VORAGO VA10820 EDAC programming application note

April 19, 2017 Version 1.0

VA10800/VA10820

Abstract

The VA10820 has error detection and correction (EDAC) and scrub features to help address single-bit errors that might result from high energy particle strikes in space. The scrub feature works in conjunction with EDAC and periodically reads memory locations in sequence to correct single-bit errors that might have occurred. This application note covers the basics of using these two features.

Table of Contents

1 Overview of EDAC and Scrub

Hamming error codes are well documented. See the reference section of this document for an excellent website. Only a small introduction is offered here. By exclusive ORing certain bits in an 8, 16 or 32-bit word, a unique value, commonly called syndrome, can be created to correct a single-bit. By exclusive ORing all the data bits and parity bits, a double-bit error can be detected.

The VA10820 EDAC has a 5-bit syndrome for every 8-bits of data. This allows byte level operations on memory which is commonly done on microcontrollers with limited memory. When data is written to memory, the EDAC controller calculates the syndrome and stores it along with the data. When memory is read, the syndrome is read at the same time as the data and a calculation is performed. If a single-bit error (SBE) is found, the data (with error

corrected) is output to the system and a write-back automatically places the corrected information in the memory location. A multi-bit error (MBE) is detected but not correctable using the syndrome implementation of the VA10820.

There are separate, always enabled, EDAC circuits for the ROM and RAM. Both have separate interrupt enable bits for single-bit errors and multi-bit errors. The failing address is not stored. The interrupt service routine can decide if a system RESET is required or to just track the error frequency.

Table 1 – EDAC and Scrub support summary

	Counter	Interrupt enable	Scrub rate (once per x cycles)
ROM	SBE	SBE	0 – 16.7 M cycles
	MBE	MBE	
RAM	SBE	SBE	0 – 16.7 M cycles
	MBE	MBE	

The purpose of the scrub engine is to periodically walk through the memory looking for and automatically correcting single-bit errors. The engine only processes one address (of 32 data bits) per scrub cycle. For example, to scrub the entire 128 KB ROM space one time takes 32k scrub cycles.

There a separate scrub engines for the ROM (code space) and RAM (data space). Each has its own programmable period. If the period is set to zero, the scrub engine is off. The entire memory space is checked, 128 KB ROM and 32 KB RAM. It is not possible to set a subset of either memory block for scrub activity. Scrub is more important for locations that are not frequently updated which allows uncorrected errors to accumulate over time. An uncorrectable error would occur if there were more than one bit error per byte.

2 EDAC functionality

The full description of the block can be found in the VA10800/820 Programmers Guide. This section provides supplemental information on how use the block in a final application.

2.1 Register summary

The following table lists all the registers associated with EDAC, scrub and memory error handling.

System Configuration Peripheral		
Name	**Description**	**Overview and Use**
ROM_SCRUB	ROM scrub rate	Sets the period of the single read scrub operation that will correct any single-bit errors that have occurred.
ROM_SCRUB	RAM scrub rate	
ROM_TRAP_ADDR	Debug feature to artificially create a memory error	Used to cause a memory error so that the response can be tested.
ROM_TRAP_SYND	Manually created syndrome	Can be used to store syndrome information for the data in the trap address. This can allow the generation of SBE or MBE conditions.
ROM_TRAP_ADDR	Debug feature to artificially create a memory error	Used to cause a memory error so that the response can be tested.
ROM_TRAP_SYND	Manually created syndrome	Can be used to store syndrome information for the data in the trap address. This can allow the generation of SBE or MBE conditions.
IRQ_ENAB	Interrupt enable signals for the EDAC errors	Individual bits for RAMSBE, RAMMBE, ROMSBE and ROMMBE.
RST_STAT	Reports source of last RESET	MEMERR is one of the 6 sources. This can be used as a condition for conducting diagnostics if necessary.
RST_CNTL_ROM	Determines if ROM is reloaded for each RESET source	MEMERR is one for the 6 sources. Normally this bit is set so that fresh ROM data is fetched every time a memory error RESET occurs.
RST_CNTL_RAM	Determines if RAM is reset to zero (0x00) for each RESET source	MEMERR is one for the 6 sources. Normally this bit is set so that RAM is all initialized to zero after a memory error RESET occurs.
Interrupt Select Peripheral		
Name	**Description**	**Overview and Use**
IRQSEL	Directs the interrupt source signal to one of the NVIC inputs or generate a RESET	When a memory error occurs, it can generate an interrupt. That interrupt signal can connect to NVIC inputs 0-31 or immediately cause a reset with unique signature such as a Memory error (IRQ_DST_MERESET) or Watchdog (WATCHDOG)
INT_RAM_SBE	Pointer for RAM_SBE interrupt	Contains index number of destination for interrupt signal
INT_RAM_MBE	Pointer for RAM_MBE interrupt	
INT_ROM_SBE	Pointer for ROM_SBE interrupt	
INT_ROM_MBE	Pointer for ROM_MBE interrupt	

2.2 Enabling Scrub

Out of RESET, the RAM and ROM scrub registers are set to zero which translates to them being disabled. By setting these to a non-zero value (X), scrub is enabled and single address scrub operation occurs once every X CPU cycles. It is not possible to adjust the address range of scrub, it will always span the complete memory range. The CPU performance is never impaired by scrub cycles however a small current draw increase may be seen for very high scrub rates. For instance, Vorago measured a 6.3 mA increase when running at 50 MHz with both ROM and RAM scrub rate set to 2 (one scrub cycle for every 2 CPU cycles)

Figure 1 - Code snippet to show scrub rate being set

```
VOR_SYSCONFIG->ROM_SCRUB = 1000 ;    //  Set ROM scrub rate to once every 1k cycles
VOR_SYSCONFIG->RAM_SCRUB = 10000 ;   //  Set RAM scrub rate to once every 10k cycles
```

2.3 Enabling interrupts

When EDAC detects an error, it can generate an interrupt or cause a RESET.

When an MBE occurs, there is no way to correct it. In most cases, causing a system RESET via the interrupt select module is recommended. This will load fresh program data into the MCU from the external SPI memory.

SBE's can be monitored and decisions made based on the frequency of which they occur. In some instances, it may possible to completely shutdown a module if the error rate is above a certain threshold as it passes through a high radiation field.

The below code enables interrupts for single-bit errors and resets for multibit errors.

Figure 2 - Code snippet to show interrupts being enabled

```
case VOR_IRQ_EDAC:
    VOR_IRQSEL->INT_RAM_SBE = IRQ_EDAC_IRQn;
    VOR_IRQSEL->INT_RAM_MBE = IRQ_DST_MERESET;    //  force memory error reset if we get MBE
    VOR_IRQSEL->INT_ROM_SBE = IRQ_EDAC_IRQn;
    VOR_IRQSEL->INT_ROM_MBE = IRQ_DST_MERESET;    //  force memory error reset if we get MBE
    NVIC_SetPriority(IRQ_EDAC_IRQn, IRQ_EDAC_PRIORITY);
    NVIC_EnableIRQ(IRQ_EDAC_IRQn);
    break;
```

2.4 Checking that error handling code works

The VA10820 is equipped with a feature to allow a customized syndrome to be loaded for a single address in ROM and RAM. When this feature is enabled, it is possible to inject errors in the memory for test purposes.

Injecting a single-bit error will force the EDAC SBE count to be incremented once when that location is read. Then the EDAC logic will correct the single-bit error in the syndrome. All future read accesses to the address will not generate an EDAC error.

Multi-bit errors cannot be corrected automatically by the EDAC logic and they will continue to occur every time the location is read.

The below code shows one implementation for causing EDAC errors. The ROM trap syndrome was randomly selected and will generate a multi-bit error. Each time this location is read, an MBE event will be triggered.

Figure 3 – EDAC debug code example – introducing a MBE error in ROM and an SBE in RAM

```
//   added for debug purposes    *  introducing error in EDAC
      VOR_SYSCONFIG->ROM_TRAP_ADDR = 0x80010000   ;   //  enable and set wrong syndrome for address 0x10000
      VOR_SYSCONFIG ->ROM_TRAP_SYND = 0x1234   ;  //
      VOR_SYSCONFIG->RAM_TRAP_ADDR = 0x80006434   ;   //  enable and set wrong syndrome for address 0xf000
//  enable only one of the following three lines
//       VOR_SYSCONFIG ->RAM_TRAP_SYND = 0xbdef7 ; // case = no errors =>   syndrome for 0x5555555 should be this value
      VOR_SYSCONFIG ->RAM_TRAP_SYND = 0xbdeff   ; //  case = SBE =>        syndrome for
//       VOR_SYSCONFIG ->RAM_TRAP_SYND = 0xbde00 ; //  case = MBE =>        syndrome for

// end of code for debugging EDAC
```

2.5 Monitoring errors

When a memory error occurs and interrupts are enabled, the below interrupt subroutine can be used to isolate the cause of the error and report the counts for all the memory error sources.

Figure 4 - Example ISR to process errors

```
void xEDAC_Error_Details( )
{
    printf(",IRQ_END Reg: 0x%08x\n", VOR_SYSCONFIG->IRQ_END);
    if( VOR_SYSCONFIG->IRQ_END & SYSCONFIG_IRQ_END_RAMSBE_Msk )
        printf(",Data RAM single-bit error \n");
    if( VOR_SYSCONFIG->IRQ_END & SYSCONFIG_IRQ_END_RAMMBE_Msk )
        printf(",Data RAM multi-bit error \n");
    if( VOR_SYSCONFIG->IRQ_END & SYSCONFIG_IRQ_END_ROMSBE_Msk )
        printf(",Data ROM single-bit error \n");
    if( VOR_SYSCONFIG->IRQ_END & SYSCONFIG_IRQ_END_ROMMBE_Msk )
        printf(",Data ROM multi-bit error \n");


    VOR_SYSCONFIG->IRQ_CLR = 0xF;

    // record error tallies for all sources
    printf(",RAM_SBE Reg: 0x%08x\n", VOR_SYSCONFIG->RAM_SBE);
    printf(",RAM_MBE Reg: 0x%08x\n", VOR_SYSCONFIG->RAM_MBE);
    printf(",ROM_SBE Reg: 0x%08x\n", VOR_SYSCONFIG->ROM_SBE);
    printf(",ROM_MBE Reg: 0x%08x\n", VOR_SYSCONFIG->ROM_MBE);

}
```

3 Conclusions

The VA10820 has EDAC and scrub features which are essential to combating single event upsets that will occur in Space environments. This application note has provided several example operations for setting up these features and testing them. You should be able to quickly enable these features in your application.

4 Common questions and issues

1. What is the right scrub rate?
    a. Answer = This will depend upon the type of radiation environment the part will operate and the power consumption sensitivity. The important thing is to perform scrub frequently enough to prevent the accumulation of single-bit errors in each memory block. The following table shows some examples.

Scrub rate	Bus Frequency (MHz)	ROM / RAM	Time to scrub entire array (seconds)
1000	50	ROM	.655
1000		RAM	.164
4000		RAM	.655
1000	2	ROM	16.4
1000		RAM	4.1
4000		RAM	16.4

2. Why not reset the part whenever an MBE occurs?
    a. Answer = In most cases an MBE justifies a full reboot since the error is uncorrectable and may reside in code space or stack space. However, if an MBE occurs in data RAM, the user may decide this is not critical and may rely on software to refresh this area without a full boot.

5 Other Resources

VORAGO VA108x0 programmers guide:
http://www.voragotech.com/sites/default/files/VA10800_VA10820_PG_July2016revision1.16%5B4%5D.pdf

VORAGO MCU products: http://www.voragotech.com/VORAGO-products

VORAGO Application notes: http://www.voragotech.com/resources

VORAGO VA108xx REB1board user guide: Part of Board Support Package (BSP)
http://www.voragotech.com/products/reb1

Hamming code calculator -
http://www.ecs.umass.edu/ece/koren/FaultTolerantSystems/simulator/Hamming/HammingCodes.html