MSc Eng. Marcin Stolarski
Warsaw University of Technology
Faculty of Electronics and Information Technology
Institute of Radioelectronics
M.Stolarski@elka.pw.edu.pl

STEC 2005

Program methods of masking errors in programs, data and calculations, which are the result of computer malfunction caused by space radiation.

***Abstract*: During operations in space, computer is exposed to radiation. One of the results of this radiation is a single bit error (single event upset) in ram memory, processor or in another logic component of a computer. This error can stop processor or (which is very dangerous) can generate wrong results of calculation, because of invalid data processing or running an illegal program code. To prevent such situation we can use hardware solutions like excess encoding of data (for example EEC to the RAM memory) or triple redundancy of processing units (voting the right result). In this brief I would like to present another way of solving this problem – program masking. I shall present tree methods of masking: library of templates with triple redundancy of data and calculation procedures, error detection in programs using the CFCSS method and triple redundancy of processes.**

The first method relies on developing a library of template classes for C++ language. In code of template class, an algorithm will be placed making the triple data redundancy. The next task of this template is the overloading of the operators (for example: +) which will provide the triple calculation for redundant data. Variables in code (primitives) have to be replaced with variables from this library. Use of this library will prevent us from single errors of data and single errors of counting, at cost of processor memory usage and processor time.

The next method is to change the program code to the network-like structure. The nodes of the network are conditional instructions and the edges represent code between these conditional instructions. Each node has its own name - signature. CFCSS algorithm checks if the program code is making 'permitted' jumps. If not, and the program made the 'forbidden' jump, it is highly possible it has happened because of executing wrong (changed by SEU) instruction or data.

The third method is to run three identical processes simultaneously. If one of these processes achieve different results comparing to the other two, we can assume that this process is operating on invalid data or has not executed properly and should be terminated. Next step is to fork one of unaffected processes, to keep constant number of redundant processes. This method seems to work very good and raises significantly immunity of computer system for errors having source at single event upsets. Nevertheless, this method creates new problems such as synchronization between processes or using the real time tasks.

This brief is about pros and cons of these three methods and their influence on making computer systems in high radiation environments more reliable.

1. Introduction

Electronic hardware and software working in space is vulnerable to radiation that may cause malfunctions. To provide reliable life time many hardware solutions are used as in-system error correction or radiation hardened technology. But these solutions are not cost effective, and implicates on board design from scratch. On the other hand, software solutions are not that reliable, but much more cost-effective and easy to duplicate.

2. Space enviroment conditions

For Earth Orbit's satellites space environment varies from, let us say, neutral on LEO (Low Earth's Orbit), through hostile Van Allen belts area to unprotected outer space. Based on research [1], it has been found out, that commercial electronics is radiation-proof up to 10krad. Unfortunately, this value is to small for reliable work in space. To decrease interaction of radiation and electronics, hardware is shielded by aluminum chassis of thickness above 5 mm. That wall thickness should protect OBE (On Board Electronics) against radiation below, and above Van Allen belts. To prevent satellite from general failure during passing the Van Allen belts On Board Computer should be powered off. In high-budget solutions some hi-tech specialized components are used, that the Author will describe later in the article.

Electronics in space is vulnerable to several types of damage. Under influence of ionizing radiation such things can happen:

- SEL Single Event Latchups - turning on the parasitic tiristor in CMOS structure which usually effects is short-circuiting power lines.
- SEB Single Event Burnouts - destruction of MOSFET structure - effects of SEL
- SET Single Event Transients - voltage peaks, very dangerous in combinatorial logic where it can act as false clock edges
- SEU Single Event Upsets - changes in flip-flop state.
- SEL is a very dangerous phenomena, because it is a source of non-temporary electronic damage. As a protection, thick chassis walls are used, as well as devices being made in special technology.
- SET and SEU are nod-destructive errors, but cause improper acting of logical elements and could gener-

ate false data. This can be avoided by using hardware redundancy and special algorithms in software.

3. Hardaware methods of error masking

Hardware solutions can be divided into several categories. Every of these increases computer systems immunity against radiation, but does not guarantee it. That is why it is very important to apply as many of this solutions in single project as possible. First solution for increasing hardware radiation immunity is the use of proper technology. In the older satellites electron lamp electronics was applied. There were no observed influences of radiation on that type of electronics, but that technology produced devices utilizing much of weigh, space and power budget. Also, they were not very complicated. In modern spacecrafts IC's are commonly used which are made in technologies using bigger smallest size of structure than in typical earth environment. There also can be found a new generation of IC's made in Silicon On Insulator technology.

Another category of solutions is hardware redundancy. In this way, several identical logic components are used to process the same data. The return data is then processed by voting unit, whereas data from malfunctioning device should be over voted by data from other devices (possibly working properly). This could be applied to several modules of which On Board Computer consists of, for example: a memory. Even whole computer systems could be redundant. This solution is highly reliable, but also one of the most expensive (this idea is used in NASA's Space Shuttle where in voting process 5 computers take part).

Another group of hardware solutions are correction codes used in memories. When error is detected, system is capable to recover true data using an excess coding. If excess co/decoding is implemented in hardware, whole protection could be transparent from the system point of view.

When all methods fail, and computer system breaks down, it is common that reset is enough for recovery. Very simple device providing this functionality is a program called Watchdog. Running system should set watchdog with information of proper functioning. It could be even one pulse on selected line. When there is no pulse in determined period, watchdog overflows, and resets master system. Simple and effective.

4. Software methods for error masking.

Another way of error masking is software error masking. It is based on additional computing and additional memory size, directed to mask errors caused by space radiation. Unfortunately, this method can deal only with temporary damage (SEU), or with damage that does not affect execution of code (partially damaged memory or CPU, which influences the effect of code execution not fetching nor decoding).

4.1. CFCSS - Control flow checking by software signatures

```
//File name: ftl.h
//Author: Marcin Stolarski

//Fault Tolerance Digit Template
template <class T> class FTDT {
   private:
      T x1, x2, x3;

   public:
      FTDT();
      FTDT(T in);
      ~FTDT();
      void compare();
      FTDT<T>& operator+(FTDT<T>& in);
      FTDT<T>& operator+(T in);
      FTDT<T>& operator++();
      FTDT<T>& operator-(FTDT<T>& in);
      FTDT<T>& operator-(T in);
      FTDT<T>& operator--();
      FTDT<T>& operator*(FTDT<T>& in);
      FTDT<T>& operator*(T in);
      FTDT<T>& operator/(FTDT<T>& in);
      FTDT<T>& operator/(T in);
      FTDT<T>& operator=(FTDT<T>& in);
      FTDT<T>& operator=(T in);
      int  operator==(FTDT<T>& in);
      int  operator==(T in);
      int  operator<(FTDT<T>& in);
      int  operator<(T in);
      int  operator<=(FTDT<T>& in);
      int  operator<=(T in);
      int  operator>(FTDT<T>& in);
      int  operator>(T in);
      int  operator>=(FTDT<T>& in);
      int  operator>=(T in);
      T x();
      void show();
      void showf();

};

//------------------------
//Fault Tolerance Digit Template metods
template <class T> FTDT<T>::FTDT() {
   x1=0;
   x2=0;
   x3=0;
   compare();
};

template <class T> FTDT<T>&
FTDT<T>::operator+(FTDT<T>& in){
   x1+=in.x1;
   x2+=in.x2;
   x3+=in.x3;
   compare();
   return *this;
};

template <class T> void FTDT<T>::compare()
{
   while (!((x1==x2)&&(x2==x3))) {
      if (x1==x2) x3=x2;
         else if (x2==x3) x1=x2;
            else x2=x1;
   }
};

//------------------------
//Predefined types
typedef FTDT<int> FTint;
```

Fig 1. Template class FTDT with example of methods and execute.

First presented solution is an idea to distribute whole code over a graph of unforbidden passes. Nodes of this graph are branch instructions, and edges of this graph are bulk code gathered between these branch instructions. Each node is marked with proper signature. CFCSS algorithm checks if flow of control over program code goes through possible ways, by comparing run-time processed signatures with static one, embedded in code by compiler (or post-processor). With this control we are able to detect changes in execution flow that are not possible in correctly executing software.

4.2. Tripled data and calculations redundancy.

Another interesting mechanism of improving system stability is tripled redundancy of data and calculations.

Author proposes using special library of variables, which implies tripling data and calculations made on that data.

The idea is very simple. When using C++ it is obvious that we can use classes with overloaded operators instead of simple variables [Fig. 1].

Now, instead of using common "int" type, "FTint" shall be used, and methods of this class shall mask data errors from RAM and calculation errors made during computation in CPU. All calculations are made 3 times on tripled data, and then data integrity is checked. Unfortunately, overloading operators do not give all properties of normal variables, and when using "printf", in case of displaying value, "x" method should be used. This problem does not exist when using iostream library instead of stdio. Of course, this method does not protect program code in any way.

4.3. Tripled process redundancy.

Another way of protecting systems from software errors generated by radiation is process redundancy. It is based on tripling running processes, and voting it's outcome is voted. If one of the processes appears to be erroneous, it is killed, and one of the two that seems to work properly is forked - duplicated, to keep the same amount of processes all the time. Keep in mind that when running on single processor machine these processes do not execute in parallel, but are switched continuously. This could cause loss of synchronization so it is important to keep some mechanisms providing it.

Another problem is a real-time tasks processing. Usually, these tasks contain time stamps which will differ in each of this tripled processes. Despite that this solution has great advantage - it protects data and program code at the same time.

In special cases we could imagine 3 virtual machines running on one computer which will obtain exactly the same amount of processor time and other system resources.

Results of computation shall be voted on the virtual machine level. In theory, this solution could replace triple redundancy of computers with voting system. Unfortunately, it is very hard to predict if the realization of this idea is applicable in addition to the fact that virtual machine's software should protect own resources against errors.

4.4. Data storage systems.

It has been a long time since the first data storage system was protected against errors in stored data. In this system data is usually stored on hard discs, which can be easily damage, so solutions like RAID 1 (mirroring) or RAID 5 have been developed.

In RAID 1 the same data is stored on 2 discs. If data on one disc is lost or damaged (damage is detected on discs and sectors), it still remains valid on another one. Unfortunately this implies hardware doubling. It is possible to mirror data on one disc, but only errors originating from damaged sector are covered.

In RAID 5 data is distributed over n=3 (max. n= 32) discs. Algorithm distributes data over n-1 discs, and on the last one checksum is kept. If a disc containing checksum is destroyed, system alerts, but data remains safe. If data disc is malfunctioning, system is able to recover data stored on it relying on other data discs and the checksum. At cost of processing power data integrity is kept. If RAID 5 service is implemented in software, storage system is slowed due to complicated algorithms of data recovery.

After discs are replaced, they are automatically coupled into the system.

5. Summary

Building computer systems for space environments is not an easy task and is very challenging for their constructors. System should be reliable and able to self-recover from minor malfunctions. It is intended to work in harsh environment: high radiation, high temperature changes, vacuum. On the other side there are limitations in weight and power consumption.

In this paper, methods of hardware error masking like redundancy or overflow coding used in space industry have been shown. These solutions are good but expensive. Use of COTS elements does not provide enough error protection.

In the second part of the paper, software error masking methods have been shown, which, at cost of memory utilization and processing power, are able to replace hardware methods.

Author's idea of prototypes library using tripled data and calculations redundancy for covering CPU and ram errors has been proposed. Also, Author proposed using of virtual machines in terms of replacing hardware and system redundancy. Unfortunately, these solutions are still in testing phase or theoretical discussions and need more research to be carried out.

If research prove a significant increase in overall protection using only software methods, it would boost development of space computer systems due to decrease of its prototyping and final costs.

References

[1] National Semiconductor *Radiation Owner's Manual*

[2] dr inż. Krzysztof Mroczek *Porównanie technologii układów FPGA pod względem podatności na błędy spowodowane przez promieniowanie jonizujące w warunkach ziemskich.*

[3] Nahmsuk Oh*, Philip P. Shirvani and Edward J. McCluskey *Control Flow Checking by Software Signatures* IEEE Transactions on Reliability Special Section on: Fault Tolerant VLSI Systems