

Using EDAC RAM for RadTolerant RTAX-S/SL and Axcelerator® FPGAs

Applies to EDAC Core from Libero IDE v7.2 and Newer

Introduction

The newest Actel designed-for-space field programmable gate array (FPGA) family, RTAX-S/SL, is a high-performance, high-density, antifuse-based FPGA with embedded user static RAM (SRAM). Based on the new Actel AX architecture, the RTAX-S/SL family is the high-reliability version of its commercial counterpart—the Axcelerator FPGA family. RTAX-S/SL 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 a streaming video feed. However, it is not acceptable if data is sensitive to single-bit changes, such as 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 the Actel 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 as to 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 implemented with 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, as no two valid codes can exist within d_{\min} bits of each other.

When a single-bit error is added to a code word, 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 $v = (1110011)$, 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

1. Lin and Costello, *Error Control Coding*.

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 so 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 is SEC/DED (single-error-correcting/double-error-detecting). 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

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 several 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 ones 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.

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

The RTAX-S/SL and Axcelerator FPGA families contain 36 (RTAX1000S/SL, RTAX4000S, and AX1000) or 64 (RTAX2000S/SL, AX2000) blocks of embedded memory. Each block is a 4.5 k variable-aspect-ratio, dual-port RAM. The allowable variable aspect ratios are 128×36, 256×18, 512×9, 1k×4, 2k×2, and 4k×1. 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.

2. Hsiao, "Optimal Minimum Odd-Weight-Column SEC-DED Codes."

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

Actel chose shortened Hamming codes (18,12), (36,29), and (54,47) for RTAX-S/SL 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		
User wdata/rdata width	8	16	32
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. See the "Modifying the EDAC RAM Generated in SmartGen" section on page 13 for more information. Refer to "Appendix B" on page 22 for more information on shortened Hamming codes.

Implementation of Shortened Hamming Encoding and Decoding

To implement shortened Hamming codes, you need an encoder that converts the input data to coded words. The encoding can be done by matrix multiplication of the input word by a generator matrix. The encoding process is essentially the same as the original Hamming code.

The following 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

$$v = (v_0, v_1, v_2, v_3, v_4, v_5, v_6) = (u_0, u_1, u_2, u_3) \times \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

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

$$\begin{aligned} v_6 &= u_3 \\ v_5 &= u_2 \\ v_4 &= u_1 \\ v_3 &= u_0 \\ v_2 &= u_1 + u_2 + u_3 \\ v_1 &= u_0 + u_1 + u_2 \\ v_0 &= u_0 + u_2 + u_3 \end{aligned}$$

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', the coded word is '111000100000000000'. Note that 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 with a lookup table (Table 2). For example, if the six syndrome bits are '000000', the coded bits are correct. If the syndrome bits are '100000', an error exists in 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 ones in the syndrome vector. If the syndrome contains an even number of ones, 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
0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1	0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0	0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 1 0 1 0 0	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 1 0 1 1	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 1 0 0 1 0	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 1 1 0 1	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
1 1 0 0 0 1	0 0 0 0 0 0 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 0 0 0 0 1 0 0 0 0
1 0 1 1 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 1 0 0 1 1	0 0 0 0 0 0 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 0 0 0 0 0 0 1
0 1 0 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

3. Hsiao, "Optimal Minimum Odd-Weight-Column SEC-DED Codes."



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

This parity-check matrix was derived from the method by Hsiao and was generated to result in the smallest circuit implementation. The parity-check matrix for the same shortened Hamming code (18,12) is as follows:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

In summary, during decoding, perform the following:

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

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

The following sections discuss how to generate EDAC RAM for prototyping the RTAX-S/SL FPGA family with the RTAX-S/SL or Axcelerator families, employing Actel software. The next sections also describe how you 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/SL 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/SL or 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.

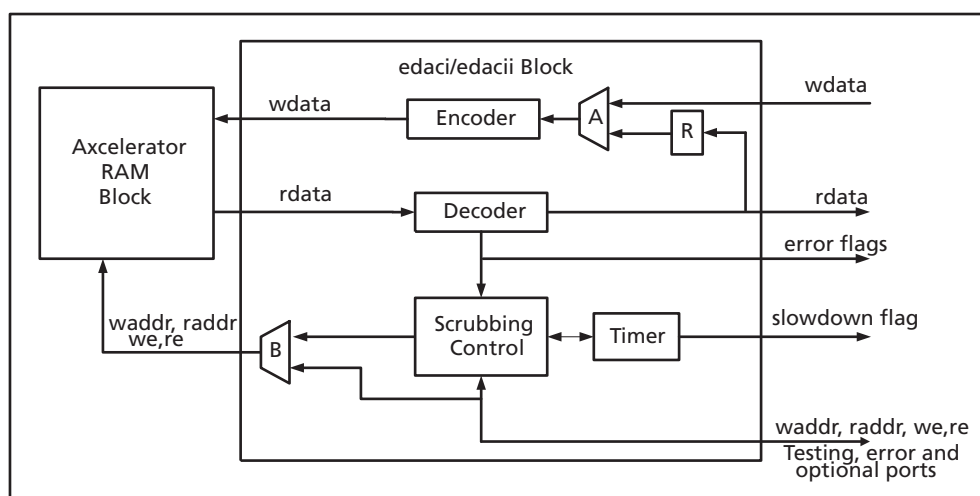


Figure 1 • Functional Diagram of EDAC RAM

During a user write, the write data is encoded by the encoder and then fed into the RTAX-S/SL 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 on page 5. During a user read, the read address and enable (raddr and re) are fed into the RAM block through 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 RTAX-S/SL and Axcelerator RAM access, but access time is increased due to the additional encoding, decoding, and scrubbing processes.

You can generate an EDAC RAM using the SmartGen tool. When you generate an EDAC RAM from SmartGen, the tool generates a top-level wrapper for RAM encoding, an RTAX-S/SL or Axcelerator RAM block, and an EDAC module. Figure 2 shows the EDAC module and RAM block inside the EDAC wrapper. SmartGen allows you to generate two clock mode types for the EDAC RAM: single clock mode and independent read and write clock mode. For single clock mode, the same clock will be used for read and write, and the user will see one clock port called CLK. For independent read and write clock mode, the user will see RCLK and WCLK ports. Refer to "Appendix A" on page 20 for the EDAC RAM port definitions.

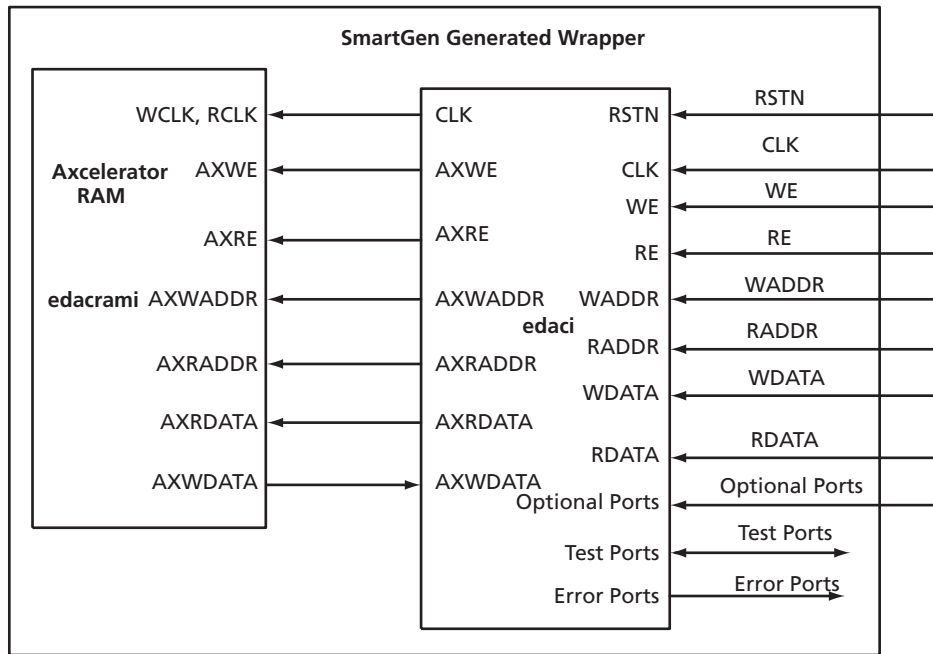


Figure 2 • EDAC Module in EDAC RAM

Features of EDAC RAM from SmartGen

The SmartGen core generator can generate an EDAC module with various configurations. The main features of the EDAC module are as follows:

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

Scrubbing Control Block

The scrubbing control block handles access to the memory background scrubbing process. During scrubbing, 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/SL 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/SL or Axcelerator RAM.

When STOP_SCRUB is set HIGH, the scrubbing stops. Note that for independent read and write clock mode, when STOP_SCRUB is turned off and the scrubber restarts, the scrubber picks up from the address where it was stopped. However, for single clock mode, the scrubber starts from address zero when it restarts. The output signal SCRUB_DONE indicates that the scrubbing logic has gone through all RAM addresses. You 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 indicates the address of the corrected error. [Table 3](#) describes the scrubbing signals. Timing diagrams of scrubber and read/write activity are shown in [Figure 4 on page 17](#) through [Figure 9 on page 18](#).

You need to be careful about RAM access when scrubbing is on. "[Scrubbing with EDAC RAM Using Independent Read and Write Clocks](#)" goes over the recommended usage. You should not perform reads from or writes to the RAM while the scrubber is active in either single or independent read and write clock mode.

Table 3 • Scrubbing Signals

Signal	Description
STOP_SCRUB	LOW to turn on scrubbing; HIGH to turn off scrubbing
SCRUB_DONE	Transition from LOW to HIGH means scrubbing is done.
SLOWDOWN	HIGH indicates scrubbing logic has corrected one error; sample with the write clock.
SCRUB_CORRECTED	HIGH indicates scrubbing logic has corrected one error.
CADDR	The address being corrected; sample with write clock.
TMOUTFLG	HIGH indicates timer is timed out.

Scrubbing with EDAC RAM Using Independent Read and Write Clocks

The independent read and write clock EDAC RAM has some additional restrictions beyond those for the single-clocked core.

- Scrubbing must be stopped before user accesses can take place (i.e., no background scrubbing is allowed).
- After STOP_SCRUB is asserted, you must wait for two WCLK and two RCLK clock edges before asserting WE or RE.
- When STOP_SCRUB is asserted (assuming BYPASS stays inactive), the current scrub address is not reset. When STOP_SCRUB is deasserted, scrubbing will continue at the next address, allowing the entire memory to be scrubbed. On the single-clock version of the core, whenever STOP_SCRUB is asserted, the scrub address is reset to zero.
- If BYPASS is asserted when STOP_SCRUB is asserted, the scrub address will be reset. To have access to the BYPASS input, the **Test Ports** option must be set within SmartGen; otherwise, BYPASS is permanently inactive.

Refresh Rate

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

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

where *tmout* is the refresh timer's timeout value (set in the SmartGen GUI; see the "EDAC RAM Generation Using SmartGen" section on page 9).

When the read address counter returns to zero (the condition for asserting the STOP_SCRUB signal), the scrubbing process will stop until it receives a timeout from the EDAC timer. This implementation helps the background scrubbing process save power because needless refresh cycles are eliminated. When a timer timeout is indicated (TMOUTFLG is asserted HIGH; see Table 4) and the read address counter has NOT returned to zero, 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). Note that if STOP_SCRUB is asserted when SLOWDOWN has already been asserted, SLOWDOWN will remain asserted until STOP_SCRUB is deasserted. The *tmout* setting should be larger than the RAM depth to ensure that a complete refresh cycle can be completed within the refresh period. See Figure 8 on page 18 and Figure 9 on page 18 for timing diagrams of the SLOWDOWN and TMOUTFLG flags.

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/SL 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/SL or Axcelerator RAM block. During this time, the scrubbing process is stopped. 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	8	16	32
EDAC module WDATA/RDATA	18	36	54
EDAC module WP/RP	6	7	7

Error Flags

The EDAC RAM has two error flags, CORRECTABLE and ERROR, which allow you to monitor error correction and detection (Table 6 and Figure 4 on page 17 through Figure 7 on page 17).

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.

EDAC RAM Generation Using SmartGen

The SmartGen tool enables you to generate the EDAC RAM. You can generate an EDAC RAM with either a single read/write clock or independent read and write clocks. [Figure 3](#) shows the SmartGen GUI for EDAC RAM generation, and [Table 7](#) lists the user-definable fields. The **Test Ports** and **Error Flags** check boxes allow you to generate additional ports. You cannot generate a Test Port without selecting **Error Flags**. Use the generated netlist in the design to go through the complete design flow.

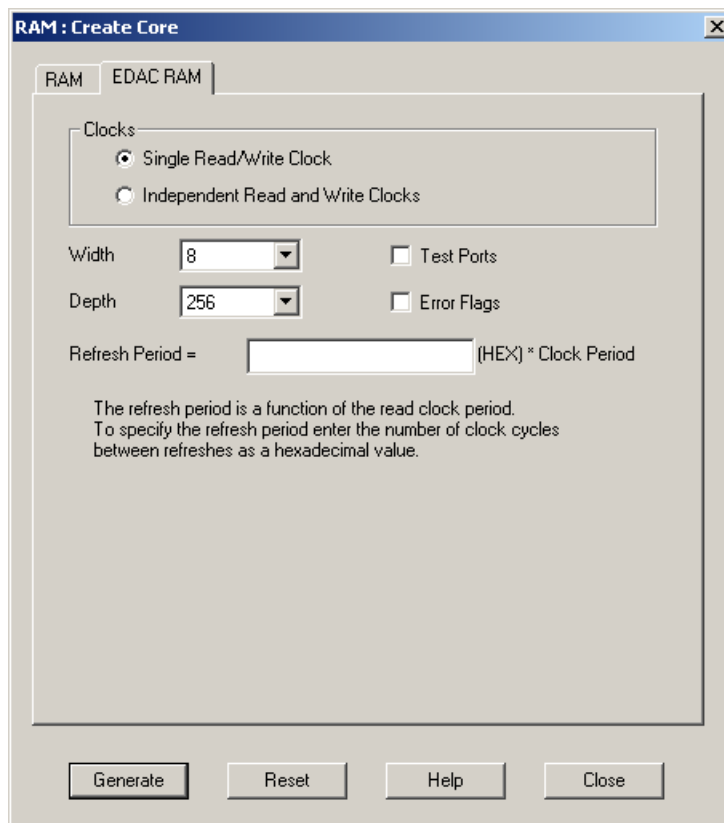


Figure 3 • SmartGen GUI for EDAC Module

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 <ul style="list-style-type: none"> N is from 1 to 16 when Width is 8 N is from 1 to 8 when Width is 16 N is 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; default is No. If Test Ports is selected (Yes), Error Flags will be selected automatically. Error Flags cannot be selected as long as Test Ports is selected.
Refresh Period	Refresh Period is equal to the refresh rate multiplied by the clock period. Refresh rate is a hexadecimal number between 0 and 3FFFFFFF. The data entry box for the value is limited to 11 characters. There is no default value for refresh rate.

Limitations of the SmartGen GUI

The SmartGen GUI has the following limitations:

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

To change the input word width or add more Test Ports, you need to modify the SmartGen HDL file as shown in the "Adding More User Ports" section on page 13.

Detecting Simultaneous User Reads and Writes

The RTAX-S/SL and Axcelerator RAM does not support simultaneous read and write accesses to the same address. If this should occur, the write will take place correctly, but the read data may contain one of the following:

1. The old data
2. The new data
3. A mixture of old and new data

Cases (1) and (2) do not create any issues; however, in case (3), the data value contains a mixture of the old and new data and will result in erroneous data that the EDAC will fail to correct. You should ensure, through the system design, that simultaneous user reads and writes to the same address do not occur and do not access the EDAC RAM while the scrubber is on. You can include the following RTL code in your design to detect this condition.

Single Clock Example

```
-- synthesis translate_off
process(rstn,clk)
begin
  if rstn = '0' then
    samerw <= '0';
  elsif clk'event and clk = CLKP then
    samerw <= '0';
    if re = '1' and we = '1' and raddr = waddr then
      samerw <= '1';
      assert FALSE
      report "EDAC Detected Simultaneous User Read and Write Cycle.
        See EDAC Application Note."
      severity WARNING;
    end if;
  end if;
end process;
--synthesis translate_on
```

Independent Clock Example

```
--synthesis translate_off
process(rstn,wclk)
begin
  if rstn = '0' then
    samerw_w <= '0';
  elsif wclk'event and wclk = CLKP then
    samerw_w <= '0';
    if re = '1' and we = '1' and raddr = waddr then
      samerw_w <= '1';
      assert FALSE
      report "EDAC Detected Simultaneous User Read and Write Cycle.
        See EDAC Application Note."
      severity WARNING;
    end if;
  end if;
end process;

process(rstn,rclk)
begin
  if rstn = '0' then
    samerw_r <= '0';
  elsif rclk'event and rclk = CLKP then
    samerw_r <= '0';
    if re = '1' and we = '1' and raddr = waddr then
      samerw_r <= '1';
      assert FALSE
      report "EDAC Detected Simultaneous User Read and Write Cycle.
        See EDAC Application Note."
      severity WARNING;
    end if;
  end if;
end process;

samerw <= samerw_r or samerw_w;
--synthesis translate_on
```

This code can be used either for simulation only or to create a flag detecting the condition. For use just in simulation, the "synthesis translate_off" and "synthesis translate_on" directives prevent synthesis. The code will generate a simulation warning message should simultaneous accesses occur. Alternatively, the code can be included in your design and the samerw signal used to indicate a potential system failure. It is not recommended that the independent clock logic example be synthesized, since the comparison logic is not protected from metastable conditions.

Core Utilization of EDAC RAM

You can generate the EDAC RAM from SmartGen and instantiate it in your code. However, you should be cautious about the core utilization and ensure that it has enough space to fit the EDAC RAM. [Table 8](#) describes the utilization for three configurations of the EDAC RAM. Note that the utilization table was generated using Actel Libero® Integrated Design Environment (IDE) v7.2.

Table 8 • EDAC Core Utilization

EDAC RAM Block	Data	Depth	Utilization
Same read and write clock	8	4,096	Sequential (R-cells): 81 Combinatorial (C-cells): 308 RAM/FIFO blocks: 16 I/O with clocks: 76
	16	2,048	Sequential (R-cells): 99 Combinatorial (C-cells): 397 RAM/FIFO blocks: 16 I/O with clocks: 91
	32	1,280	Sequential (R-cells): 117 Combinatorial (C-cells): 528 RAM/FIFO blocks: 15 I/O with clocks: 123
Independent read and write clocks	8	4,096	Sequential (R-cells): 103 Combinatorial (C-cells): 290 RAM/FIFO blocks: 16 I/O with clocks: 77
	16	2,048	Sequential (R-cells): 124 Combinatorial (C-cells): 386 RAM/FIFO blocks: 16 I/O with clocks: 92
	32	1,280	Sequential (R-cells): 146 Combinatorial (C-cells): 510 RAM/FIFO blocks: 15 I/O with clocks: 124

Modifying the EDAC RAM Generated in SmartGen

You 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 you to generate a word width of 8, 16, or 32 bits. However, you 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 at the top level. However, you can open the top-level netlist and use these ports. For example, to bring out the RDS and TMOUT ports, you must modify the top-level file as shown in the following code:

```
.....
entity edacrami_top is
    port(.....
          TMOUT : in 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,
.....

```

Simulating EDAC Using Libero IDE v7.2 and v7.3

To simulate the single-clock EDAC RAM macros targeted for the RTAX-S/SL or Axcelerator device family, follow the steps below:

1. Invoke Libero IDE.
2. Create a new project with following settings:
HDL type = VHDL, Family = Axcelerator, Die = RTAX1000S/SL, Package = 352 CQFP
3. Once the project is created, invoke SmartGen in the Libero IDE Design Flow window.
4. Invoke **EDAC RAM** from Core Varieties in the Axcelerator Family window and enter the following:
 - Single read/write clock
 - Depth: 256
 - Width: 8 bits
 - Refresh period: 3FFFFFFFFF cycles
 - Select the **Test Ports** and **Error Flags** check boxes
5. Generate the EDAC RAM with core name "edac_256_8". The following files are generated:
 - *edac_256_8_top.vhd*: EDAC wrapper
 - *edac_256_8.vhd*: RAM block
 - *edaci_18.vhd*: EDAC core
6. Import the package *edacconfig.vhd* as a stimulus file and modify as follows:

```

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;

```

7. Import the testbench *edactb.vhd* as a stimulus file and modify as follows:

```

.....
entity testbench is --modified edactb
end testbench; --modified edactb
architecture edactb_behav of testbench is

```

```

component edac_256_8_top --modified edacrami
.....
begin
--modified --tmout : in std_logic_vector(TL-1 downto 0); -- TL=42 : 2^42*2*10^-8
--/24/3600 = 1 day, period = tout*period of
--modified --rds : in std_logic_vector(3 downto 0); -- RAM depth selection,
--total RAM words=(rds+1)*256
.....

end component;

.....

uedacrami : edac_256_8_top --modified edacrami
port map(
-- tmout => drvsig.tmout, --modified
-- rds => drvsig.rds, --modified
.....

```

8. Open **Option > Project Settings Simulation** and change the following:
Top-level instance name in the testbench: uedacrami
Simulation run time: 2 ms
9. Right-click the top level and create Post-Synthesis Simulation.
10. When prompted to organize the stimulus file, add both *edacconfig.vhd* and *edactb.vhd* to associated files.
Libero IDE will launch ModelSim® and run simulation for 2 ms.

Simulating EDAC Using Libero IDE v8.0

To simulate the single-clock EDAC RAM macros targeted for the RTAX-S/SL or Axcelerator device family, follow the steps below:

1. Invoke Libero IDE.
2. Create a new project with following settings:
HDL type = VHDL, Family = Axcelerator, Die = RTAX1000S/SL, Package = 352 CQFP
3. Once the project is created, expand the SmartGen Cores List in the Catalog window
4. Invoke **EDAC RAM** and enter the following:
 - Single read/write clock
 - Depth: 256
 - Width: 8 bits
 - Refresh period: 3FFFFFFFFF cycles
 - Select the **Test Ports** and **Error Flags** check boxes
5. Generate the EDAC RAM with core name "edac_256_8". Select **Show Modules** under Design Explorer to see the following files that are generated:
 - *edac_256_8.vhd*: EDAC wrapper

- *edac_256_8_RAM.vhd*: RAM block
- *edaci_18.vhd*: EDAC core

6. Import the package *edacconfig.vhd* as a stimulus file and modify as follows:

```
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;
```

7. Import the testbench file *edactb.vhd* as a stimulus file and modify as follows.

```
.....
entity testbench is --modified edactb
end testbench; --modified edactb
architecture edactb_behav of testbench is
component edac_256_8 --modified edacrami
.....
begin
--modified --tmout : in std_logic_vector(TL-1 downto 0); -- TL=42 : 2^42*2*10^-8
--/24/3600 = 1 day, period = tout*period of
--modified --rds : in std_logic_vector(3 downto 0); -- RAM depth selection,
--total RAM words=(rds+1)*256
.....
end component;
.....
uedacrami : edac_256_8 --modified edacrami
port map(
-- tmout => drvsig.tmout, --modified
-- rds => drvsig.rds, --modified
.....
```

8. Open **Project > Project Settings Simulation** and change the following:

Top-level instance name in the testbench: uedacrami
Simulation run time: 2 ms

9. Right-click the top level and create Pre-Synthesis Simulation.
10. When prompted to organize stimulus file, add both *edacconfig.vhd* and *edactb.vhd* to associated files.

Libero IDE will launch ModelSim and run simulation for 2 ms.

Note that the file name for the EDAC wrapper generated by SmartGen has been changed in Libero IDE v8.0. If you regenerated the EDAC macro in Libero IDE v8.0 for an existing design, make the necessary changes to match the file name.

Timing Waveforms

Figure 4 through Figure 9 on page 18 show various EDAC activities.

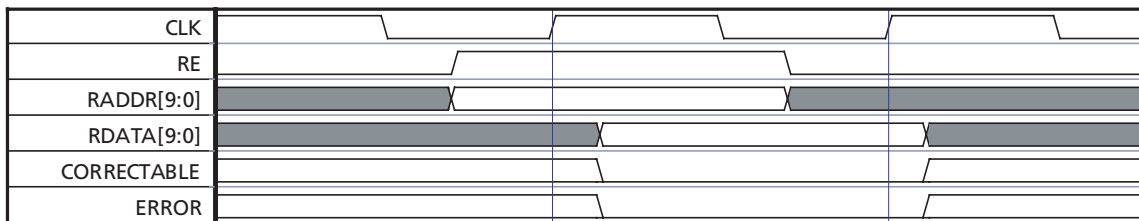


Figure 4 • Normal Read Cycle

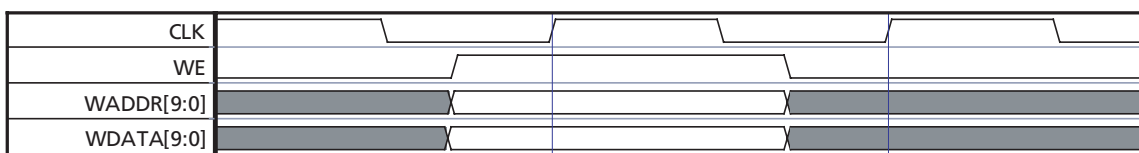


Figure 5 • Normal Write Cycle

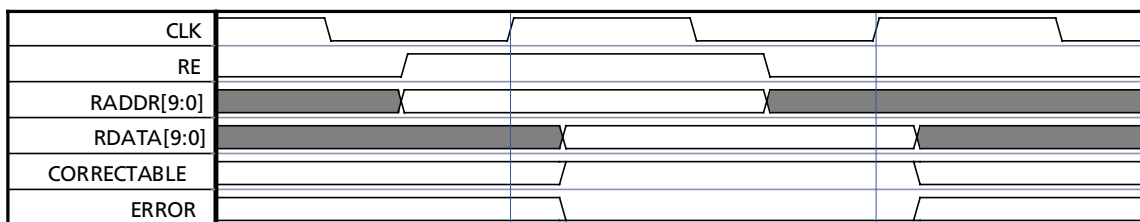


Figure 6 • Read Cycle with Correctable Error

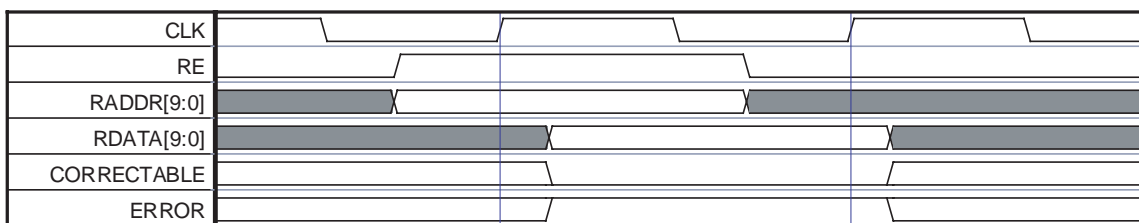


Figure 7 • Read Cycle with Uncorrectable Error

Figure 8 shows a scrub cycle that completes within the timeout period. When scrub activity completes at 2.2 μ s, the EDAC core asserts SCRUB_DONE and then waits for the internal timer to timeout at 3.3 μ s. At this point, TMOUTFLG is asserted and the scrub cycle is repeated.

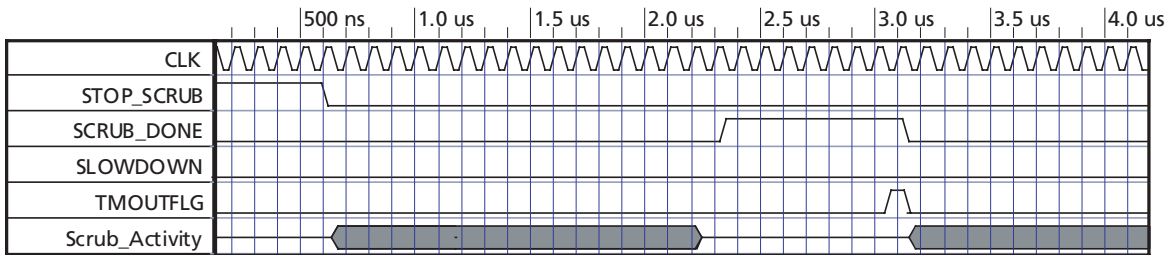


Figure 8 • Scrub Cycle that Completes within Timeout Period

Figure 9 shows a scrub cycle that fails to complete within the timeout period, as not enough spare memory cycles are available to scrub all RAM locations. In this case, when the timeout occurs at 3.3 μ s, the EDAC core asserts SLOWDOWN and continues scrubbing until the top of memory is reached, at which point it asserts SCRUB_DONE and the TMOUTFLG, indicating that the scrub cycle is complete. In this case, if possible, the user access rate to the memory should be decreased to allow the scrubbing to complete within the desired time.

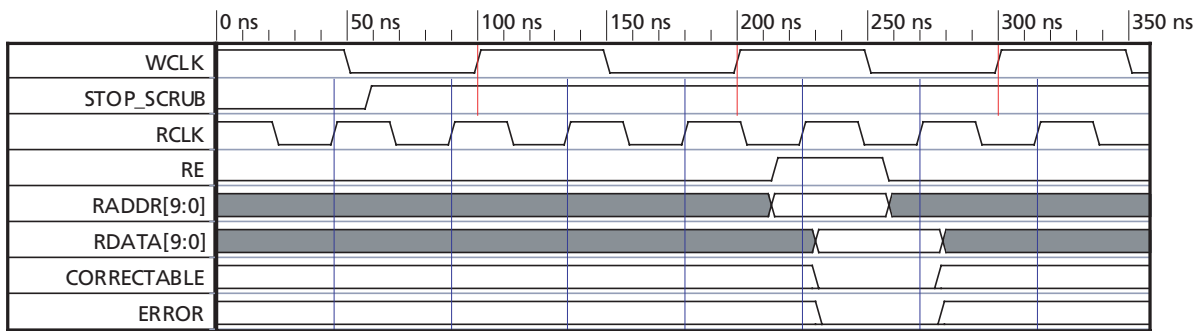


Figure 9 • STOP_SCRUB Assertion Prior to Memory Access (independent clocked EDAC)

Figure 9 shows STOP_SCRUB being asserted for two RCLK and WCLK clock edges before RE is asserted. At 60 ns, STOP_SCRUB is asserted; the two RCLK edges occur at 90 and 135 ns, but the read access must wait for the two WCLK edges to occur, at 100 and 200 ns. This means that RE must be asserted after 200 ns in this case. The external read enable RE is now valid on the RCLK edge occurring at 225 ns, with valid output data being provided just after the clock edge.

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 the Actel SmartGen tool and generate the EDAC module, which checks all the data read from memory and corrects single-bit errors. Also, the EDAC module enables users to control the background scrubbing. Further, users can add Test Ports for debugging. The Actel 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 (51900145-2/2.08*)	Page
51900145-1/5.07	2n and 2k were changed to 2 ⁿ and 2 ^k in the "Coding Theory Background" section.	1
51900145-0/6.06	Figure 3 was updated.	9
	In the "Adding Optional Ports" section, the following statement was changed from: TMOUT: out std_logic_vector (41 downto 0); --added to TMOUT: in std_logic_vector (41 downto 0); --added	13
	Software version numbers were added to the title: "Simulating EDAC Using Libero IDE v7.2 and v7.3" section. The text was corrected in instruction 4 to the following: Depth: 256 Width: 8 bits The text was corrected in instruction 5 to the following: edac_256_8.vhd: RAM block edaci_18.vhd: EDAC core Instruction 10 is new.	14
	The "Simulating EDAC Using Libero IDE v8.0" section is new.	15
	Figure 6 was updated.	17

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

Appendix A

Table 9 • I/O Definitions for EDAC RAM

Port Name	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	N/A	12	Write address bus
	RADDR	In	N/A	12	Read address bus
	WDATA	In	N/A	12,29,47	Write data bus
	RDATA	In	N/A	12,29,47	Read data bus
	RSTN	In	LOW	1	Asynchronous reset
STOP_SCRUB	In	N/A	1	HIGH to stop scrubbing	
Test Ports	BYPASS	In	N/A	1	Bypass mode
	WP	In	N/A	6,7,7	Write ports for parity bits in bypass mode
	RP	Out	N/A	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 indicates scrubbing logic has corrected one error. Sample with write clock.
	CADDR	Out	N/A	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 clock.
	TMOUTFLG	Out	HIGH	1	HIGH indicates timer is timed out.
Optional Ports	RDS	In	N/A	4	Depth setting; RAM depth = (RDS + 1) × 256
	TMOUT	In	N/A	42	Refresh period = TMOUT × clk period extra

Table 10 • I/O Definitions for EDAC RAM with Independent Read and Write Clocks

Port Name	Signal	Type	Active	Size	Description
Regular Ports	RCLK	In	Rising	1	Read clock
	WCLK	In	Rising	1	Write clock
	WE	In	HIGH	1	Write enable
	RE	In	HIGH	1	Read enable
	WADDR	Out	N/A	12	Write address bus
	RADDR	In	N/A	12	Read address bus
	WDATA	In	N/A	12,29,47	Write data bus
	RDATA	In	N/A	12,29,47	Read data bus
	RSTN	In	LOW	1	Asynchronous reset
	SCRUB_STOP	In	N/A	1	HIGH to stop scrubbing
Test Ports	BYPASS	In	N/A	1	Bypass mode
	WP	In	N/A	6,7,7	Write ports for parity bits in bypass mode
	RP	Out	N/A	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 indicates scrubbing logic has corrected one error. Sample with write clock.
	CADDR	Out	N/A	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 clock.
	TMOUTFLG	Out	HIGH	1	HIGH indicates timer is timed out.
Optional Ports	RDS	In	N/A	4	Depth setting; RAM depth = (RDS + 1) × 256
	TMOUT	In	N/A	42	Refresh period = TMOUT × clk period extra

Notes:

1. WE, WADDR, WADATA, and WP are synchronous to WCLK.
2. RE, RADDR, RDATA, and RP are synchronous to RCLK.
3. STOP_SCRUB and BYPASS are double-sampled internally by both RCLK and WCLK to avoid any timing issues. However, the signal pulse widths must be greater than both clock periods to ensure that they are correctly sampled.

The W2R port is not used with the version of the EDAC core provided with Libero IDE v7.2 and later. The port should be tied LOW or HIGH rather than left disconnected.

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.

River Court, Meadows Business Park
Station Approach, Blackwater
Camberley Surrey GU17 9AB
United Kingdom

Phone +44 (0) 1276 609 300
Fax +44 (0) 1276 607 540

Actel Japan

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

Phone +81.03.3445.7671
Fax +81.03.3445.7668
www.jp.actel.com

Actel Hong Kong

Room 2107, China Resources Building
26 Harbour Road
Wanchai, Hong Kong

Phone +852 2185 6460
Fax +852 2185 6488
www.actel.com.cn

