to correct

4 HANDLING UNCORRECTABLE ERRORS

4.1 Background

LEON/GRLIB systems such as UT699, UT699E, UT700, LEON3-RTAX, GR712RC and GR740 have the option to protect memory areas with error detection and correction (EDAC). When EDAC corrects an error in memory then a side-band signal will be raised and software will typically be notified about this through the AHB status register. How to use this functionality to implement memory scrubbing in order to prevent build up of errors is described in section 3 of this document.

When a memory controller detects an uncorrectable error it will respond with an AMBA ERROR response. An AMBA ERROR response will have the following effects:

- If the design has an AHB status register (present in all Cobham Gaisler LEON ASIC devices) then the error response will cause it to copy in the address of the access into the

address register, set the New error (NE) bit, and raise an interrupt. The AHB status register is typically assigned to IRQ line #1 on systems designed by Cobham Gaisler, including the UT699 - UT700 and GR712.
- If the processor is the AMBA master that initiated the transfer causing the AMBA ERROR response, then a trap will be generated.
- If the AMBA ERROR response is triggered by a peripheral capable of direct-memory access, such as a SpaceWire or Ethernet controller, then the peripheral will typically stop with an error condition being signalled through its status register and interrupt.

Uncorrectable errors and subsequent AMBA ERROR responses are a critical condition that is expected to be rare. By default the operating systems and run time environments distributed by Cobham Gaisler do not contain handlers for the traps caused by AMBA ERROR responses. This means that the processor will stop and enter error mode upon receiving an AMBA ERROR response. The watchdog timer will then time out and the system will be reset.

Some applications may want to handle uncorrectable errors by trying to mitigate them, shut down the affected software task, or try to log the occurrence of the uncorrectable error before performing a reset of the device.

4.2 LEON Processor Behaviour on Uncorrectable Errors

When a LEON processor receives an AMBA ERROR response for a memory location that the processor will use, one of the following conditions will happen:

- An AHB ERROR response received while fetching instructions will normally cause an instruction access exception (tt=0x1). However if this occurs during streaming on an address which is not needed, the I cache controller will just not set the corresponding valid bit in the cache tag. If the IU later fetches an instruction from the failed address, a cache miss will occur, triggering a new access to the failed address.

- An AHB ERROR response while fetching data into the data cache will normally trigger a data_access_exception trap (tt=0x9). If the error was for a part of the cache line other than what was currently being requested by the pipeline, a trap is not generated and the valid bit for that line is not set. A write_error trap (tt=0x2B) can be generated on sub-word writes that cause a read-modify-write cycle, where the EDAC error occurs on the read.

- An ERROR response during an MMU table walk will lead the MMU to set the fault type to Internal error (1) and generate an instruction or data access exception, depending on which type of access that caused the table walk.

Note that the default behaviour of the GRMON2 debug monitor is to break on these traps. If you are connected to the system with GRMON2 when an AMBA ERROR response is received by the processor then GRMON2 will break execution on the above traps. This can be prevented by starting GRMON2 with the flag *-nb*.

4.3 Data Trap Handler

In Cobham Gaisler BCC version 2, trap handlers can be installed with the *bcc_set_trap()* function. The listing below show an example that installs a trap handler and generates correctable and uncorrectable errors in a system with the FTMCTRL memory controller and AHB status register peripheral.

== main.c start ==

```
/*
 * This is an example on how the processor inject test check bit errors with
 * the FTMCTRL. A trap handler is registered for the dat access exception trap.
 * RAM accesses are performed to demonstrate how the AHBSTATUS and LEON trap
 * mechanism responds. 0, 1 and 2 BCH check bit errors are injected.
 *
 * USAGE:
 * Set the defines FTMCTRL_REGS and AHBSTAT_REGS to match your system.
 *   $ make
 *   $ grmon <dbglink> -u -edac -nb
 *   grmon2> wash
 *   grmon2> load bchexample.elf
 *   grmon2> run
 *
 * For more information, see Cobham Gaisler Application note GRLIB-AN-0004.
 */
#include <stdint.h>
#include <stdio.h>
#include <bcc/bcc.h>

#define FTMCTRL_REGS    0x80000000
#define AHBSTAT_REGS    0x80000f00
#define FTMCTRL_MCFG3_WB        (0x01 << 11)
#define FTMCTRL_MCFG3_TCB       (0xff <<  0)
struct ftmctrl_regs { unsigned int mcfg1, mcfg2, mcfg3, mcfg4; };
struct ahbstat_regs { unsigned int status, faddr; };
volatile struct ftmctrl_regs *mctrl     = (void *) FTMCTRL_REGS;
volatile struct ahbstat_regs *ahbstat   = (void *) AHBSTAT_REGS;

static const int TT_DATA_ACCESS_EXCEPTION = 0x09;
extern void tt_0x09(void);

/* VAL_DATA_A => BCH TCB = 0, VAL_DATA_B => BCH TCB = 0x7f */
static const unsigned int BCH_VAL_DATA_A = 0x00000028;
static const unsigned int BCH_VAL_DATA_B = 0x0001012c;

void write_data_and_tcb(volatile unsigned int *addr, uint32_t val, uint32_t tcb)
{
  uint32_t mcfg3 = mctrl->mcfg3;

  mcfg3 &= ~FTMCTRL_MCFG3_TCB;
  mcfg3 |= FTMCTRL_MCFG3_WB;
  mcfg3 |= tcb & FTMCTRL_MCFG3_TCB;
  mctrl->mcfg3 = mcfg3;
  *addr = val;
  mctrl->mcfg3 = mcfg3 & ~FTMCTRL_MCFG3_WB;
}

volatile unsigned int a = 1;
```

```
int main(void)
{
  bcc_set_trap(TT_DATA_ACCESS_EXCEPTION, tt_0x09);
  printf("a is a test word at RAM address %p\n\n", &a);
  static const uint32_t TCBS[3] = { 0x00, 0x01, 0x03 };
  for (int i = 0; i < 3; i++) {
    unsigned int readval;

    printf("TEST %d\n", i);
    printf("  Clear AHBSTAT\n");
    ahbstat->status = 0;

    printf("  Write a := %08x with %d TCB bit errors.\n", BCH_VAL_DATA_A, i);
    write_data_and_tcb(&a, BCH_VAL_DATA_A, TCBS[i]);

    readval = bcc_loadnocache((uint32_t *) &a);
    printf("  Read  a -> %08x\n", readval);
    printf("  Read  ahbstat -> (status=%08x, faddr=%08x)\n\n",
      ahbstat->status, ahbstat->faddr);
  }
  return 0;
}
```

== main.c end ==

The following code contains the example trap handler:

== tt_0x09.s start ==

```
/*
 * Demonstration trap handler for 'data access exception'. It writes the
 * failing location with a magic value.
 *
 * Assumes ahbstat contains address of AHBSTAT device to use.
 *
 * NOTE: This is code is for demonstration of the AHBSTAT/FTMCTRL diagnostic
 * interface.
 *
 * Registers at trap entry: l0=psr, l1=pc, l2=npc, l6=tt
 */
        .section ".text"
        .global tt_0x09
tt_0x09:
        /* Find AHBSTAT */
        set     ahbstat, %l3
        ld      [%l3], %l3
        /* Get failing address from AHBSTAT. */
        ld      [%l3 + 4], %l4
        /* Write a magic number to failing location. */
        set     0xDEADBEEF, %l5
        st      %l5, [%l4]
        /* Clear AHBSTAT and re-activate monitoring */
        st      %g0, [%l3]
        /* Re-execute trapped instruction. */
        jmp     %l1
         rett   %l2
```

== tt_0x09.s end ==

The following Makefile can be used to build the application:

== Makefile start ==

```
# Use BCC 2.0.1 or later to compile this example
CC      = sparc-gaisler-elf-gcc
CFLAGS  = -O2 -std=c99
PROG    = bchexample.elf
$(PROG): main.c tt_0x09.s
        $(CC) $(CFLAGS) main.c tt_0x09.s -o $@
clean:
        rm -f $(PROG)
```

== Makefile end ==

When putting the above files in one directory and issuing make with the BCC sparc-gaisler-elf-gcc compiler in the $PATH, the resulting *bchexample.elf* binary will demonstrate generation and handling of uncorrectable errors.

Following is the output when running the example on a GR712RC development board:

== Output start ==

```
grmon2> run
a is a test word at RAM address 0x40010744

TEST 0
  Clear AHBSTAT
  Write a := 00000028 with 0 TCB bit errors.
  Read  a -> 00000028
  Read  ahbstat -> (status=00000002, faddr=80000f04)

TEST 1
  Clear AHBSTAT
  Write a := 00000028 with 1 TCB bit errors.
  Read  a -> 00000028
  Read  ahbstat -> (status=00000302, faddr=40010744)

TEST 2
  Clear AHBSTAT
  Write a := 00000028 with 2 TCB bit errors.
  Read  a -> deadbeef
  Read  ahbstat -> (status=00000002, faddr=80000f04)


  CPU 0:  Program exited normally.
  CPU 1:  Power down mode

grmon2>
```

== Output end ==

When 1 TCB error injected, then the trap handler will not be engaged and the AHBSTATUS registers a correctable error. When 2 TCB errors are injected, then the trap handler will rewrite the failing location with a magic value and restart the AHB Status Register monitoring.

Starting GRMON without the -nb option will make the program abort instead of executing the data access exception trap.

4.4 Additional considerations

In addition to the notes in the previous sections, the following items should be considered:

- For uncorrectable errors (ERROR responses), software needs to check that the memory address is on the memory controller. The AHB (bus) controller in GRLIB replies with ERROR if an unmapped memory area is accessed.
- The trap may not be related to the currently executing instruction.
- The location of uncorrectable error may itself used by the error handling routine (for example a counter variable, or even the instructions performing the handling).
- There may be more than one uncorrectable error.

- There is no way for software to recover from uncorrectable error in the general case, except from restarting the system.
- When the FTMCTRL EDAC diagnostic Read Bypass is activated, then any memory read accesses will cause MCFG3.TCB to be updated. This includes instruction fetches made by the CPU instruction cache, CPU data cache and other DMA devices.
- The FTMCTRL EDAC diagnostic Read Bypass functionality requires that EDAC is enabled.