# Simulating SEU Events in EDAC RAM

## Introduction

The Actel RTAX-S Field Programmable Gate Array (FPGA) provides embedded user static RAM in addition to single-event-upset (SEU)-enhanced logic, including embedded triple-module redundancy (TMR) registers. The embedded user SRAM coupled with the error detection and correction (EDAC) core, included in the Actel SmartGen core generator, provides mitigation of soft errors within the SRAM blocks for data-critical applications. The Actel Axcelerator® family of FPGAs is the commercially-available version, and does not contain the SEU-enhanced logic. It can be used in high-reliability applications such as avionics, where the use of EDAC to mitigate SEU events in user SRAM may be required.

The Actel EDAC core uses a shortened Hamming code and is capable of detecting up to two errors or automatically correcting a single error within a single address location. This EDAC core, which includes a background scrubber that can be used to continually read and correct any single-bit errors found, provides a very reliable method for mitigating SEU effects in the embedded SRAMs, resulting in error rates better than $10^{-10}$ errors/bit-day. For more information on implementing the EDAC core in Actel RTAX-S or Axcelerator FPGAs, refer to the *Using EDAC RAM for RadTolerant RTAX-S FPGAs and Axcelerator FPGAs* application note.

Although the Actel EDAC core provides mitigation and/or notification of SEU-induced errors within the embedded SRAMs, the ability to test the user control logic associated with an SEU-induced error is a bit problematic, as the user must induce an error that simulates a cosmic event. This application note shows how to accomplish such a feat by using the capabilities of the Model*Sim*® AE simulator included in the Actel Libero® Integrated Design Environment (IDE) tool suite.

## EDAC Background

The integrity requirement of the data contained within the embedded SRAM is determined by the particular application and may range from noncritical (such as the as pixel data for an image) to very critical (such as a header for a communication packet or a data word for a control application). The designer will need to determine whether the EDAC core is required in the application. If required, the EDAC core must be configured and the remainder of the design must react to the notification of SEU events by the EDAC core.

The Actel EDAC core within the Actel SmartGen core generator is configurable for various memory sizes and has the ability to add optional Error Flags and Test Ports. Table 1 on page 2 gives descriptions for the Error Flags and Test Ports.

Note that the ability to change the "detect 2 / correct 1" capability of the EDAC core is not user-configurable, nor is the ability to remove the background scrubber feature. The optional Test Ports are used to directly access the EDAC SRAM locations and change the coded parity bits directly via the wp and rp ports. To induce an SEU event using these ports, however, would require calculating the syndrome bits for a given data word. For an 8-bit data word, this requires padding the data word to 12 bits and doing matrix multiplication of this word with the generator matrix for the shortened Hamming code (18, 12), which is found in the *Using EDAC RAM for RadTolerant RTAX-S FPGAs and Axcelerator FPGAs* application note. Then the syndrome bits must be changed so that on the next read of that location, when the 18-bit encoded data word is presented to the EDAC decoder, either a CORRECTABLE Flag and the corrected data is put out from the EDAC block or an Error Flag and the errant data is put out from the EDAC block.

Additionally, adding Test Ports to the EDAC SRAMs and modifying the stimulus may add excessive complexity to the user design and testbench in order to verify correct functionality of the user control logic when an SEU event occurs in the embedded user SRAM.

*Table 1* • **Error Flags and Test Ports**

| Name | Type | Polarity | Bit Width | Description |
|---|---|---|---|---|
| **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 the 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. |

# Creating SEU Events in EDAC SRAMs

Depending upon the selected configuration of the EDAC core in a design, a few possibilities may need to be tested:

- EDAC memories with no Error Flags or Test Ports
- EDAC memories with Error Flags
- EDAC memories with Error Flags and Test Ports

For the purpose of this application note, the case of EDAC memories with Error Flags will be used in the example design. Comments will be made to distinguish what would be done in the other cases where applicable. When simulating EDAC errors in the above three cases, the following is generally true:

- EDAC memories with no Error Flags or Test Ports
  SEU events can be simulated, but observability is limited to verifying that the read data in the simulation is correct for one induced SEU event and is incorrect for two induced SEU events.
- EDAC memories with only Error Flags
  SEU events can be simulated. In addition to observing the correct data for one induced SEU event, the corrected signal will indicate that the EDAC core found and corrected an error in the SRAM data. In the case of two induced SEU events, not only will the read data be incorrect, but the ERROR signal will also indicate that the EDAC core found an uncorrectable error.
- EDAC memories with Error Flags and Test Ports
  As with the case above, SEU events can be simulated and observed at the data word as well as the Error or corrected signals. Additionally, the Test Ports are available if the user wishes to add simulation vectors to manipulate the parity bits of the coded word.

# General Description

SEU events are typically initiated by cosmic events that cause heavy ions from galactic rays to collide with the silicon lattice of a RAM cell. If the heavy ions have sufficient photovoltaic energy, they can produce a state change in a stored bit, resulting in errors in the stored data. These unpredictable events have been proven capable of affecting state-of-the-art geometry devices, even in terrestrial applications. If the stored data in the SRAMs is sensitive to SEU events, mitigation techniques such as the Actel EDAC SRAM blocks should be employed.

The Actel SmartGen macro generator generates EDAC SRAM blocks for the Axcelerator and RTAX-S FPGA memories with various configurations such as width, depth, error flags, and test ports. The EDAC SRAM contains an EDACI (EDAC core) block and an SRAM block instantiated within a top-level wrapper. SRAM data is written to and read from the SRAMs in the same fashion as with the non-EDAC SRAMs; however, the EDAC block manages implementation of the shortened Hamming codes—(18, 12), (36, 29), or (54, 47), depending upon the memory data width (8-, 16-, or 32-bits)—and the background scrubber.

Although the Actel EDAC SRAM block with its detect 2 / correct 1 error capability and the background scrubber should be sufficient to mitigate most SEU events within the SRAM data, the designer may want to be notified about an SEU event and have the system take subsequent action, depending upon whether the EDAC block was able to correct the errors or not.

The Actel Libero IDE tool suite includes the Actel Edition of the Mentor Graphics® Model*Sim* HDL simulator. One of the capabilities included in the Model*Sim* simulator is the ability to force a value onto a node within the design until a new event triggers a new value to be driven onto that same node. This application note describes how to use this Model*Sim* feature to force a node within the EDAC SRAM block to simulate an SEU event during the writing of the data to the actual SRAM. This error will be detected either during a read from the EDAC SRAM or during the background scrubbing operation if it is enabled. The same technique is used to generate either a single SEU event or multiple SEU events in a single memory location. These would then generate a CORRECTABLE Flag or ERROR Flag, respectively, out of the EDAC RAM block. The portion of the user circuitry dedicated to these flags is verified in simulation.

# EDAC SRAM Design

Figure 1 shows a simplified circuit including the EDAC SRAM block. This simple circuit is sufficient to demonstrate the ability to simulate SEU events using the Model*Sim* simulator. The Libero IDE project containing this design is available from the Actel website for those who would like to have a working example to test with.

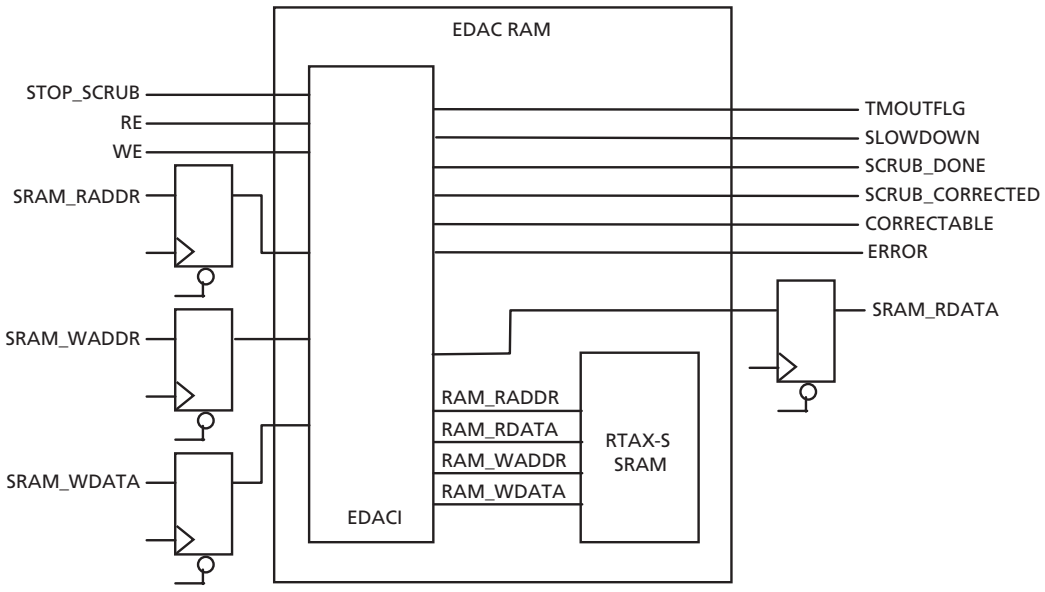


*Figure 1* • **Axcelerator EDAC SRAM Example Design**

The EDAC block was created with the Actel SmartGen tool. The EDAC SRAM block was configured with the options shown in Figure 2.

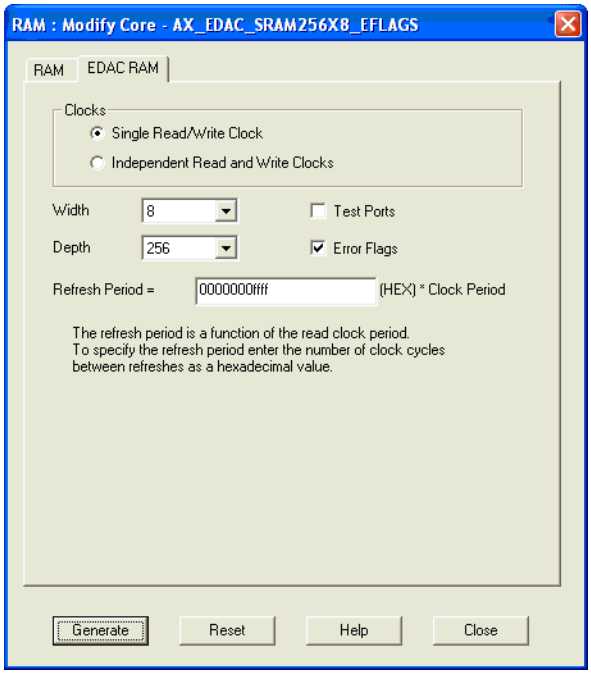


*Figure 2* • **SmartGen EDAC SRAM Block**

Notice that the Error Flags box is checked and the Test Ports box is not. Despite the fact that the Test Ports provide access to the coded parity bits, they are not needed for this method of implementing SEU events in the design. The SRAM configuration was chosen to be 256 locations deep by 8 bits wide, which, with the additional EDAC coded bits and coded parity bits, will still fit in a single RTAX-S memory block. Remember, for each of the allowed bit width configurations a different shortened Hamming code is used. In this method for generating SEU events, the user need not be concerned with the different Hamming codes used for each width configuration. The EDAC core generated by the SmartGen macro generator is instantiated into a top-level block with additional logic to register the SRAM write address, SRAM write data, SRAM read address, and SRAM read data. The stimulus is generated providing a reset and clocks. This disables the background scrubbing circuitry and writes data to the first 8 locations. This stimulus then reads those locations back. A snapshot of the simulation is shown in Figure 3 on page 5. There are small glitches on the CORRECTABLE Flag and the ERROR Flag; however, they are too small to be captured.

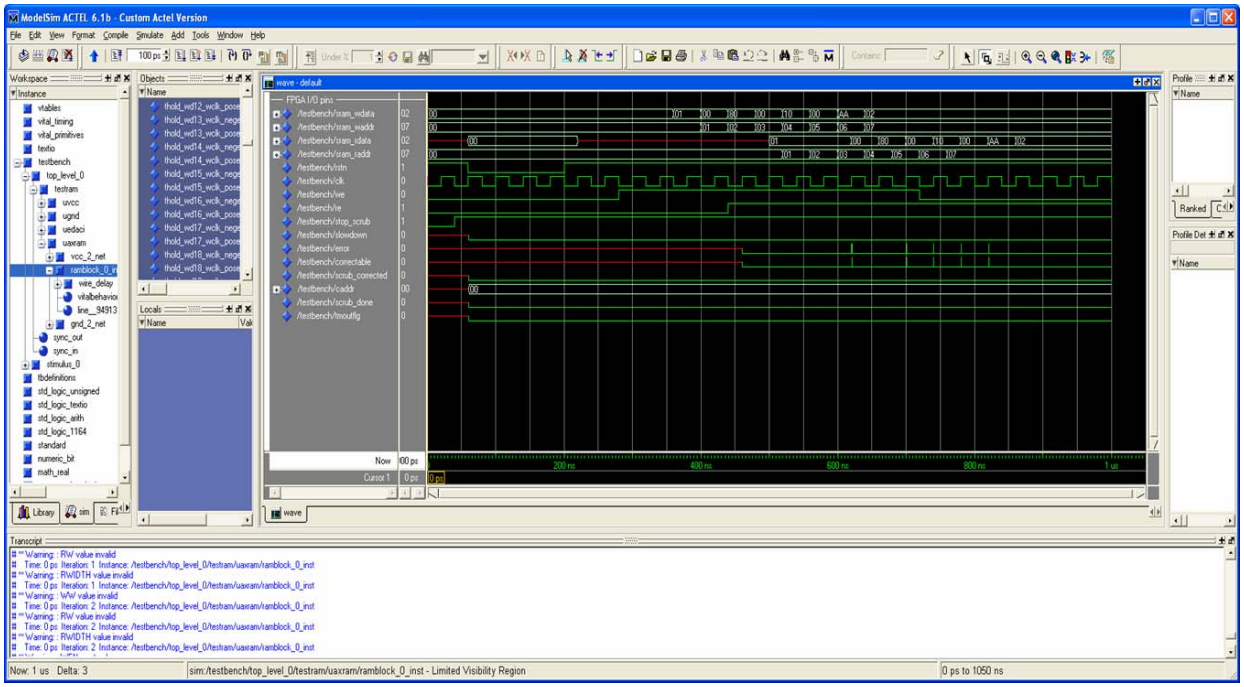

*Figure 3* • **EDAC SRAM Write/Read Operations**

Two items are noteworthy in this simulation. First, the STOP_SCRUB line is set active (HIGH), which stops the operation of the background scrubber. This allows the user to verify the operation of the user-designed circuitry in response to the EDAC core without the background scrubber running. The operation of the background scrubber is tested separately, either after the SRAM locations have been initialized or using the SRAM initialization for simulation. Also in this particular case, the SRAM is initialized for simulation with all zeroes, using the *meminit.dat* file to initialize the contents of the SRAM for simulation. This initialization is done for the Axcelerator and RTAX-S SRAMs including those configurations with the EDAC core surrounding the embedded SRAM. In simulation, this will prevent unknown SRAM contents from creating unknown signal values until the SRAM is loaded with known values. The idea behind this is that initializing the contents of the SRAM can save considerable simulation time that would normally be used to initialize these SRAMs before using them. Though this initialization is done in simulation, the Actel Axcelerator and RTAX-S embedded SRAMs do not have a global reset or preset that can drive the contents of the SRAMs to a known state. In actual silicon, the SRAM contents should be considered unknown until the user has initialized them. If the background scrubber is enabled and begins scrubbing the contents of the SRAM prior to the user initialization of the EDAC SRAM block, the scrubber circuitry could very well generate SCRUB_CORRECTED or ERROR Flags in response to random data in the SRAMs prior to initializing those SRAMs.

## Implementing SEU Events in an EDAC SRAM Design Simulation

To avoid the extra work involved in trying to calculate the syndrome bits and using the Test Ports to implement SEU events in the user's simulation, we will use the capabilities of the Model*Sim* simulator to change the state of a data line after the EDAC circuitry, just before it is written to the SRAM block in the Actel device. This ensures that the data in the SRAM now holds a value that simulates an SEU in which a single bit within the SRAM has been changed. This is accomplished by adding the signals that connect directly to the Actel SRAM block after the EDAC circuitry in the simulation. The user will need to work through the hierarchy of the instantiated EDAC blocks in the design until the Actel SRAM primitive is found instantiated in the EDAC RAM block. In this case, ram64K36 is the Actel SRAM primitive used. Add the signals of interest (write address, wrad*x,* and write data, wd*x*) to the simulation. Note that once the signals in the object window are selected, you can use **View** > **Filter** and clear the check box for **Internal Signals** to remove the internal signals from the object window. Run the simulation past the point where some or all data is written to the SRAMs.

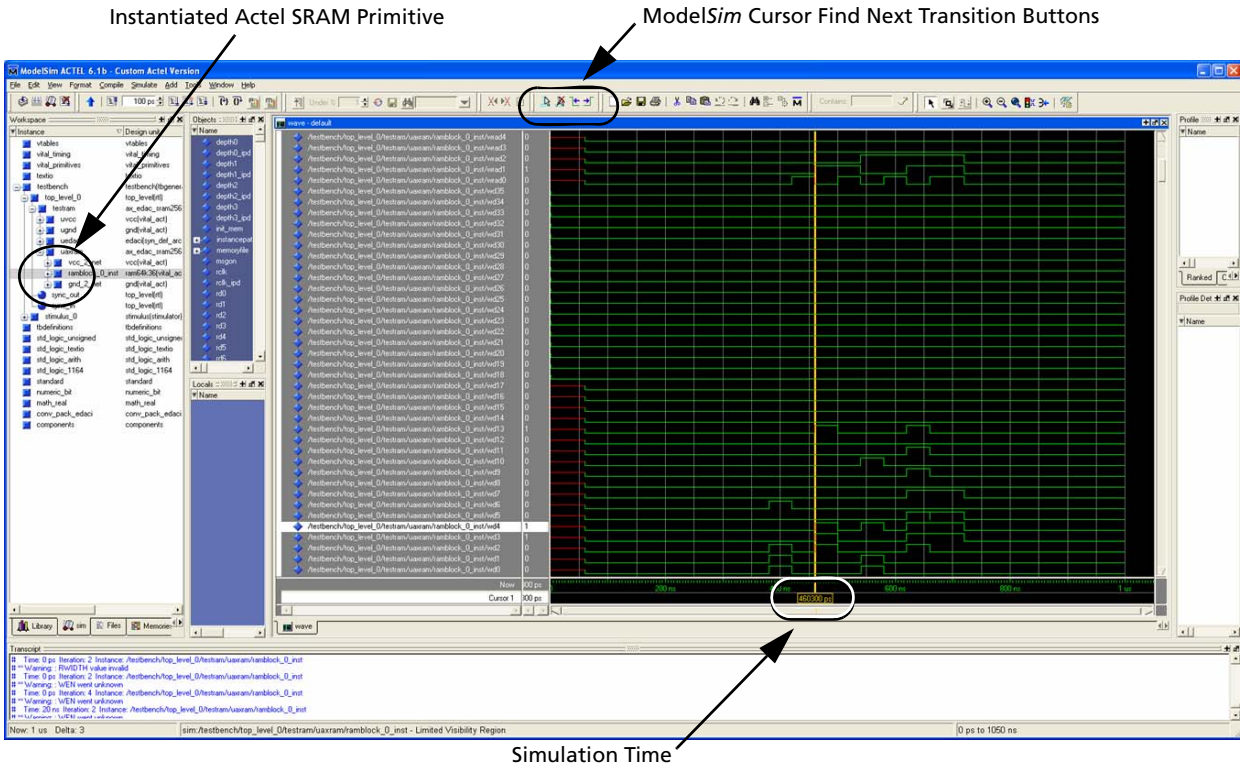


*Figure 4* • **EDAC SRAM Write Operations**

Figure 4 shows that the write address, wrad*x,* at the Actel SRAM primitive block is consistent with the write address applied to the EDAC module. This is useful for keeping track of the location in the SRAM where the SEU will be located. Note that the write data has been increased to 18 bits, which are used by the EDAC circuitry to include the extra data bits as well as the Syndrome bits for implementing the (18, 12) shortened Hamming code for an 8-bit data width, as described in the *Using EDAC RAM for RadTolerant RTAX-S FPGAs and Axcelerator FPGAs* application note.

The next step is to add a cursor to the Simulation Results window and determine the time in the simulation where one or more SRAM write data lines will be manipulated. To add a cursor in the Model*Sim* Wave window, select the wave window and choose **Add** > **Cursor**. Once the cursor has been added to the waveform window, select the write data signal that transitions during write operation and use the Find Next Transition buttons to accurately position the cursor on the transition and read the simulation time at the bottom of the cursor.

The next operation is to restart the simulation; this is done by clicking the **Restart** button on the Model*Sim* toolbar and accepting the defaults in the Restart dialog box. Now the simulation is ready to run again. In the Model*Sim* Transcript window, use the **Run** command to run the simulation to the time found earlier, where the write data signal would normally begin to transition. It is critical that the simulation be run to the exact point of the write data transition. If the simulation is stopped too early, even by 0.3 ns, the force –deposit command, which forces a signal to the designated value and is valid only from the time forced until the next transition, is applied too early and the transition we are attempting to modify will occur as normal.

In this example, the following command was issued in the Model*Sim* transcript window:

```
run 460.3 ns
```

Now the simulation is at the point where we can issue the **force -deposit** command to suppress the normally high transition on the write data wd4 signal using the following command in the Model*Sim* transcript window. Notice that the force -deposit command requires the full hierarchical path to the signal, which is found in the Model*Sim* wave window. The value of 0 is applied to this signal at the current simulation time.

```
force –deposit /testbench/top_level_0/testram/uaxram/ramblock_0_inst/wd4 0
```

The force -deposit command can also be run using the GUI. Select **wd4** in the object window, right-click, and select **Force**. The Force Selected Signal window appears. Use **Value** = **0** and **Kind** = **Deposit**, and click **OK**, as shown in Figure 5.

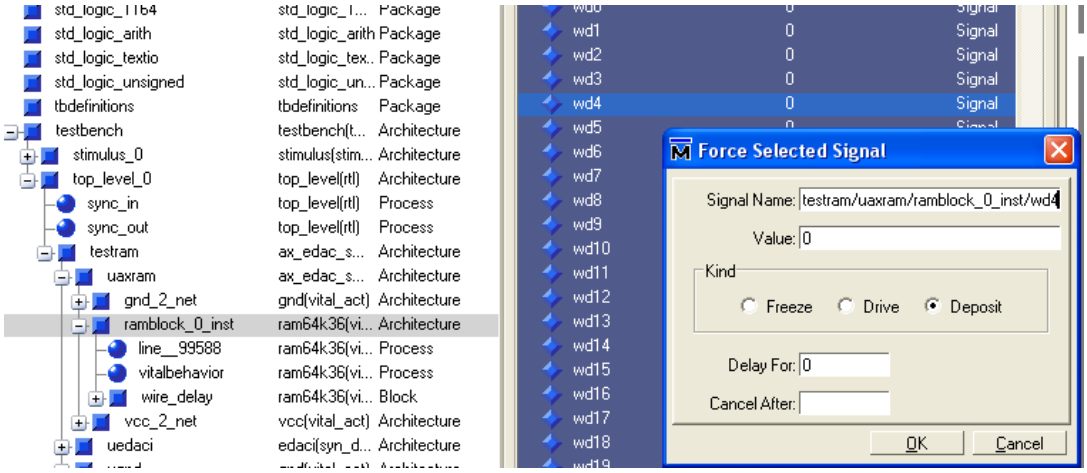


*Figure 5* • **Force Selected Signal**

The Run command can now be issued to run the simulation past the point where data is read from this SRAM location.

```
run 540 ns
```

Now the simulation appears as shown in Figure 6.

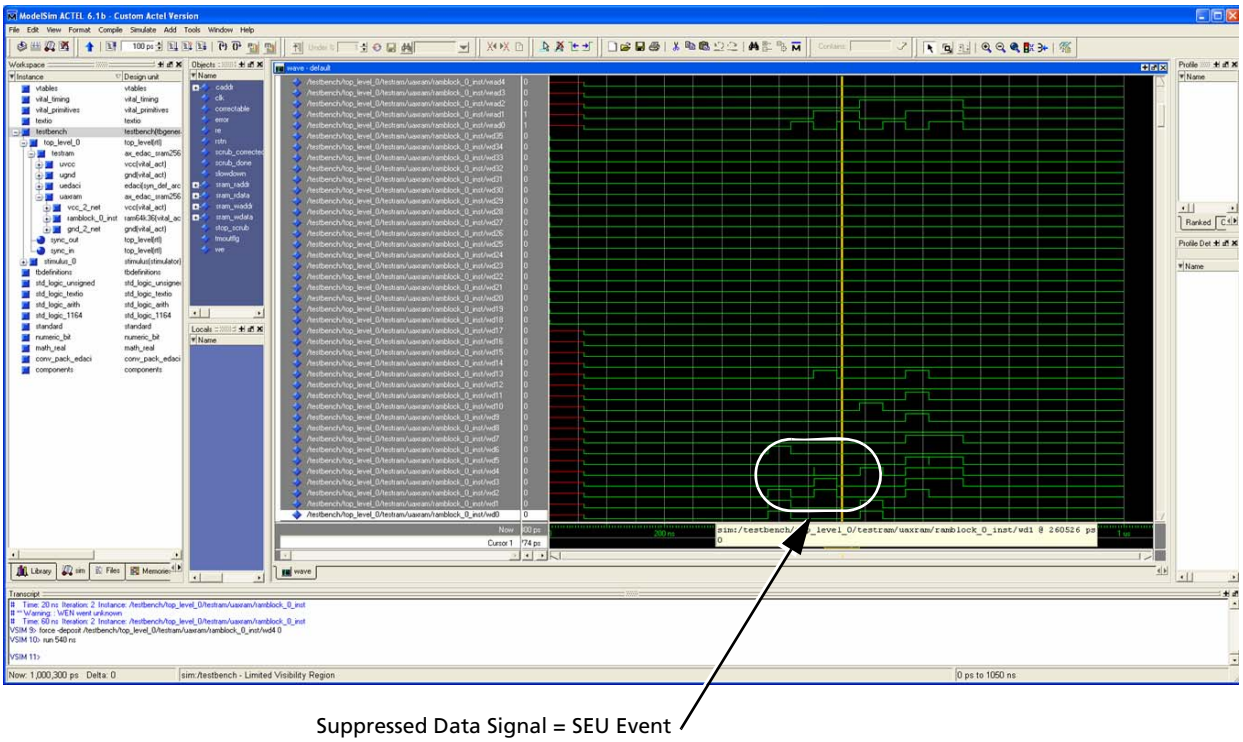


*Figure 6* • **EDAC SRAM Simulation with SEU**

In comparing Figure 6 with Figure 4 on page 6, note that the active high data bit on write data wd4 has been forced to a LOW data bit for the one write cycle at the 460.3 ns point in the simulation. For reference, look at the write address lines. We have now effectively generated an SEU event at address 0x02 by causing the data written into the SRAM to be one bit different from what the EDAC core expects to be in this location. Also note that the force -deposit command is only active until the next point where write data wd4 would normally transition; in this case, only for the single write cycle we are interested in. Although the simulation shows write data wd4 LOW for two write cycles, the force -deposit is only active for the single write cycle that the wd4 signal should have been HIGH. The second cycle is the valid 0 level, shown in Figure 4 on page 6 for the write to write address location 0x03.

## Validating SEU Events in an EDAC SRAM Design Simulation

In the simulation shown in Figure 7, the EDAC core indicates a CORRECTABLE Flag at the point in the simulation where the data is read back from the EDAC SRAM at address 0x02. If the read address signals have been added to the simulation, the CORRECTABLE Flag occurs at the end of the 0x02 read cycle and the correct data is transmitted from the EDAC core to the SRAM read data port. Remember that the actual signals shown at the Testbench level of the simulation include a user register delay from the actual SRAM block read data; however, the correct data, 0x80, which was written to address 0x02, was read back from

address 0x02, indicating that the EDAC core is functioning correctly and any user logic associated with reacting or logging a CORRECTABLE Flag is validated as well.

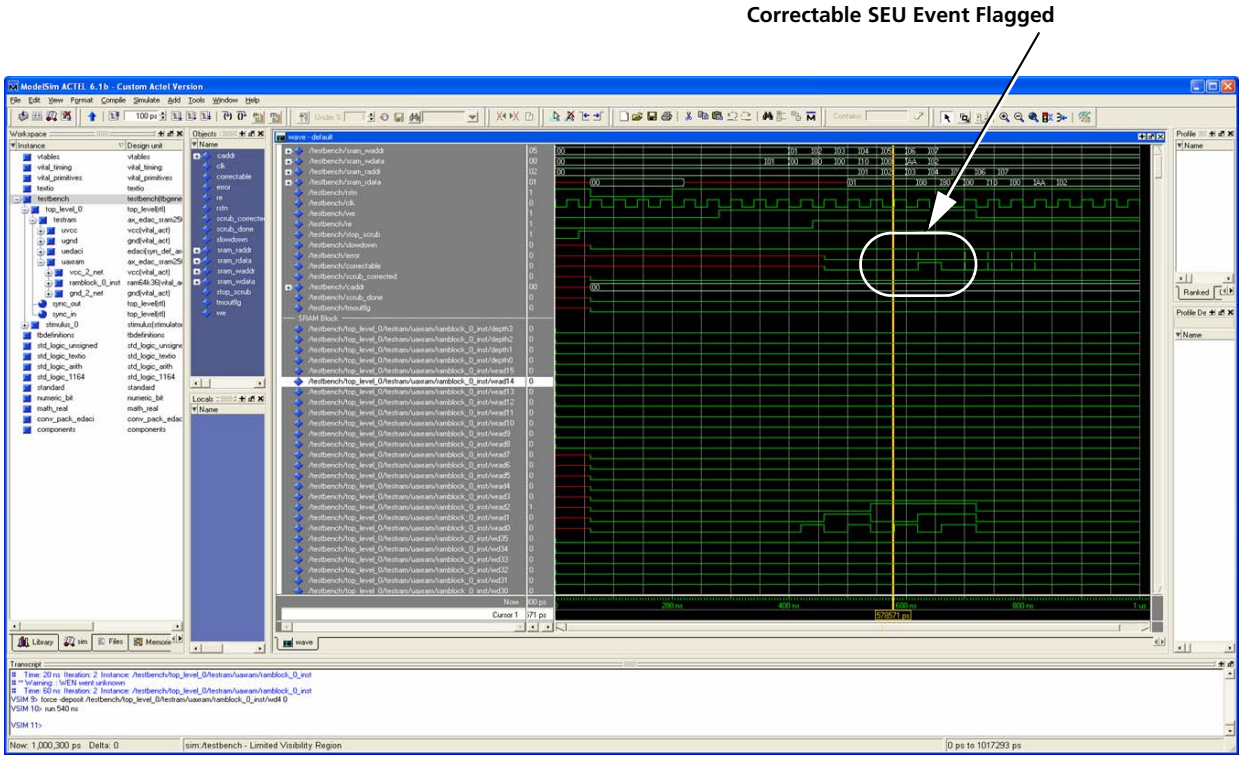


*Figure 7* • **EDAC SRAM Simulation with Correctable SEU Event Flagged**

The same methodology is used to generate the equivalent of multiple-bit errors so an Error Flag can be generated. This would be done with exactly the same method; however, rather than using the force -deposit command to change the value on a single write data bit, it can be used to change the value on two write data lines. Now the operation of the Error Flag and appropriate logic can be validated as well.

In the example shown in Figure 7, click **Restart** on the Model*Sim* toolbar, accept the default settings in the Model*Sim* Restart popup window, and run the simulation to 460.3 ns. The two commands shown below will be issued:

```
force -deposit /testbench/top_level_0/testram/uaxram/ramblock_0_inst/wd4 0
force -deposit /testbench/top_level_0/testram/uaxram/ramblock_0_inst/wd3 0
```

The results are shown in Figure 8 and Figure 9.

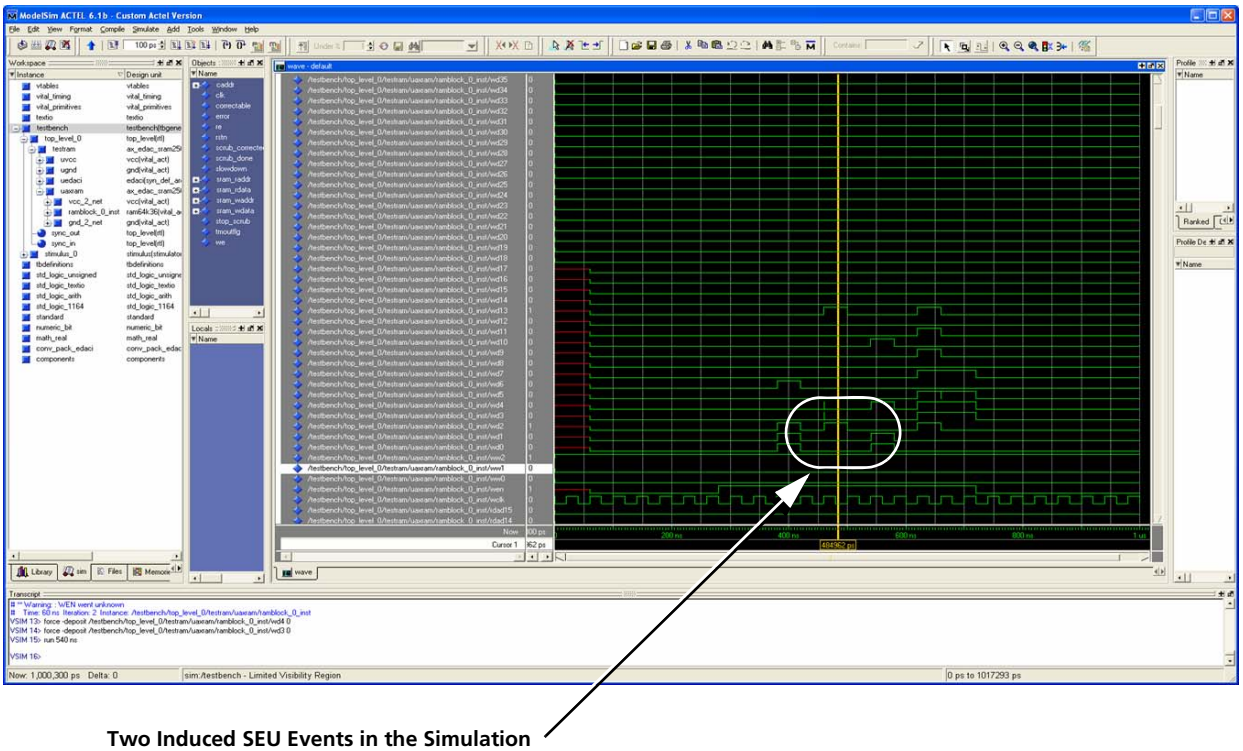


*Figure 8* • **EDAC SRAM Simulation with Two Induced SEU Events**

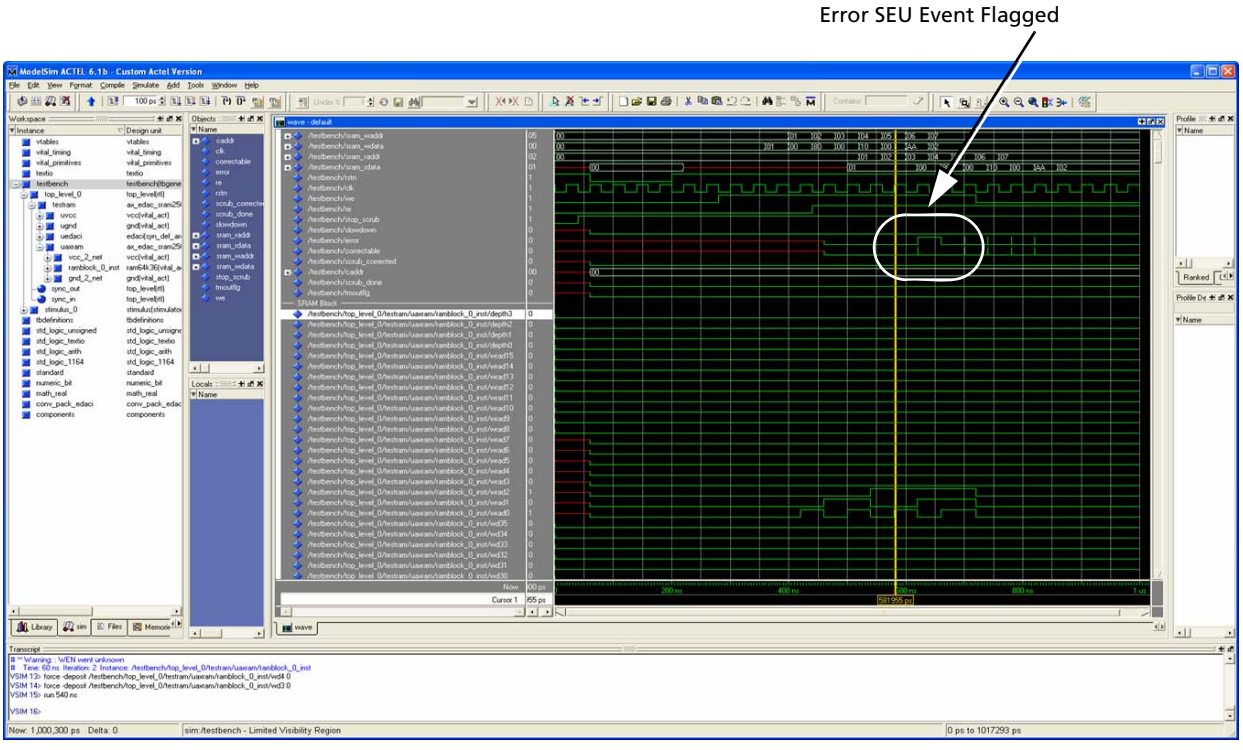


*Figure 9* • **EDAC SRAM Simulation with ERROR SEU Event Flagged**

In summary, the force -deposit command supported by the Model*Sim* simulator is utilized to simulate SEU events by modifying the write data to the EDAC SRAM block after the EDAC circuitry but prior to the actual SRAM write. The read data appears to have undergone an SEU event in the SRAM when read back from the EDAC core. The appropriate EDAC core flag is generated, depending upon the number of SEU events the EDAC core sees in the data word. Validation of the user's logic to these SEU flags is then completed. The advantage is that the user does not need to understand and predict the proper data to write into the EDAC SRAMs via the Test Ports, nor is a separate routine in the testbench required to validate the operation of the system when one of the EDAC core flags appears.

The operation of the EDAC background scrubber is also validated using this method of inducing single- or multiple-SEU events in the SRAM. As shown in Figure 10, the testbench has been changed to suppress the SRAM reads but to enable the STOP_SCRUB signal after the completion of the data writes to the SRAM core. Figure 10 shows STOP_SCRUB being enabled halfway through the seven write cycles. The RE signal is inactive and the read address is held at 0x00. Once the STOP_SCRUB signal to the EDAC core is deactivated, the background scrubber within the EDAC core begins supplying read addresses and validating the data from the SRAM within the EDAC core. In this case, the data read from the SRAM by the EDAC background scrubber is clocked by the registers in the top-level design and brought out on the read data ports.

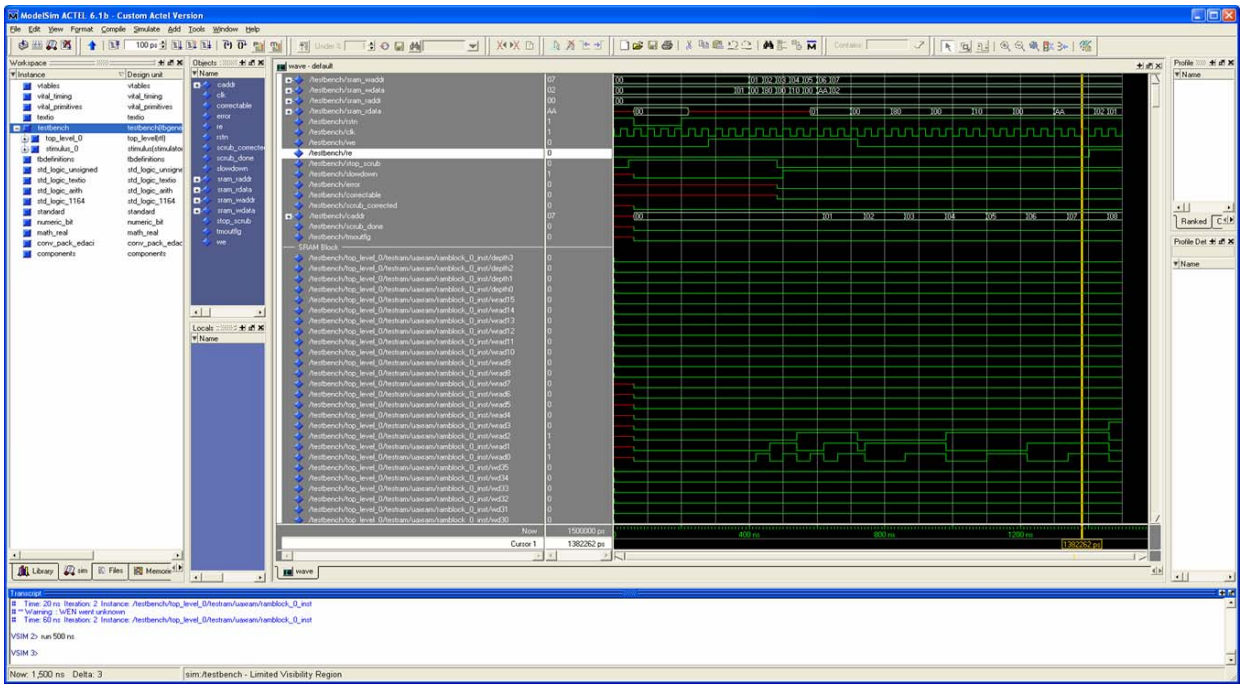

*Figure 10* • **EDAC Background Scrubber Working**

The same methodology is used to invoke single or multiple SEU events in the SRAM simulation using the force -deposit command in Model*Sim*. For consistency, the same data lines in the same memory location are used to cause the same SEU event in the SRAM that was flagged by the EDAC core as a correctable error when reading that SRAM location. In this case, the background scrubber within the EDAC core is tested to find and correct this SEU error on its own. Figure 11 on page 12 shows the results after applying the force –deposit command on the wd4 line at 460.3 ns exactly, as done in Figure 10, and then running the simulation to a point where the background scrubber encounters the errant data in the SRAM location. At this point, the background scrubber issues a SCRUB_CORRECTED signal to indicate that the error was found and corrected. Notice that the timing is delayed to accommodate the EDAC core's rewriting the corrected data back to the SRAM location. This refreshed data is now error-free and reduces the possibility of that memory location experiencing another SEU event, which becomes an error when

two SEU events are found in the same location and the EDAC core can no longer automatically correct the data based on the shortened Hamming code used.

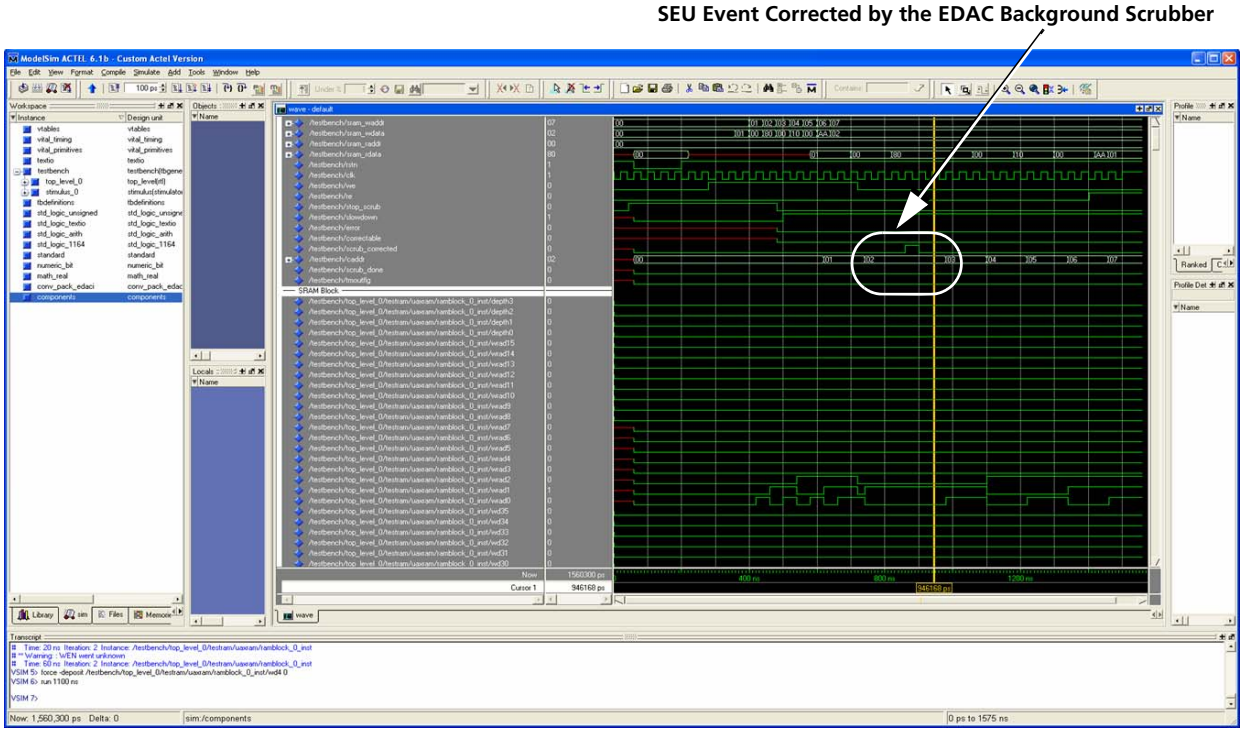


*Figure 11* • **EDAC Background Scrubber Finding and Correcting a Single Bit Error**

## Using SRAM Initialization Files with EDAC SRAM Designs

Actel SRAM blocks are not initialized at power-up in either the Axcelerator or RTAX-S families of FPGAs. It is imperative to initialize the SRAM blocks as part of the system initialization and to be aware of the potential issues that can occur, especially with an EDAC core and a running background scrubber prior to loading the SRAMs. All of the above simulation has used the memory initialization capability of the SRAM simulation models to call out an initialization file containing memory values that are loaded into the simulation model for the SRAMs. The benefit of this is that long simulations are not required to load up the SRAM blocks before system functionality is observed. However, if these models are used, a simulation should be run to verify the loading of the SRAMs and the operation of the EDAC circuitry, including the background scrubber during the loading of the SRAMs. Careful consideration given to the configuration of the EDAC circuitry during initialization of the SRAMs could prevent errant system operation prior to or during the loading of the SRAMs. Consider the simulation in Figure 12 on page 13, which shows the EDAC circuitry with the background scrubber enabled and an SRAM initialization file with some random bits in it, possibly representing a random data word in the SRAM block after power-up but prior to initialization. In this case, the EDAC background scrubber issues a SCRUB_CORRECTED message before the system starts initializing the SRAMs. This could cause the system to react incorrectly based on irrelevant data in the SRAMs, likely a residual effect of power-up. This is avoided by disabling the background scrubber until the initialization of the SRAMs is complete. This should be considered carefully in controlling the system operation at power-up.

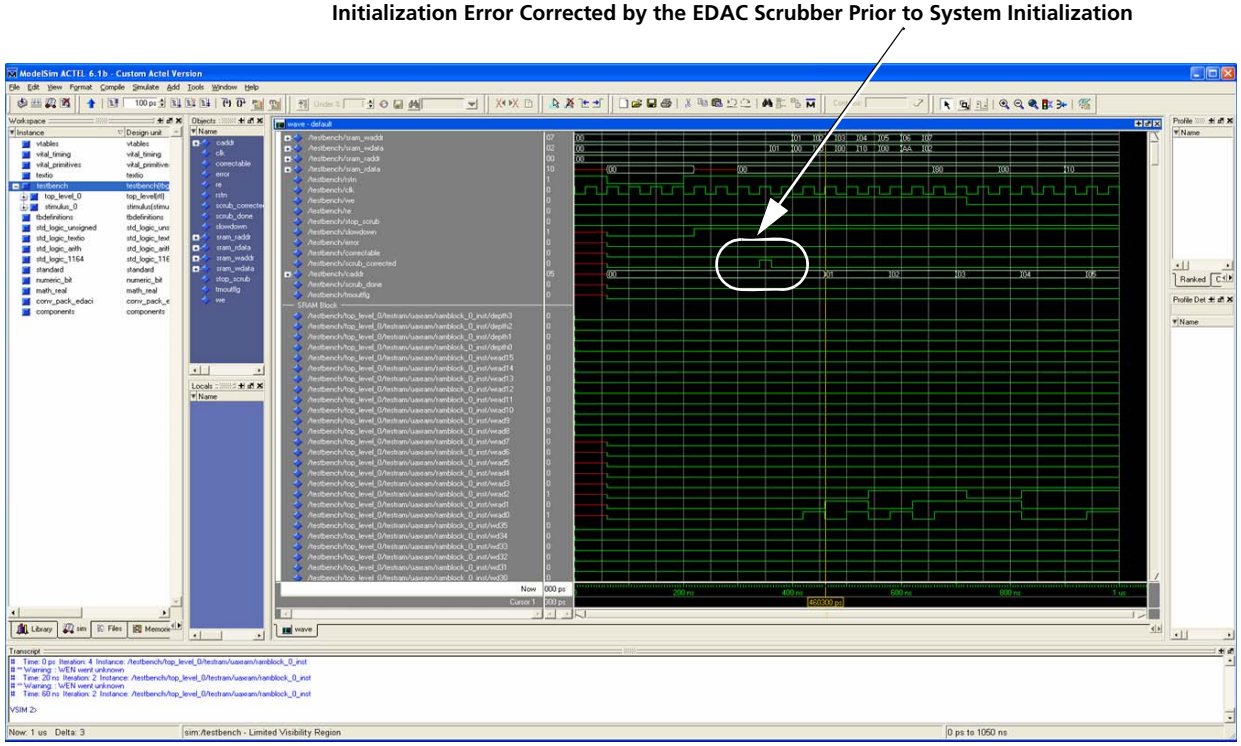


*Figure 12* • **EDAC Background Scrubber Finding an Error Prior to Initialization**

The EDAC SRAM core allows depths in increments of 256 only, which may be larger than the application requires; however, the background scrubber is assigned to scrub the full depth of the EDAC SRAM block. In some cases, the application may only be using a portion of the SRAM block, such as 64 or 128 locations. Careful consideration should also be given to properly initialize the unused portion of the SRAM block so the background scrubber does not generate flags when it moves beyond the user data in the SRAM block. Consider a simulation (Figure 13 on page 14) in which the memory initialization file contains a single bit in the last memory location. As the EDAC scrubber runs the full depth of the EDAC SRAM block, it encounters what it considers to be an SEU event in the last location prior to beginning again. This might not have been an SEU event, but an SRAM location that was not properly initialized because the user only expected to use a certain number of locations in the EDAC SRAM.

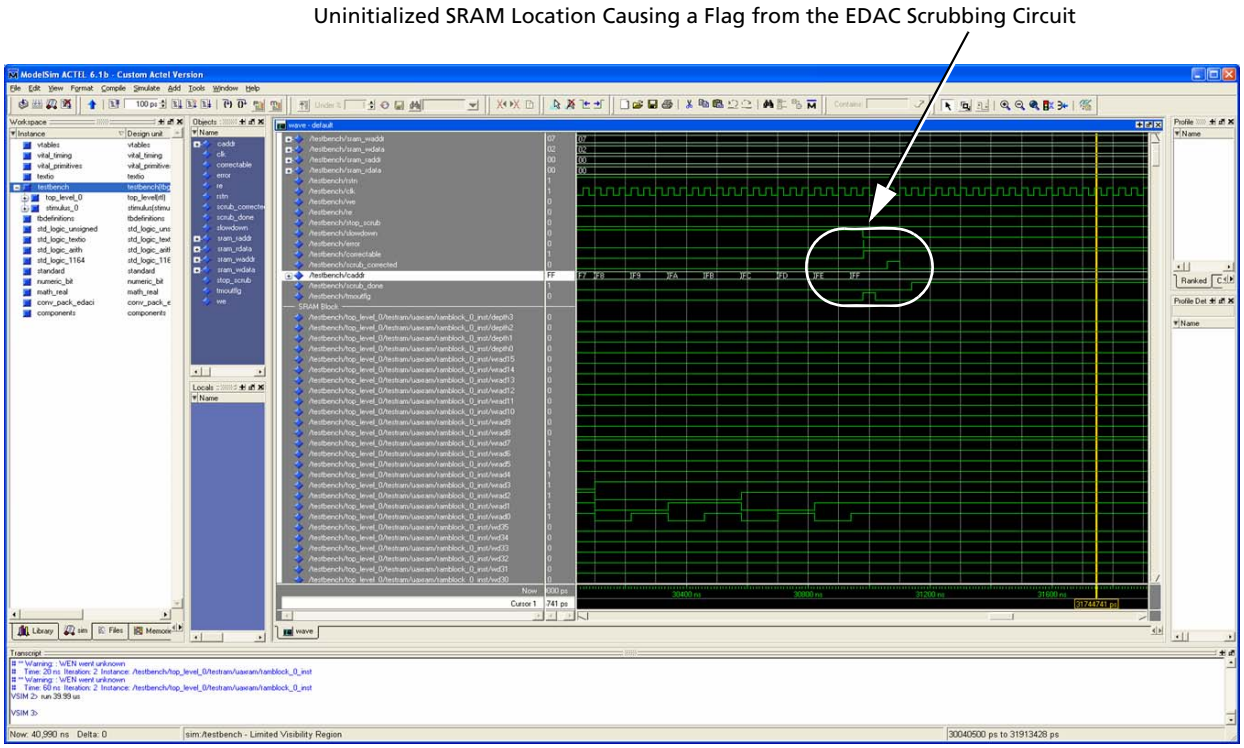


*Figure 13* • **EDAC Background Scrubber Generating Flag at the Last Memory Location**

# Summary

Semiconductor memories can exhibit SEU errors when exposed to radiation. The use of error-correcting codes such as the Actel EDAC module can improve memory reliability by correcting single-bit errors automatically and flagging uncorrectable errors so that overall system reliability is improved. Verifying that the overall system is reacting properly to such SEU events is not easily accomplished except with the use of a simulator. This application note has presented a simple method for implementing SEU events in a user design to verify the operation of a design when one or more SEU events are injected into the simulation.

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 Bldg. 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**
Suite 2114, Two Pacific Place
88 Queensway, Admiralty
Hong Kong
**Phone** +852 2185 6460
**Fax** +852 2185 6488
www.actel.com.cn

51900165-0/8.07