

Using EDAC RAM for RadTolerant RTAX-S FPGAs and Axcelerator FPGAs

Applies to EDAC Core from Libero IDE v7.1 or Older

Introduction

Actel's newest designed-for-space Field Programmable Gate Array (FPGA) family, the RTAX-S, is a high-performance, high-density antifuse-based FPGA with embedded user static RAM (SRAM). Based on Actel's new AX architecture, the RTAX-S family is the high-reliability version of its commercial counterpart – the Axcelerator FPGA family. RTAX-S is single-event-upset (SEU) enhanced, specifically designed for space and contains embedded triple-modular-redundancy (TMR) registers, transparent to the user, which provide protection against heavy ions. In space applications, storage elements like SRAMs are susceptible to the impact of heavy ions from cosmic galactic rays (CGRs). CGRs can collide with the silicon lattice of a RAM cell with sufficient photovoltaic energy to produce a change of state, thus invalidating the stored data and producing bit errors. These errors are called soft errors.

There are several ways to implement static RAM in space devices. One option is to do nothing to mitigate soft errors. This is acceptable if the quality of the data stored in the SRAM is insensitive to single bit changes, such as an image comprising millions of pixels or streaming video feed. However, it is not acceptable if data is sensitive to single bit changes, for example communication packet headers. Another option is to implement a special hardened SRAM circuit. This consumes more chip real-estate and may require custom or boutique processing, and therefore is an expensive proposition.

Actel has chosen to use a combination of the Axcelerator standard SRAM circuits and an error detection and correction (EDAC) intellectual property (IP) core. The core is accessed via Actel's SmartGen Macro Builder software and implements a class of linear block codes called shortened Hamming codes (see the "[Regular Hamming Codes and Shortened Hamming Codes](#)" section on page 2). The SRAM/EDAC core combination greatly mitigates the effects of soft errors. Error rates are better than 10^{-10} errors/bit-day.

Coding Theory Background

In recent years, there has been an increasing demand for efficient, reliable digital data transmission and storage systems. A major concern is the control of errors so that one can obtain reliable data reproduction.

Coding refers to a class of signal transformations designed to improve communications or data reliability. The object of channel encoding is to reduce the probability of bit errors. Linear sums (using modulo-2 arithmetic) of the parity bits are called parity-check codes. Parity-check codes are widely used for EDAC. Linear block codes are a class of parity check codes that are usually represented with an (n,k) notation where k denotes the number of message digits in a longer code word of n digits. Each unique message of k digits maps to a unique code word of n digits. The entire potential message space consists of 2^k distinct message sequences with each sequence forming what is referred to as a k -tuple (sequence of k digits or bits). Similarly, the entire code word space has 2^n code words or n -tuples. The mapping between the 2^k possible messages and the 2^n code word n -tuples can be accomplished by the use of a lookup table.

An important parameter of a block code is called the minimum distance. Minimum distance determines the random error-detecting and random error-correcting capabilities of a code. The greater the distance the less likely an error will be made in the decoding process because no two valid codes can exist within d_{\min} bits of each other.

When a single-bit error is added to a code word, then the resultant code word differs from the original in one position. If the minimum distance of a block code C is d_{\min} , any two distinct code vectors of C differ in at least d_{\min} places. For example, if $u=(1010010)$ and $v=(1110011)$, then the minimum distance is two since

the second and last bits differ. For this example code C, no error pattern of $d_{\min}-1$ or fewer errors can change one code vector into another.¹

EDAC

As mentioned earlier, heavy ions lead to soft errors in memory subsystems. In many computer systems, the memory contents are protected effectively by EDAC codes. These codes are usually implemented by employing redundant bits.

An error-correcting coding technique specifies how to add redundant bits to data to allow error detection and correction if one (or possibly more) of the resulting bits are changed. Linear block codes are named because each code word or vector is a linear combination of a set of generator code words that are segmented into a message of separate blocks of a finite length. One class of linear block codes are SEC/DED (single-error correcting/double-error-detecting) codes. One application of EDAC calls for the use of SEC/DED codes in memory and storage applications where data may become corrupted. The following section discusses the widely used Hamming codes and methods of implementing them.

Regular Hamming Codes and Shortened Hamming Codes

In recent years, there has been an increasing demand for efficient, reliable digital data transmission and storage systems. A major concern is the control of errors so that one can obtain reliable data reproduction.

Hamming codes are the first class of linear block systematic codes devised for error correction; these codes and their variations are widely used for error correction. For any positive integer $m>3$, there exists a Hamming code with the following parameters:

Code length: $n=2^m-1$

Number of data bits: $k=2^m-m-1$

Number of parity check symbols: $n-k = m$

Error correction capability: one error ($d_{\min}=3$)

The first few members of the class are (7,4), (15,11), (31,26), (63,57), and (127,120). The parity-check matrix can be used to generate parity-check bits during the process of encoding and to generate bits indicating the presence and location of errors – 'syndrome bits' – during the decoding process. The total number of 1's in a given row is related to the number of logic levels required to generate the parity-check or syndrome bits for the row. Hsiao developed a new class of SEC-DED codes obtained by shortening the Hamming Codes. He showed that by deleting columns from H, a new matrix H_0 could be developed such that the code length could be reduced by L bits (where L is the number of deleted columns), thus reducing the number of logic levels and minimizing the complexity of the hardware implementation. A modified coding scheme was developed from the regular Hamming codes called shortened Hamming codes.²

Shortened Hamming codes can have arbitrary code lengths.

Code length: $n=2^m - L - 1$

Number of information symbols: $k=2^m-m-1-L$

Number of parity check symbols: $n-k = m$

Minimum distance: $d_{\min} = 4$

This code has a minimum distance of four and can correct one error and detect two errors.

-
1. Lin, Shu and Daniel J. Costello, Jr, Error Control Coding: Fundamentals and Applications (New Jersey: Prentice Hall, 1983).
 2. . M.Y. Hsiao, "A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes," IBM Journal of Research and Development [online] (Vol. 14, No. 4) (July 1970), avoable from: <http://www.research.ibm.com/journal/rd/144/ibmrd1404l.pdf>.

User Word, EDAC RAM Word, and RTAX-S or Axcelerator RAM Word

The RTAX-S and Axcelerator FPGA families contain 36 (RTAX1000S, AX1000) and 64 (RTAX2000S, AX2000) blocks of embedded memory. Each block is a 4.5k variable-aspect-ratio dual-port RAM. The allowable variable aspect ratios are 128x36, 256x18, 512x9, 1kx4, 2kx2 or 4kx1. However, only the memory blocks in one column can be cascaded in both width and depth to build larger blocks. This allows a maximum of 16 blocks in the AX2000 to be cascaded in both width and depth to build larger blocks.

For EDAC RAM Actel used the shortened Hamming code, which fully utilizes the data width of RTAX-S or Axcelerator RAM.

Actel chose shortened Hamming codes (18,12), (36,29), and (54,47) for RTAX-S RAMs with data widths of 18, 36, and 54 bits, respectively. The relationship between different data port widths is shown in Table 1.

Table 1 • Relationship of Different Data Port Widths

Data Port	Width		
	Users wdata/rdata width	8	16
EDAC module wdata/rdata width	12	29	47
Axcelerator RAM wdata/rdata width	18	36	54

As a result, if a user requests a 16-bit EDAC Protected RAM, the backend Axcelerator RAM has a word length of 36. Although SmartGen generates EDAC modules assuming input data word widths of 8, 16, or 32 bits, the designer can use up to 12, 29, or 47 bits of input data for the EDAC module. Please see the "Modifying the EDAC RAM Generated in SmartGen" section on page 11 for more information. Refer to "Appendix B" on page 16 for more information on shortened Hamming codes.

Implementation of Shortened Hamming Encoding and Decoding

In order to implement shortened Hamming codes, the user needs an encoder that converts the input data to coded words. The encoding can be done by matrix multiplication of the input word with a generator matrix. The encoding process is essentially the same as the original Hamming Code.

Below is an example given in systematic form of matrix multiplication for an (n,k) code. Specifically, this example addresses a (7,4) code. Let $u = (u_0, u_1, u_2, u_3)$ be the message to be encoded and let $v = (v_0, v_1, v_2, v_3, v_4, v_5, v_6)$ be the corresponding code word. Then

$$v = (v_0, v_1, v_2, v_3, v_4, v_5, v_6) = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

By matrix multiplication, the digits of the code word v were obtained:

$$v_6 = u_3$$

$$v_5 = u_2$$

$$v_4 = u_1$$

$$v_3 = u_0$$

$$v_2 = u_1 + u_2 + u_3$$

$$v_1 = u_0 + u_1 + u_2$$

$$v_0 = u_0 + u_2 + u_3$$

The code word corresponding to the message (1 0 1 1) is (1 0 0 1 0 1 1).

The following example shows the generator matrix for shortened Hamming code (18,12):³

1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1
1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1

Therefore, if the input data is "100000000000," then the coded word is "111000100000000000." Note there are six extra bits in the coded word – these are the syndrome bits, which indicate the presence and location of errors.

During decoding, the coded bits enter the decoding circuit; the syndrome bits are computed using a parity check matrix.

The single-bit error correction is accomplished by using a look-up table (Table 2). For example, if the six syndrome bits are "000000," then the coded bits are correct. If the syndrome bits are "100000," then an error exists at the first bit of the coded word. For any possible sequence of syndrome bits, Table 2 indicates which bits need to be corrected. Double error detection is accomplished by examining the number of '1's in the syndrome vector. If the syndrome contains an even number of '1's, then either a double-error pattern or a multiple-even-error pattern has occurred.

Table 2 • Syndrome Bits and Lookup Table

Syndrome Bits	Lookup Table
000000	00000000000000000000
100000	10000000000000000000
010000	01000000000000000000
001000	00100000000000000000
000100	00010000000000000000
000010	00001000000000000000
000001	00000100000000000000
111000	00000010000000000000
000111	00000001000000000000
110100	00000000100000000000
001011	00000000010000000000
110010	00000000001000000000
001101	00000000000100000000
110001	00000000000010000000
001110	00000000000001000000
101100	00000000000000010000
010011	00000000000000000100
101010	000000000000000000010
010101	000000000000000000001

3. . M.Y. Hsiao, "A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes."

Syndrome $(1 \times n-k) = \text{coded word } (1 \times n) * \text{parity check matrix } (n-k \times n)^T$, where T denotes the transpose of the matrix.

This parity check matrix was derived from the method by M.Y.Hsiao. This matrix was generated so that it results in the smallest circuit implementation. The parity check matrix for the same shortened Hamming code (18,12) is shown below:

1	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	0	0	0	1	0	1	0	1	0	1	0	0	0	1	0
0	0	1	0	0	1	0	0	1	0	1	0	1	1	0	1	0
0	0	0	1	0	0	1	1	0	0	1	0	1	1	0	0	1
0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	1	1
0	0	0	0	0	1	0	1	0	1	0	1	1	0	0	1	0

In summary, during decoding perform the following:

Syndrome $(1 \times n-k) = \text{coded word } (1 \times n) * \text{parity check matrix } (n-k \times n)^T$

Decoded word $(1 \times n) = \text{lookup table (syndrome) + coded word}$

The following sections discuss how to generate EDAC RAM for prototyping the RTAX-S FPGA family with the RTAX-S or Axcelerator families employing Actel software. The next sections also describe how the user can instantiate these components in the code. Since the EDAC RAM uses the shortened Hamming Code described above, the hardware performs single-bit error correction and double-bit error detection.

Functional Overview of EDAC RAM

The EDAC RAM provides an interface compatible with the RTAX-S or Axcelerator RAM transparent access mode. In addition to encoding and decoding, it reads the contents of memory periodically and resolves all correctable errors where possible. This periodic operation is called scrubbing. The EDAC RAM block, shown in Figure 1, consists of an RTAX-S or an Axcelerator RAM block and an EDAC block. The EDAC block, which is the heart of the EDAC RAM, contains a shortened Hamming code encoder, a shortened Hamming code decoder, a background scrubbing control unit, two sets of multiplexers, and a timer. The background scrubbing control block controls the multiplexers that provide access to the user memory and the background scrubbing process.

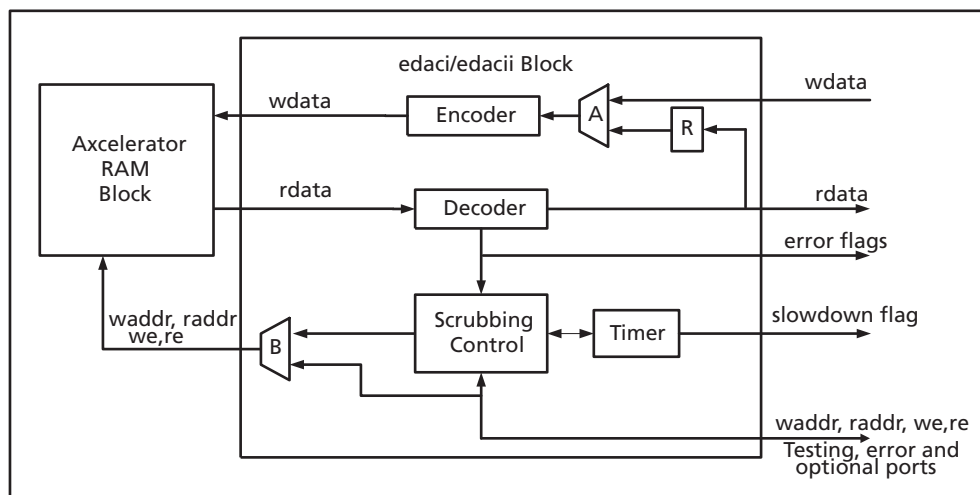


Figure 1 • Functional Diagram of EDAC RAM

During a user write, the write data is encoded by the encoder and then fed into the RTAX-S RAM block. The write address and enable (waddr and we) are fed into the RAM block through the B set of multiplexers as shown in Figure 1. During a user read, the read address line and enable (raddr and re) are

fed into the RAM block through the MUX B, then a coded word is read into the decoder and reconstructed into its original word. In addition, during this process, the decoder corrects any error detected in the coded word. During a user read/write, the scrubbing control block halts the background scrubbing process until there is no user read/write access. The background scrubbing process only runs when the read and write enable signals for memory (we and re) are inactive. The user read/write always takes priority over background scrubbing.

The user RAM interface to the EDAC RAM is identical to Axcelerator RAM access, but access time is increased due to additional encoding, decoding, and scrubbing processes.

The user can generate an EDAC RAM using the SmartGen tool. When the user generates an EDAC RAM from SmartGen, the tool generates a top-level shell for RAM encoding, an RTAX-S or Axcelerator RAM block and an EDAC module (Figure 2). Please see "Appendix A" on page 15 for the port definition of the EDAC RAM.

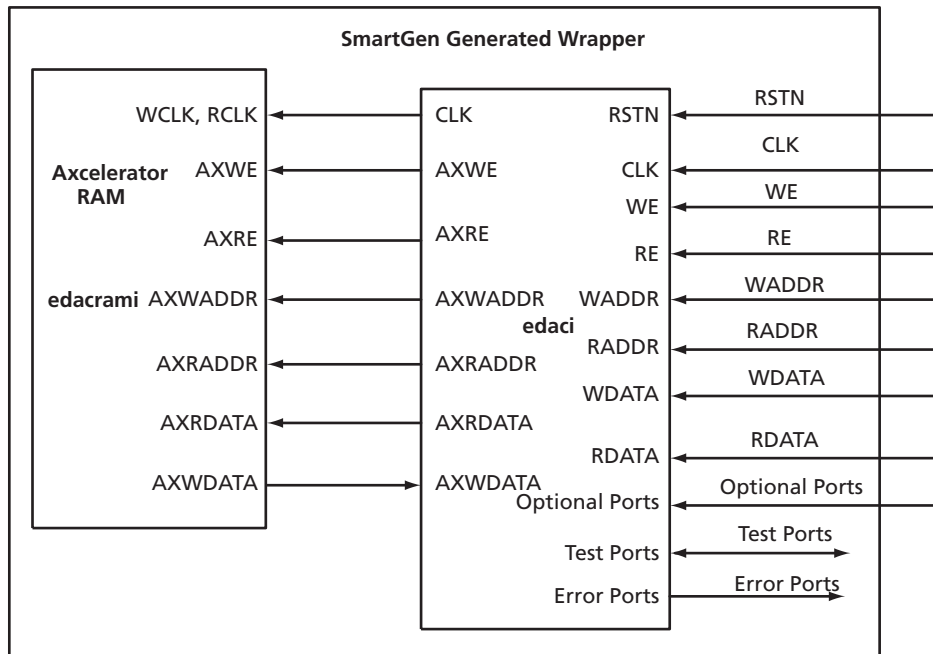


Figure 2 • EDAC Module in EDAC RAM

Features of EDAC RAM from SmartGen

The SmartGen Macro Builder can generate an EDAC module with various configurations. The main features of the EDAC module are:

1. 8-, 16-, 32-bit word widths
2. Background refreshing with variable refresh rate
3. Read and write clocks from the same clock source or separate read and write clocks
4. Encoder/decoder supports correction of one error and detection of two errors
5. Variable RAM depth support from 256 to 4k words

Scrubbing Control Block

The scrubbing control block handles access to the memory background scrubbing process. During this process, the background scrubbing control generates control signals for two sets of multiplexers (A and B in Figure 1 on page 5). These control signals switch the generation of the read/write data, read/write addresses, and read/write enables of the RTAX-S or Axcelerator RAM to the background scrubbing control

unit. A read address counter inside the background scrubbing control unit generates addresses, which sweep through all RAM locations. Meanwhile, the control unit monitors the decoder's error flag to determine whether or not to write the corrected data back into the RTAX-S or Axcelerator RAM. When STOP_SCRUB is set to high, the scrubbing stops. The output signal SCRUB_DONE indicates the scrubbing logic has gone through all RAM cells. The user can monitor the SCRUB_DONE signal and turn the STOP_SCRUB signal on or off accordingly. The scrub logic also sets the SCRUB_CORRECTED flag high when the scrubbing logic has corrected a data error; the output bus CADDR tells the user the address of the corrected error. Table 3 describes the scrubbing signals.

Table 3 • Scrubbing Signals

Signal	Description
SCRUB_STOP	Low to turn on scrubbing, High to turn off scrubbing
SCRUB_DONE	Transition from Low to High means scrubbing is done
SLOWDOWN	High when scrubbing cannot finish within designated period
SCRUB_CORRECTED	High indicates scrubbing logic has corrected one error
CADDR	The address of RAM has been corrected
TMOUTFLG	High indicates timer has expired

The user needs to exercise care when using the EDAC RAM with different read and write clocks. For this type of EDAC RAM, if the write clock is faster than the read clock, set the W2R to the write-clock-to-read-clock frequency ratio. This gives the state machine inside the EDAC module time to reject a correction request in case there is a user write to the same RAM location that the read state machine has just read. If the read clock frequency is faster or equal to write clock, set the W2R to "0000."

If RCLK is greater than or equal to WCLK, then W2R = "0000" and if RCLK < WCLK, then W2R= integer2vector (WCLK/RCLK). The maximum value of W2R is "1111," so WCLK can be a maximum of 15 times faster than RCLK.

Refresh Rate

The user can control the scrubbing rate to help reduce power consumption. The maximum allowable refresh period is given by:

$$\text{Refresh period} = \text{clock period} * \text{tmout}$$

where tmout is the refresh timer's time-out value (set in the SmartGen GUI; see the "Design Flow with SmartGen" section on page 8).

When the read address counter returns to 0 (the condition for asserting the STOP_SCRUB signal), the scrubbing process will stop until it receives a time-out from the EDAC timer. This implementation helps the background scrubbing process save power because needless refresh cycles are eliminated. When a timer time-out is indicated (TMOUTFLG is asserted high; see Table 4) and the read address counter has NOT returned to 0, the SLOWDOWN flag is asserted (Table 4). This warns the system to slow down memory accesses or risk losing data. If the system can guarantee idle time to memory access, the SLOWDOWN flag can be ignored. (This is the exception rather than the rule; monitoring SLOWDOWN is necessary in most systems). The tmout setting should be larger than the RAM depth to ensure that a complete refresh cycle can be completed within the refresh period.

Table 4 • TMOUTFLG Signal and SLOWDOWN Signal

Signal	Type	Active	Size	Description
SLOWDOWN	out	high	1	Optional flag when scrubbing cannot finish within designated period
TMOUTFLG	out	high	1	High indicates timer has expired

Test Ports

The EDAC RAM is capable of reading the coded data directly from the RTAX-S or Axcelerator memory, which is useful during debugging. If the input signal BYPASS is set to 1, the user has direct access to the RTAX-S or Axcelerator RAM block. During this time, the scrubbing process is stopped. In order to access the full coded word, port wp is used to write to the coded parity bits directly and port rp is used to read the coded parity bits directly. The widths of wp and rp are given in [Table 5](#) for various data widths.

Table 5 • Relationship of User Ports and Test Ports

Data Signals	Width		
User WDATA/RDATA width	8	16	32
EDAC module WDATA/RDATA width	18	36	54
EDAC module wp/rp	6	7	7

Error Flags

The EDAC RAM has two error flags, CORRECTABLE and ERROR, which allow the user to monitor error correction and detection ([Table 6](#)).

Table 6 • Error Flags

CORRECTABLE	ERROR	Description
0	0	No error has occurred
1	0	One bit error occurred
0	1	Two or more bit errors detected

Design Flow with SmartGen

The SmartGen tool enables the user to generate the EDAC RAM. The designer can generate an EDAC RAM with either a single read/write clock or independent read and write clocks. [Figure 3 on page 9](#) shows the SmartGen GUI for EDAC RAM generation, while [Table 7 on page 10](#) lists the user-definable fields.

To generate the EDAC RAM:

1. Invoke SmartGen
2. Open a new macro file. From the File menu, select New, or click the New button
3. Specify the family. Select RTAX-S or Axcelerator from pull-down menu
4. From the Macros menu, select RAM
5. Click the EDAC RAM tab
6. Set the desired variations for EDAC RAM and click Generate

Click the Test Ports and Error Flags check boxes to generate additional ports. Users cannot generate a Test Port without selecting Error Flags.

Use the generated netlist in the design to take the complete design through a synthesis flow and to generate a netlist that can be imported into Actel's Designer software. By employing the Designer software's Timer tool, the user can set timing constraints and perform timing-driven layout to meet timing needs.

Limitations of the SmartGen GUI

The SmartGen GUI has the following limitations:

- Supports only specific width and depth values
- Does not support pipelined read for RTAX-S or Accelerator RAM
- Read and write enables must always be present
- The No-Enable case is not supported

Users can change the input word width by modifying the netlist located in the "Adding More User Ports" section on page 11.

Core Utilization of EDAC RAM

The user can generate the EDAC RAM from SmartGen and instantiate it in their code. However, designers should be cautious about the core utilization and ensure that it has enough space to fit the EDAC RAM. Table 8 on page 10 describes the utilization for three configurations of the EDAC RAM.

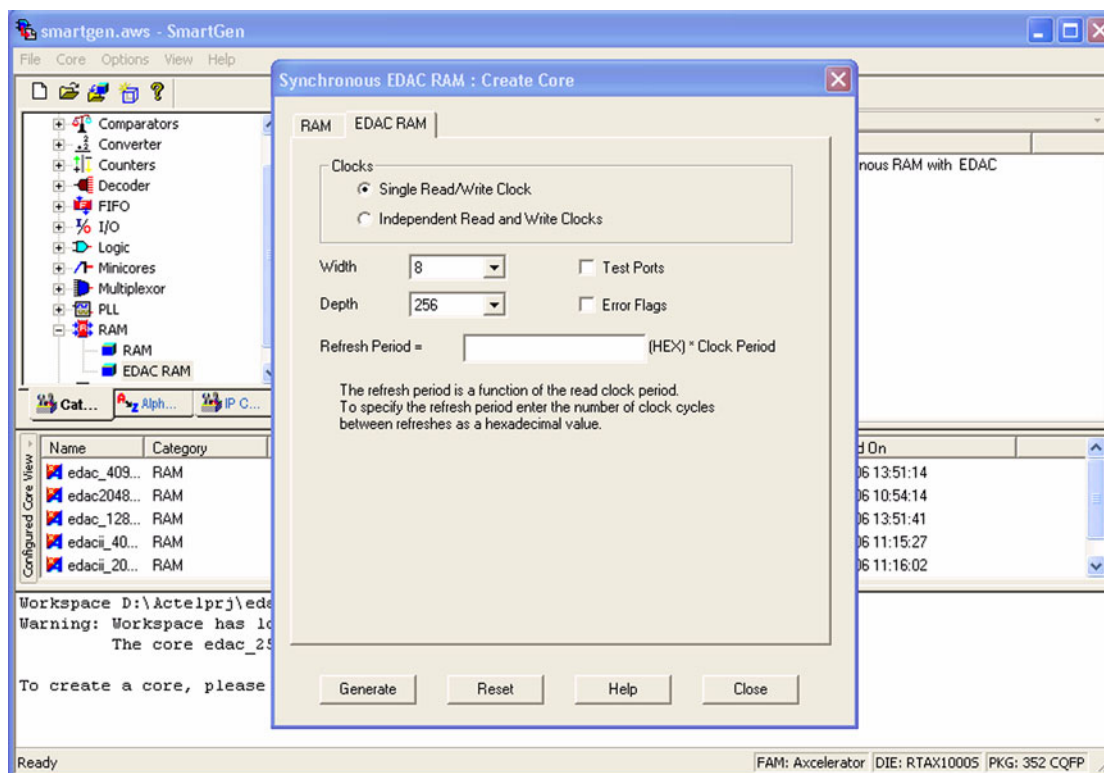


Figure 3 • SmartGen GUI for EDAC Module

Table 7 • User Definable Fields for the EDAC RAM Module

Selection	Description
Clocks	Single Read/Write Clocks Independent Read/Write Clocks (The default value is "single Read/Write" Clocks)
Width	RAM word width 8, 16 or 32; the default is 8
Depth	Data word depth 256 * N where N from 1 to 16 when Width is 8 N from 1 to 8 when Width is 16 N from 1 to 5 when Width is 32; The default value of N is 1 (Depth=256)
Test Ports	Enables or disables Test ports, requires Error flags
Error Flags	Yes or No; the default is No. If "Test Ports" is checked in (Yes), "Error Ports" will be checked automatically. "Error Flags" cannot be checked as long as "Test Ports" is checked
Refresh Period is equal to refresh rate multiplied by the clock period	Refresh rate is a hexadecimal number between 0 and 3FFFFFFF. The edit box for entering a value is limited to 11 characters. There is no default value for refresh rate

Table 8 • EDAC Core Utilization

EDAC RAM Block	Data	Depth	Utilization
Same read and write clock	8	4,096	Sequential (R-cells): 85 Combinatorial (C-cells): 303 RAM/FIFO blocks: 16 I/O with clocks: 76
	16	2,048	Sequential (R-cells): 103 Combinatorial (C-cells): 441 RAM/FIFO blocks: 16 I/O with clocks: 91
	32	1,280	Sequential (R-cells): 121 Combinatorial (C-cells): 516 RAM/FIFO blocks: 15 I/O with clocks: 123
Different read and write clock	8	4,096	Sequential (R-cells): 101 Combinatorial (C-cells): 383 RAM/FIFO blocks: 16 I/O with clocks: 81
	16	2,048	Sequential (R-cells): 119 Combinatorial (C-cells): 483 RAM/FIFO blocks: 16 I/O with clocks: 96
	32	1,280	Sequential (R-cells): 141 Combinatorial (C-cells): 614 RAM/FIFO blocks: 15 I/O with clocks: 128

Modifying the EDAC RAM Generated in SmartGen

Users can modify the VHDL/Verilog EDAC RAM netlist generated from SmartGen to add ports for more data inputs or debugging. The following examples contain code showing how to modify the VHDL netlist. Use a similar procedure to modify the Verilog netlist.

Adding More User Ports

SmartGen allows the designer to generate a word width of 8, 16, or 32 bits. However, designers can use up to 12, 29, or 47 bits of input data by modifying the netlist. Open the top-level netlist and modify the port width and component port map as follows:

```
.....
entity edacrami_top is
    port(WDATA : in std_logic_vector(11 downto 0); --modified
         --WDATA : in std_logic_vector(7 downto 0); --modified
.....
uedaci : edaci
port map(
.....
    wdata(11 downto 0)=>WDATA (11 downto 0),
    -- wdata(8)=>gnd_1_net, --modified
    -- wdata(9)=>gnd_1_net, --modified
    -- wdata(10)=>gnd_1_net, --modified
    -- wdata(11)=>gnd_1_net, --modified
```

Adding Optional Ports

SmartGen ties some EDAC ports to GND in the top level. However, the user can open the top-level netlist and use these ports. For example, users who want to bring out the RDS and TMOUT ports must modify the top-level file as shown in the following code:

```
.....
entity edacrami_top is
    port(.....
        TMOUT : out std_logic_vector (41 downto 0); --added
        RDS : in std_logic_vector (3 downto 0); --added
.....
uedaci : edaci
port map(
.....
    tmout(41 downto 0) => TMOUT (41 downto 0),
    -- tmout(0)=>gnd_1_net,
    -- tmout(1)=>gnd_1_net,
    .
    .
    .
```

```

tmout(41)=>gnd_1_net,
.
.
rds(3 downto 0) => RDS(3 downto 0),
-- rds(0)=>gnd_1_net,
-- rds(1)=>gnd_1_net,
-- rds(2)=>gnd_1_net,
-- rds(3)=>vcc_1_net,
.....

```

Compiling a VITAL VHDL Library for Full Version ModelSim®

To simulate the EDAC RAM macros targeted for the RTAX-S or Axcelerator device families, first compile the Actel VITAL VHDL library for the Axcelerator family. To complete the library:

1. Create a directory called "mti" in the "\$ALSDIR\lib\vtl\95" directory.
2. Invoke the ModelSim simulator.
3. Change the current working directory to "\$ALSDIR\lib\vtl\95\mti" directory.
4. Create an Actel family library directory. Type the following command at the prompt:
vlib Axcelerator
5. Compile the Actel VITAL VHDL library. Type the following command at the prompt:
vcom -work Axcelerator ..\Axcelerator.vhd
6. Map the Actel VITAL VHDL library to the family library directory. Type the following command at the prompt:
vmap Axcelerator \$ALSDIR\lib\vtl\95\mti\Axcelerator

Simulating the EDAC RAM Using ModelSim

Before starting simulation, modify the testbench, since this testbench is meant for testing all depths and all ports. When generating an EDAC RAM with Test Ports and Error Flags using name EDAC_8, notice the following files are generated:

- EDAC_8_top.vhd: EDAC wrapper
- edaci_18.vhd: Main EDAC
- EDAC_8.vhd: Ram block

This EDAC testbench has two extra ports (RAM depth setting and refresh rate) to check the main EDAC block. To match the testbench and EDAC wrapper, open the testbench "edactb.vhd" and modify as shown below:

```

.....
entity edactb is
end edactb;

architecture edactb_behav of edactb is
component edac_8_256_top --modified edacrami
.....
begin
<top>_0 : edac_8_256_top --modified edacrami
-- TL=42 : 2^42 * 2 * 10^-8 --/24/3600 = 1 day, period = tout*period of

```

```

--modified  --tmout      : in std_logic_vector(TL-1 downto 0);
-- RAM depth selection, --total RAM words=(rds+1)*256
--modified  --rds        : in std_logic_vector(3 downto 0);
.....
end component;
.....

uedacrami : edac_8_256_top -modified edacrami
  port map(
    --modified  -- tmout      => drvsig.tmout,
    --modified  -- rds        => drvsig.rds,
    .....

```

Then open the `edacconfig.vhd` file and modify it according to the EDAC RAM. For example, if the EDAC RAM has input data width of 8 and address of 256, change to the following parameter:

```

package edacconfig is
constant ML : integer := 8;-- user data length
constant CL : integer := 14;-- coded word length
constant SL : integer := CL-ML;-- ecc syndrome length
constant AL : integer := 8;-- address bus width
constant TL : integer := 42;-- timer length
constant CLKP : std_logic := '1';-- clock polarity
constant CLKPN : std_logic := '0';-- clock polarity NOT CLKP
end edacconfig;

```

The following steps describe how to simulate the EDAC RAM in *ModelSim*:

1. Invoke *ModelSim* simulator.
2. Change to the working directory. This directory contains your EDAC RAM and testbench and the package files. Type the following commands:
cd <working folder>
3. Create the "work" libraries. This step creates the required library directories. Type the following command at the prompt:
vlib work
vmap work ./work
4. Compile the EDAC RAM and the testbench. Type the following commands at the prompt or create a `.do` file and execute the `.do` file:
vcom -93 -work work edac_8_256.vhd
vcom -93 -work work edaci_18.vhd
vcom -93 -work work edac_8_256_top.vhd
vcom -93 -work work edacconfig.vhd
vcom -work work edactb.vhd
5. Simulate the testbench. Type the following command at the prompt:
vsim work.edactb
6. Add the waveforms for display and run the simulations.
add wave sim:/edactb/*
run-all

This performs the operations mentioned in the testbench.

Conclusion

Semiconductor memories are susceptible to errors when exposed to radiation. The use of error-correcting codes to improve semiconductor memory reliability is becoming a standard design feature. This application note presented a hardware-implemented EDAC technique for protecting memories. Users can employ Actel's SmartGen tool and generate the EDAC module. This EDAC module checks all the data that is read from memory and corrects single-bit errors. Also, it enables users to control the background scrubbing. Users can add Test Ports and use them for debugging. Actel's EDAC RAM module provides better reliability and, when possible, should be the first choice for protecting the main memory.

List of Changes

The following table lists critical changes that were made in the current version of the document.

Previous version	Changes in current version (51900041-1/7.06*)	Page
51900041-0/7.03*	The title was updated to include the subtitle.	N/A
	ACTgen was changed to SmartGen	N/A

Note: *This is the part number located on the last page of the document.

References

Hsiao, M.Y. July 1970. A Class of Optimal Minumum Odd-Weight-Column SEC-DED Codes. [online]. *IBM Journal of Research and Development*. vol.14 no. 4. Available from World Wide Web: <http://www.research.ibm.com/journal/rd/144/ibmrd1404l.pdf>

Lin, Shu and Daniel J. Costello, Jr. 1983. *Error Control Coding: Fundamentals and Applications*.(New JerseyPrentice Hall).

Appendix A

The I/O Definitions for EDAC RAM: (Single Read/Write Clock)

Table 9 • I/O Definitions for EDAC RAM

	Signal	Type	Active	Size	Description
Regular Ports	CLK	In	rising	1	Same clock for read/write
	WE	In	high	1	Write enable
	RE	In	high	1	Read enable
	WADDR	out	na	12	Write address bus
	RADDR	in	na	12	Read address bus
	WDATA	In	na	12,29,47	Write data bus
	RDATA	In	na	12,29,47	Read data bus
	RSTN	In	low	1	Asynchronous reset
	SCRUB_STOP	In	na	1	High to stop scrubbing
Test Ports	BYPASS	In	na	1	Bypass mode
	WP	In	na	6,7,7	Write ports for parity bits in bypass mode
	RP	out	na	6,7,7	Read ports for parity bits in bypass mode
Error Ports	SLOWDOWN	out	high	1	Optional flag when scrubbing cannot finish within designated period
	ERROR	out	high	1	High when two or more errors occurred during one read. Sample with read data
	CORRECTABLE	out	high	1	Low when two or more errors occurred during one read. High when one correctable error occurred. sample with read data.
	SCRUB_CORRECTED	out	high	1	High indicated scrubbing logic has corrected one error sample with the write clk
	CADDR	out	na	12	The address being corrected, sample with write clock
	SCRUB_DONE	out	high	1	High indicates scrub is done, wait for timer timeout or user can turn off scrubbing. Sample with read clk
	TMOUTFLG	out	high	1	High indicates timer is timed out
Optional Ports	Rds	In	na	4	Depth setting, ram depth=(rds+1)x256
	tmout	In	na	42	Refresh period=tmout*clk period extra

If the users select "Independent Read and Write Clocks," there will be an additional port W2R. Set W2R to the ratio of the frequency for the write clock (WCLK) over the frequency of the read clock (RCLK). Also, the port name for write clock is WCLK and for the read clock it is RCLK.

Actel and the Actel logo are registered trademarks of Actel Corporation.
All other trademarks are the property of their owners.



www.actel.com

Actel Corporation

2061 Stierlin Court
Mountain View, CA
94043-4655 USA

Phone 650.318.4200
Fax 650.318.4600

Actel Europe Ltd.

Dunlop House, Riverside Way
Camberley, Surrey GU15 3YL
United Kingdom

Phone +44 (0) 1276 401 450
Fax +44 (0) 1276 401 490

Actel Japan

www.jp.actel.com

EXOS Ebisu Bldg. 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan

Phone +81.03.3445.7671
Fax +81.03.3445.7668

Actel Hong Kong

www.actel.com.cn

Suite 2114, Two Pacific Place
88 Queensway, Admiralty
Hong Kong

Phone +852 2185 6460
Fax +852 2185 6488