ON THE USE OF SCRUBBING FOR SEU MITIGATION

Rainer Storn (ROHDE & SCHWARZ GmbH & Co. KG
Radiocommunications Systems Division
81671 Muenchen, Germany, Email: rainer.storn@rohde-schwarz.com)

ABSTRACT

Today's Software Defined Radio's (SDRs) make heavy use of GPPs, DSP, and FPGAs in order to handle the high processing load incurred by the reconfigurable part of the radio. As radios and waveforms become ever more complex the memory requirement reaches the orders of many Megabytes.
In airborne environments such memory demands constitute a problem since, especially at high altitudes like. 60,000 ft there exists the problem of single event upset (SEU). These SEUs are caused by energetic neutrons hitting the surface of silicon devices causing transient errors due to bit flips.
Although there exist processors and FPGAs that are taylored to harsh environments [1], [2] there is a great tendency to use COTS devices in order to save device costs. If COTS devices are used some kind of redundancy must compensate for their SEU susceptibility. Ideally this redundancy is SW based so that it can potentially be applied to a large variety of processor devices. In this paper a technique called *fast scrubbing* is introduced to mitigate the SEU problem for DSPs and GPPs.

1. INTRODUCTION

Due to the advantages of, among others, being reconfigurable, extensible, correctable, and portable today's radios are more and more SW-defined. This implies that these Software Defined Radios (SDRs) contain a number of processing devices such as GPPs, DSPs, and FPGAs all of which are associated with memory. Memory constitutes an important flexible element in an SDR and nowadays comes in the orders of tens or even hundreds of Megabytes. Due to its advantages SDR technology is also envisaged for airborne applications which, however, poses a unique challenge to SDRs. There exists the problem of single event upset (SEU) which becomes particularly severe at high altitudes of about 60,000 ft. where airplanes and hence airborne SDRs frequently move about. These SEUs are caused by energetic neutrons hitting the surface of silicon devices causing transient errors due to bit flips especially in DRAMs and SRAMs [3], [4], [5], [6], [7].

Airborne equipment usually has very demanding requirements in terms of safety. These requirements also apply to military equipment when used in non-military situations like search and rescue. For airborne radios the loss of communication is regarded as a major event by the FAA standards ARP4754 [10] and ARP4761 [11]. A major failure condition in turn requires design assurance level C (DAL C) according to the standards DO178B [8] and DO254 [9] which are generally applied to airborne equipment. According to these standards a major event may only occur once in 100,000 flight hours if DAL C is applicable. According to [5] and [7] the FIT rate (FIT = failure in time, where time usually equals 10^9 hours) for a 1 Mbit (128kbyte) RAM at ground level is around 2000. This FIT value extrapolates to 800,000 at 60,000ft altitude as the worst case. Modern DSPs such as the TMS320C6457 ® have the 16-fold internal capacity, i.e. 2048 kByte of RAM which correspond to 1280 bit errors in 100,000 hours. Even if only 10% of the RAM content were considered as sensitive with respect to the communication functionality the resulting error rate of 128 would still be *two orders of magnitude larger* than required to meet the FAA standards. This rough estimation gets even more disadvantageous if reduced supply voltages and footprint of today's memories are taken into account [12], compared to the technologies investigated in [5] and [7].
In other words, using high end DSPs such as the TMS320C6457 ® in airborne radios without any further measures is not possible. The same holds true for GPPs in case they do not have any error correction capabilities built in.

2. GENERAL MITIGATION TECHNIQUES FOR SEU

All SEU mitigation techniques use some kind of redundancy either in space, time or both. The basic idea is to detect and correct errors which often is abbreviated by the acronym EDAC (error detection and correction) in the literature. Correction can be done by [13]:

- RAMs with built-in error correcting codes (ECC) [6]
- Selection between various outcomes (voting), a technique used in FPGAs [4], [16] and VLIW DSPs

[17] where the existence of HW and/or time parallelism is employed.
- Reconfiguration from a secure source [16]
- Scrubbing techniques [6], [14]

In this paper the focus is on scrubbing techniques. The scrubbing techniques will be regarded only for the protection of the program memory since the program memory content is static and hence can be corrected in principle. Additionally, errors in the data memory are less likely to result in the inability to communicate if proper error handling is built into the system and unless the stack memory is affected. Stack memory, however, is relatively small compared to the overall memory size and hence has a reduced probability of getting hit by neutrons.

3. CLASSIC SCRUBBING

The idea of scrubbing is to remove errors from memory before the functionality of the system under protection is compromised. This can be done in an unconditional manner where the memory content is periodically overwritten with content from an SEU-insensitive source, or it can be done conditionally where the memory is checked for errors first and overwritten only in case of errors. Classic Scrubbing has been employed to avoid two-bit errors in ECC-RAMs [6] and is done in the background.

Classic scrubbing thrives on the fact that there exists ECC memory which is able to correct single bit errors. The scrubbing mechanism analyzes the memory content in the background and detects corrected errors. After detection the codeword is replaced by its original counterpart which resides in an SEU-insensitive memory, e.g. a flash memory. This way the probability of two-bit errors, which would not be correctable by the ECC mechanism, is greatly reduced. The idea of this mechanism is visualized in Figure 1.

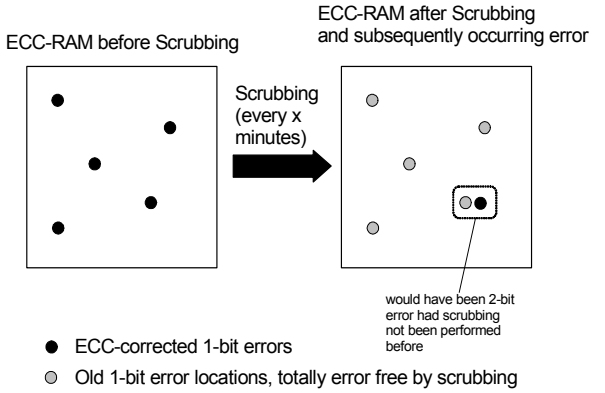

Figure 1: Mechanism of classical scrubbing. One-bit errors are removed before a second SEU-event can cause a two-bit error.

4 FAST SCRUBBING

In order to keep manufacturing costs at a reasonable level it is desirable to equip modern SDRs with standard COTS DSPs and GPPs. However, these components, not being originally intended to be used in airborne applications, often utilize external and/or internal memory which is not ECC protected. Examples for such components are the DSP TMS320C6457 ® which has no ECC support whatsoever, or the ARM Cortex-A8 based OMAP ® series. The cache controller of the latter also does not support ECC RAMs. Simply putting the program code into external SEU-insensitive flash memory is not effective since the increased access time of flash memories would unduly slow down program execution. If COTS processors, as the ones mentioned above, shall still be employed in airborne applications one has to find ways to protect the inherent non-ECC memories from SEU events.

This paper investigates whether the scrubbing method can be used in a foreground process to counteract 1-bit-errors in unprotected memories rather than to solely fulfill its classical purpose as a background job counteracting 2-bit errors for ECC-protected memories. This novel idea of using scrubbing as a foreground job will be referred to as *"fast scrubbing"* in the following.

Fast scrubbing utilizes the idea that if scrubbing of a code area is performed immediately before the code portion is used, then the opportunity for a neutron to damage this code portion reduces to the time interval of code usage.

Consider, for example, a simple round robin scheduling algorithm as depicted in Figure 2.

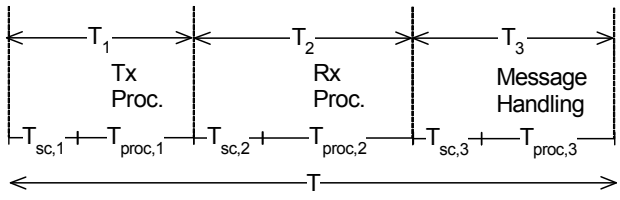

Figure 2: Example of a simple round robin scheduling system.

Let us assume that all code portions for Tx processing, Rx processing, and message handling are scrubbed before they are used. In this case the probability for a detrimental SEU event becomes

$$P(SEU_{total,scrub}) = \frac{\sum_{i=1}^{N} P(SEU_i)\cdot T_i}{\sum_{n=1}^{N} T_n} = \frac{\sum_{i=1}^{N} P(SEU_i)\cdot T_i}{T} \quad (1)$$

where the times T_i contain both the times $T_{sc,i}$ for scrubbing as well as the times $T_{proc,i}$ for processing. For the sake of simplicity let us further assume that for all of the time intervals T_i we have $T_i = T/3$ and for all probabilities the identity $P(SEU_i)=p_0$ holds. Under these assumptions we obtain in the case of no scrubbing

$$P(SEU_{total}) = \sum_{i=1}^{N} p_0 \overset{N=3}{=} 3\cdot p_0$$

while in the case of fast scrubbing we get

$$P(SEU_{total,scrub}) = \frac{\sum_{i=1}^{N=3} p_0 \cdot \frac{T}{3}}{T} = p_0$$

i.e. we obtain a probability for a detrimental SEU event which is three times less than it would have been if no fast scrubbing had been performed.

It can easily be seen that the probability for a detrimental SEU event decreases with the number of memory portions that are scrubbed before usage. In the simple example of Fig. 2 only three portions Tx processing, Rx processing, and message handling are used. It is, however, the goal to find more sections the memory can be divided into and which are used in succession. A further breakdown of memory may be, for example: Tx signal processing, Tx bit processing, Rx signal processing, Rx bit processing, configuration message handling, and notification message handling. Depending upon the architecture of the radio SW a much finer granularity of the code may be possible.

In order to be able to use a scrubbing scheme which preserves the integrity of the program memory before it is used two prerequisites must be fulfilled:

- It must be possible to detect memory errors
- It must be possible to correct memory errors

Several schemes which provide this are conceivable, two of which are schown below.

4.1 SOFTWARE-BASED ECC

The notion of software-based ECC is depicted in Fig. 3 in a simplified manner. The basic idea is to use systematic codes to represent the program memory content, i.e. the executable code is placed in a cleartext section of the memory and the redundant information is placed in the parity section. Prior to execution of, say, the Tx part the pertinent code memory partition is checked whether errors have occurred. If errors have occurred they will be repaired due to the capability of the error correcting code. Two assumptions are made here:

- The checking of the code is done often enough so that, with sufficient probability, occuring errors are just one-bit errors.
- The time for code checking and repairing is small compared to the time for code execution. Since it is also assumed that errors are a rare event the number of code words that can be corrected in the allocated time $T_{sc,i}$ is much smaller than the number n of codewords.

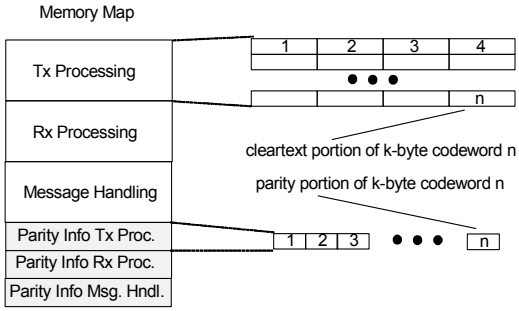

Figure 3: For SW-based ECC the memory is partitioned in a cleartext section and a parity section.

As will be discussed later this SW-based scheme is impractical in most cases due to the large amount of processing power needed for the error correction.

4.2 CODE RECOVERY FROM A SECURE SOURCE

The second scheme for fast scrubbing also divides the program memory into a cleartext section and a parity section. In contrast to the previous scheme the parity section only serves as a means to detect errors rather than correct them which allows for simpler codes. If errors are detected the codeword under test is replaced by its original counterpart which is taken from an SEU protected memory like a flash memory. In contrast to the previous scheme the error correction is not performed by using error syndrome analysis but simply by copying the corresponding memory partition from a secure source, i.e. memory expense is traded against processing power.

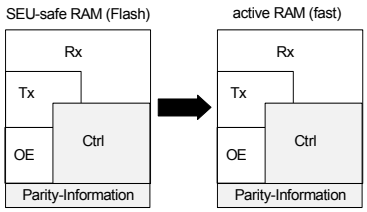

Figure 4: Fast Scrubbing by restoring data content from an SEU-safe RAM.

4.3 OPTIMUM CODEWORD SIZE

Assuming that a fast scrubbing method shall be used for SEU protection improvement the most efficient codeword size for the memory portion to be checked needs to be determined. To this end the following quantities are defined:

- A memory partition M to be checked consists of N codewords having a size of k bytes, i.e.
  $$M = N \cdot k \quad (2)$$
- The time t_{detect} needed to detect an error in a codeword is assumed to be proportional to k according to
  $$t_{\det ect} = t_{\det\_const} + c_1 \cdot k \quad (3)$$
  The time t_{det_const} is a processor dependent quantity which accounts for the initialization of the error detection mechanism.
- The time $t_{correct}$ needed to correct an error is modeled by
  $$t_{correct} = t_{corr\_const} + c_2 \cdot k \quad (4)$$

All times shall be measured in processor cycles.

The total time T for fast scrubbing, assuming that there is only one error in the code memory portion since SEUs are rare events, may be computed according to:

$$T = N \cdot t_{\det ect} + t_{correct}$$
$$= N \cdot t_{\det\_const} + N \cdot c_1 \cdot k + t_{corr\_const} + c_2 \cdot k$$

By substituting N with M/k we get

$$T = \frac{M}{k}\left(t_{\det\_const} + c_1 \cdot k\right) + t_{corr\_const} + c_2 \cdot k \quad (5)$$

It can be seen that the smaller the codeword size k the smaller the time to correct an error will be. On the other hand the overhead to restart the error detection increases with decreasing k, so in order to find the minimum scrubbing time T one has to compute

$$\frac{\delta T}{\delta k} = -\frac{M}{k^2}\left(t_{\det\_const}\right) + c_2 = 0$$

Employing standard calculus it can be computed that the optimum codeword size is

$$k_{opt} = \sqrt{\frac{M}{c_2}\left(t_{\det\_const}\right)} \quad (6)$$

so the optimum codeword size k is proportional to the square root of the memory partition size M. Since N=M/k also the optimum number of codewords N_{opt} is proportional to the square root of M.

4.3.1 EXAMPLE OF 3 PARTITIONS OF SIZE 100KB

In a simplified example we assume a total program memory size PM of 300kByte which is partitioned into p = 3 independent partitions of size M = 100kbyte. The 3 partitions may resemble Tx processing, Rx processing, and message handling as described in chapter 4.

In order to obtain realistic constants of the detection and correction times a DSP running at 400MHz clock and a 4-Byte read time of 70ns from a secure flash memory (needed for the method described in chapter 4.2) are assumed. In this case we get

$$t_{\det\_const} = 14cycles, \quad c_1 = \frac{0.5cycles}{Byte}$$

as well as

$$t_{corr\_const} = 0, \quad c_2 = \frac{7cycles}{Byte}$$

so finally there is

$$\frac{k_{opt}}{Bytes} = \sqrt{\frac{100 \cdot 2^{10} \cdot 14}{7}} = 10 \cdot 2^5 \cdot \sqrt{2} \approx 453$$

In this case N amounts to

$$N = \frac{M}{k_{opt}} = \frac{100kByte}{453Byte} \approx 226$$

For the current example the total scrubbing time would be

$$\frac{T_{\min}}{cycles} = 226 \cdot \left(14 + 0.5 \cdot 453\right) + 7 \cdot 453 = 57536$$

which, for a DSP running at 400MHz clock speed, amounts to roughly 145µs.

If it is further assumed that the other portions for Rx processing and control contribute the same scrubbing delay there would be a total scrubbing time of 435 µs. If it is further assumed that the total processing delay of a waveform must be smaller or equal to 3ms then the fast scrubbing would consume about 14% of this maximum delay which appears to be tolerable.
Note that in this example the number of cycles for codeword correction is 6328, a value which is orders of magnitude lower than what would be required by a SW-based RS decoding solution [18]. Hence the scrubbing method of chapter 4.1 doesn't appear to be promising in this scenario.

4.3.2. SCRUBBING DELAY EXPENSE

The example above for p=3 reduces the SEU probability by a factor of three which may not be enough depending on the total program memory size PM and the design assurance level (DAL) of the software. So it is crucial to investigate the scrubbing delay expense as a function of PM and the number p of independently scrubbable memory partitions M. This is because the larger PM the more code can be stored, and the larger p the lower the probability will be that an SEU event affects the system performance. Fig 5 displays the aforementioned analysis which is based on eqs. (5) and (6).

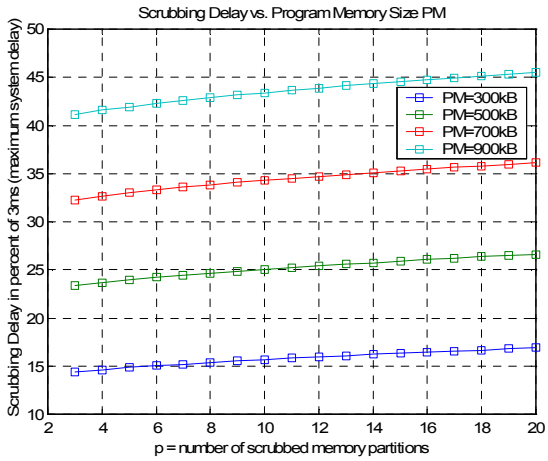

Figure 5: Scrubbing delay expense over PM and p.

It can be seen that the total scrubbing delay expense is only weakly dependent on the number p of partitions. So the increase of p comes at a comparatively low cost with respect to delay.

In chapter 1 the example for the TMS320C6457 ® revealed the need for a two orders of magnitude improvement in error rate to meet the FAA requirements provided that the entire internal memory of 2048kByte is used. If this improvement must be achieved by fast scrubbing alone eq. 1 suggests that roughly p=100 memory partitions need to be identified. Besides being a special challenge to find this many functionally separate partitions the partitioning itself has a major impact on SW architecture the consequences of which have yet to be investigated.

The total program memory size PM has a large impact on the scrubbing delay. The scrubbing scheme may become disadvantageous for a value of PM greater than 500kByte, since above this memory size roughly 25% of the system delay are attributed to scrubbing. On the one hand this reduces the number p, on the other hand much of the internal memory in the above example would need to remain unused. This calls for alternatives to relieve the scrubbing delay penalty.

5. PARALLELIZED SCRUBBING

Many modern DSPs use several parallel processing units, e.g. the TMS320C64x+™series, employing the third-generation high-performance, advanced VelociTI™ very-long-instruction-word (VLIW) architecture developed by Texas Instruments (TI). The C64x+ DSP core employs eight functional units, two register files, and two data paths, A and B.

A potential way to reduce the scrubbing delay is to dedicate one of the data paths, say data path B, for scrubbing while the other path A is used for regular processing. Of course data path B can also be used for regular processing whenever scrubbing allows to do so. The scheme is indicated in Fig. 6 showing the simple example from chapter 4.

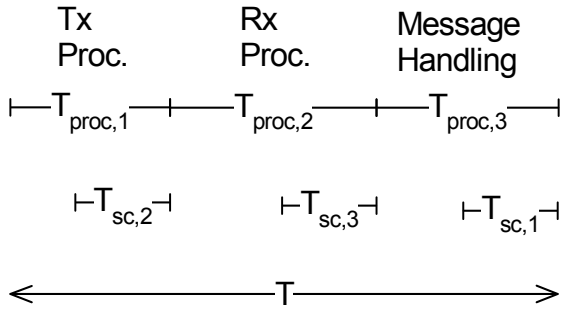

Figure 6: Scrubbing example from chapter 4 where scrubbing is done in parallel with regular processing.

As all SEU mitigation methods this parallelization of scrubbing comes at a price. In this case the price is a reduction of available computing power for SDR functionality. On the other hand, this scrubbing scheme can be put into effect using cost-efficient COTS processors.

6. RESULTS AND FUTURE WORK

It has been shown that scrubbing has more to offer than to just serve as a slowly-running background job which minimizes the probability of two-bit errors in ECC memories. The novel approach of "fast scrubbing" has been analyzed where scrubbing is executed in the foreground and removes SEU-based bit-flips in unprotected memory by repairing memory content right before its use. Fast scrubbing is especially interesting if modern COTS

processors shall be employed since these often have large internal memories which are not ECC protected.

Two versions of fast scrubbing have been discussed, a SW-based scrubbing scheme based on error correction codes, and a flash-based code recovery scheme which checks the memory for errors and replaces erroneous memory partitions with the original ones from a flash memory.

An analysis revealed that the SW-based scheme appears infeasible due to the large computational requirements for executing error correcting codes like Reed-Solomon, especially if the codewords need to be large. The code recovery scheme has been shown to be attractive if the memory size to be protected is moderate and the minimum system delay of the SDR is in the millisecond range. The computational delay of the scrubbing schemes may be reduced further by doing the scrubbing in parallel to the regular processing. Parallelization can be achieved by utilizing the availability of multiple computational units and data paths in modern processors.

The odds are that fast scrubbing alone cannot fully compensate for the lack of ECC memory, either because the required number of memory partitions becomes prohibitive rendering too many restrictions concerning the SW architecture, or because the scrubbing delay becomes too high. Hence fast scrubbing probably but must be supported by other means such as code sensitivity analysis or SW-based voting mechanisms. These findings advise a more detailed analysis of various processor types and the impact of fast scrubbing on SW architecture.

7. REFERENCES

[1] http://focus.ti.com/hirel/docs/prodcateglanding.tsp?sectionId=603. Texas Instruments SPACE Orderables, Texas Instruments, aug. 20009,

[2] http://www.actel.com/documents/, System-Critical FPGA Products Catalog, ACTEL, nov. 2009.

[3] Muhammad Imran, *Using COTS components in space applications, Masters Thesis*, Univ. Delft, 2006

[4] Fernanda Gusmão de Lima, *Single Event Upset Mitigation Techniques for Programmable Devices*, Masters Thesis, Univ. Rio Grande, 2000

[5] E. Normand, "Single event upset at ground level," IEEE Trans. Nucl. Sci., vol. 43, pp. 2742–2750, Dec. 1996

[6] Riccardo Mariani, Gabriele Boschi, *Scrubbing and Partitioning for Protection of Memory Systems*, Proceedings of the 11th IEEE International On-Line Testing Symposium (IOLTS'05), pp. 1530-1591

[7] R. Joshi and R. Daniels, Radiation Hardness Evaluation of a Class V 32-Bit Floating-Point Digital Signal Processor, IEEE NSREC, 2005.

[8] Software Considerations in Airborne Systems and Equipment Specification, RTCA/DO178B, dec. 1, 1992.

[9] Design Assurance Guidance for Airborne Electronic Hardware, RTCA/DO254, april 19, 2000.

[10] Certification Considerations for Highly-Integrated Or Complex Aircraft Systems, http://www.sae.org/technical/standards/ARP4754

[11] Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, http://www.sae.org/technical/standards/ARP4761

[12] Akira Eto et alii, Impact of Neutron flux on Soft Errors in MOS Memories, IEDM '98 Technical Digest, 1998, pp. 367 - 370.

[13] Anthony Lai, *Mitigation techniques for electronics in Single Event Upset environments*, Military Embedded Systems, 2006.

[14] Shubhendu Mukherjee et alii, *Cache Scrubbing in Microprocessors: Myth or Necessity ?,* Intl. Symp.PRDC, 2004.

[15] Tezzaron Semiconductor, *Soft Errors in Electronic Memory* - A White Paper," Version 1.1, white paper (2004); see http://tezzaron.com/about/papers/soft_errors_1_1_secure. pdf.

[16] Altera Corp., *Robust SEU Mitigation With Stratix III FPGAs*, white paper, january 2007.

[17] David Czajkowski (Space Micro Inc.), Rad Hard High Performance Computing, Spaceborne Computing Workshop 2008.

[18] Dahnoun Naim and Tarazona Luis, *Development and performance comparison of a Reed-Solomon Decoder for TMS320C6201 and TMS320C6400 DSPs*, 2000 European DSP Education & Research Conference - Poster Presentation Proceedings, ,