ON THE STUDY OF THE EFFECTIVENESS OF SW-BASED FAULT HANDLING MECHANISMS TO COPE WITH IC CONDUCTED ELECTROMAGNETIC INTERFERENCE

F. Vargas, D. Brum, D. Prestes, L. Bolzani, D. Lettnin, G. Rodrigues

Electrical Engineering Dept.
Catholic University – PUCRS
Av. Ipiranga, 6681. 90619-900 Porto Alegre – Brazil
vargas@computer.org

Abstract

We present hereafter a study on the effectiveness of two software-based fault-handling mechanisms in terms of detecting harmful conducted electromagnetic interference in ICs. Originally, these techniques were proposed to protect systems against transient faults in memory elements (i.e., single-event upsets: SEUs). One of these techniques deal with processor control flow checking. The second one is used to detect errors in code variables. In order to check the effectiveness of such techniques in RF ambient, spurious electromagnetic-induced noise according to the International Standard IEC 61000-4-29 Normative was modeled and injected into the supply lines of a commercial off-the-shelf (COTS) microcontroller-based system. Experimental results suggest that the considered techniques present a marginal effectiveness to detect this type of faults. This is due probably to the multiple-fault injection nature of electromagnetic interference in the processor control flow and data, which in most cases results in a complete system functional loss (the system must be reset).

Keywords: *Electromagnetic Interference, Electromagnetic Immunity (EMI), Conducted RF Noise Modeling, International Standard IEC 61000-4-29 Normative, Design-for-Electromagnetic Immunity (DEMI), Software-Based Fault Detection, Fault Injection.*

ON THE STUDY OF THE EFFECTIVENESS OF SW-BASED FAULT HANDLING MECHANISMS TO COPE WITH IC CONDUCTED ELECTROMAGNETIC INTERFERENCE[1]

F. Vargas, D. Brum, D. Prestes, L. Bolzani, D. Lettnin, G. Rodrigues

Electrical Engineering Dept.
Catholic University – PUCRS
Av. Ipiranga, 6681. 90619-900 Porto Alegre – Brazil
vargas@computer.org

Abstract

We present hereafter a study on the effectiveness of two software-based fault-handling mechanisms in terms of detecting harmful conducted electromagnetic interference in ICs. Originally, these techniques were proposed to protect systems against transient faults in memory elements (i.e., single-event upsets: SEUs). One of these techniques deal with processor control flow checking. The second one is used to detect errors in code variables. In order to check the effectiveness of such techniques in RF ambient, spurious electromagnetic-induced noise according to the International Standard IEC 61000-4-29 Normative was modeled and injected into the supply lines of a commercial off-the-shelf (COTS) microcontroller-based system. Experimental results suggest that the considered techniques present a marginal effectiveness to detect this type of faults. This is due probably to the multiple-fault injection nature of electromagnetic interference in the processor control flow and data, which in most cases results in a complete system functional loss (the system must be reset).

Keywords: *Electromagnetic Interference, Electromagnetic Immunity (EMI), Conducted RF Noise Modeling, International Standard IEC 61000-4-29 Normative, Design-for-Electromagnetic Immunity (DEMI), Software-Based Fault Detection, Fault Injection.*

Terminology:

- **Conducted electromagnetic interference (Conducted-EMI):** transients and/or other disturbances observed on the external terminals of a device during its normal operation. (In general, the external terminals are the power supply lines.)
- **Electromagnetic Environment (RF ambient):** totality of electromagnetic phenomena existing at a given location.
- **Electromagnetic compatibility (EMC):** ability of an integrated circuit, system or equipment to function satisfactorily in its electromagnetic environment without introducing intolerable electromagnetic disturbances to anything in that environment.
- **Radiated electromagnetic interference (Radiated-EMI):** phenomena by which energy in the form of electromagnetic waves emanates from a source into space (or: energy transferred through space in the form of electromagnetic waves.) to affect electronic systems.
- **Immunity (to a disturbance):** the ability of a device, equipment or system to perform without degradation in the presence of an electromagnetic disturbance.
- **Short interruption:** the disappearance of the supply voltage at a point of the low voltage D.C. distributed system for a period of time typically not exceeding 1 min. In practice, a dip with a least 80% of the rated voltage may be considered as an interruption.
- **Voltage dip:** a sudden reduction of the voltage at a point in the low voltage D.C. distribution system, followed by a voltage recovery after a short period of time, from a few milliseconds up to a few seconds.
- **Voltage variation:** a gradual change of the supply voltage to a higher or lower value than the rated voltage. The duration of the change can be short or long.

1. Introduction

The electromagnetic EM environment in which electronic systems have to operate is becoming increasingly hostile while dependence on electronics is widespread and increasing. The need for assurance that application upsets due to the EM environment will not occur is fundamental to acceptance of systems as fit for purpose. In order to solve such problems, design features to impart EM hardness (i.e., Design for Electromagnetic Immunity – DEMI) are beginning to be implemented, but at very high cost in terms of system performance, power consumption, and implementation complexity [1-3].

[1] *This work is partially supported by CNPq and FAPERGS.*

Fig. 1 summaries technology trends impact on ICs [1]. Note in this figure that although the reduction of supply voltages (at least for the core part) rises the hope for less electromagnetic emission (conducted and radiated), this benefit is immediately compensated by a drastically increased number of simultaneously switching transistors per die, combined with faster switching edges due to increasing clock rates. Thus, increasing the total RF noise that can affect embedded functional blocks inside the die itself, as well as affect other dies or ICs placed nearby it. For instance, it is well known that dynamic switching currents in the supply lines on the silicon die are one of the main sources of *radiated electromagnetic emission* which causes the power lines to behave as antennas and to radiate undesired noise. This RF signal induces embedded (more sensitive) functional blocks in the die to suffer from spurious current switching spikes. In addition to affect the functional blocks, these current spikes are conducted in the form of noise outside the IC through the supply and/or data lines and may affect several other components mounted on the application board [4].

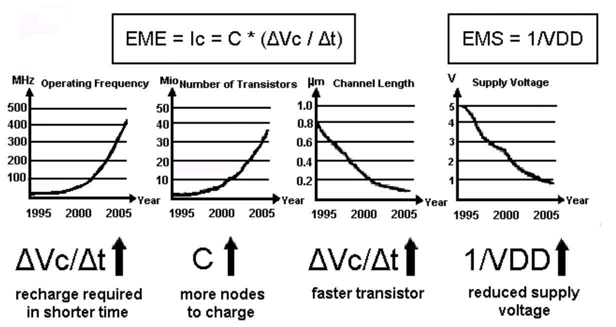

Fig. 1. Technology trends impact on ICs [1]. (**EME**: Electromagnetic Emission; **EMS**: Electromagnetic Susceptibility.)

Most of the solutions found in the literature dealing to minimize electromagnetic emissions are design-based propositions [1]. In general, they intend to *reduce the dynamic switching currents* or *optimize distribution of switching currents over time*. As examples, *block decoupling capacitors* and *improved pad-drivers design* contribute to the first, while *clock concepts with intentional non-zero skew* to the second. Note that since I/O signals are important contributors to undesired RF emission in electronic systems, the design of pad drivers as weak as possible could contribute to minimize RF emission. However, note also that this solution leads to a more sensitive IC to noise as well as may expose it to delay faults since transistors become slower with higher temperatures.

Considering the design of ICs with non-zero skew clock signals, this measure effectively reduces electromagnetic emission. However, clock smearing by defining rising and falling edges to occur at different times along with the supply lines plane goes against technology scaling, since a "good" design mandates the implementation of ICs with as perfect as possible synchronized (zero-skew) control signals distributed all over the IC functional blocks. Note also that the zero-skew trend is naturally supported by today's design tools, but if chip designers should take advantage from implementing the clock smearing concept for the sake of reduced RF emission, he/she should perform this clock distribution modification by him/herself, since today's tools do not support directly non-zero clock signal design.

Considering the above introduced, the proposed work has three objectives: **(a)** *briefly overview* the EM interference effects on electronic systems, **(b)** *evaluate the effectiveness* of some classical SW-based fault detection mechanisms in detecting EM-induced faults and **(c)** *derive alternative* low-cost approaches to improve systems EMI.

2. Preliminaries: *Conducted Electromagnetic Interference on Electronics*

The huge increase in the use of portable electronics combined with increasingly hostile electromagnetic environment has intensified the need to include design and test methods for electromagnetic immunity in particular to low power IC-based applications. In the presence of electromagnetic RF incident field (produced for example by the cellular phone transmitting antenna), cables connected to electronic systems and PCBs' tracks behave as receiving antennas capturing disturbances. The induced RF currents reach the inputs ports of ICs and often produce system malfunctioning [6].

In fact, the coupling effect between electromagnetic RF incident field and tracks routed on the die surface is lower than that one between electromagnetic (RF) incident field and PCB tracks connected to the input ports of an IC. In such a noisy environment, there are two types of failures induced by conducted RF interference on ICs [6]:

- ***a)*** *Static failures*: occur in the presence of conducted RF interference superimposed on high or low logical level. The signal at the IC input port goes out of high or low noise margins. Sometimes, it can be observed synchronization between the input signal and the RF interference. In this case, errors at the IC's output ports come from failures in the IC input ports.
- ***b)*** *Dynamic failures*: occur when conducted RF interference added to the IC input logical signal gives variation in the input port propagation delay. Thus, changing the logic gates settling time and hold time. In this case, errors due to conducted RF interference observed at the IC's output ports come from failures in internal sub-circuits.

The induced RF currents that reach the input ports of ICs may be caused from a simple voltage variation to a large voltage interruption in the IC power supply lines. Therefore, in an attempt to contribute to minimize such disruptions, we are concerned in this work with the following EM effects on electronics: **a)** *voltage dips*, **b)** *interruptions*, and **c)** *voltage variations* of the IC power supply lines. To do so, in *Section 4* we have conducted experiments based on the *International Standard IEC 61.000-4-29 Normative* [5]. This standard is one of the references to test ICs for conducted RF noise on power lines.

3. The SW-Based Fault Handling Mechanisms

This section describes in detail the two software-based techniques we have selected from the literature. In fact, a dedicated reader can find several purely software-based techniques dealing to handle, for instance, control-flow faults. Some representative examples are: [10-13]. However, in this work we decided for the two techniques [8,14] described later not only because they are purely software-implemented approaches (i.e., completely hardware-independent in order to be used in *commercial-off-the-shelf* (COTS) processors), but also because they were fully automated (there existed at the time we started this work a compiler tool [7] used to automatically modify the C-implemented code into a fault-tolerant one according to the definitions presented in the following).

a) *Faults affecting data:* having these goals in mind, we selected the technique proposed by Matteo *et al.* [8], which is based on the following three basic rules:

- **Rule #1**: every variable *x* must be duplicated: let *x1* and *x2* be the names of the two copies;
- **Rule #2**: every write operation performed on x must be performed on *x1* and *x2*;
- **Rule #3**: after each read operation on *x*, the two copies *x1* and *x2* must be checked for consistency, and an error detection procedure should be activated if an inconsistency is detected.

The above rules mean that any variable *v* must be split in two copies *v0* and *v1* and that these copies should always store the same value. A consistency check on *v0* and *v1* must be performed each time the variable is read. The check must be performed immediately after the read operation in order to block the fault effect propagation. Also, each instruction that writes variable *v* must also be duplicated in order to update the two copies of the variable.

Additionally, note that the variables should also be checked when they appear in any expression used as a condition for branches or loops, thus allowing detection of errors that corrupt the *correct control flow execution of a program*.

Every fault that occurs in any variable during the program execution can be detected as soon as the variable is the source operand of an instruction, i.e., when the variable is read. Thus resulting in minimum error latency, which is approximately equal to the distance between the fault occurrence and the first read operation. In this technique, errors affecting variables after their last usage are not detected. Fig. 2 illustrates two simple examples, by showing the code modification for an assignment operation and for a *sum* operation involving three variables *a*, *b*, and *c*.

```
Modified Code:          Original Code:
a = b;                  a0 = b0;
                        a1 = b1;
                        if(b0 != b1)
                                error();

a = b + c;              a0 = b0 + c0;
                        a1 = b1 + c1;
                        if (b0 != b1) || (c0 != c1)
                                error();
```

Fig. 2. Code modification for errors affecting data.

The parameters passed to a procedure, as well as the returned values should be considered as variables. Therefore, the rules defined previously should be extended as follows:

- Every procedure parameter is duplicated;
- Each time the procedure reads a parameter, it checks the two copies for consistency;
- The return value is also duplicated (in C, this means that the addresses of the two copies are passed as parameters to the called procedure).

Fig. 3 depicts an example of application of Rules #1 to #3 to the parameters of a procedure.

b) Faults affecting the control flow: in order to detect this type of faults, we selected also another approach proposed by Matteo and his group and first presented in [8].

For the purpose of the work proposed by Matteo, errors affecting control flow can be divided into two categories, depending on the way they transform the statement whose code is modified:

- ***Type Error-A***: errors changing the control flow by affecting data in arithmetic expressions, computations, and assignments, for instance. As example, consider an error transforming an *add* operation into a *jump* one.
- ***Type Error-B***: errors changing the control flow by affecting data in tests, loops, procedure calls, and returns, for instance. As example, consider an error transforming a data *i* into an *i'* (in the assumed example, *i* is compared to *j* in a conditional branch-taken decision instruction).

Modified Code:

```
res = search(a);
...
int search(int p)
{           int q;
            ...
            q = p + 1;
            ...
            return(1);
}
```

Original Code:

```
search(a0, a1, &res0, &res1);
...
void search(int p0, int p1, int *r0, int *r1)
{           int q0, q1;
            ...
            q0 = p0 + 1;
            q1 = p1 + 1;
            if(p0 != p1)
                        error();
            ...
            *r0 = 1;
            *r1 = 1;
            return;
}
```

Fig. 3. Code transformation for errors affecting procedure parameters.

The Matteo's solution for detecting errors of type *Error-A* is based on tracking the execution flow by trying to detect differences with respect to the correct behavior. This task is performed by first identifying all the basic blocks composing the code. A basic code is a sequence of statements, which is always indivisibly executed (i.e., they are branch-free). The following rules are then introduced, in order to check whether all the statements in every basic block are executed in sequence:

- **Rule #4**: an integer value *ki* is associated with every basic block *i* in the code;
- **Rule #5**: a global execution check flag (*ecf*) variable is defined. A statement assigning to *ecf* the value of *ki* is introduced at the very beginning of every basic block *i*. A test on the value of *ecf* is also performed at the end of the basic block.

The aim of the above rules is to check whether any error happened, whose effect is to modify the correct execution flow, and to introduce a jump to an incorrect target address. An example of this situation is an error modifying the field containing the target address in a jump instruction. As a further example, consider an error that changes an ALU instruction (e.g., an *add*) into a *branch* one: if the instruction format includes an immediate field, this may possibly be interpreted as a target address. Unfortunately, note also that the above rules have an incomplete detection capability since there are some fault types that cannot be detected (e.g., any error producing a jump to the first assembly instruction of a basic block: thus assigning to *ecf* the value corresponding to the block; or even any erroneous jump into the same basic block). Fig. 4 provides an example of application of Rules #4 and #5.

When Error-B type is considered, the issue is how to verify that the correct execution flow is followed. In order to detect errors affecting a test statement, is was introduced the following rule:

- **Rule #6**: for every test statement, the test is repeated at the beginning of the target basic block of both the true and (possible) false clause. If the two versions of the test (the original and the newly introduced) produce different results, an error is signaled.

Fig. 5. provides an example of application of the above rule. In order to simply the presentation of the rule, we do not consider in the examples the combined applications of different rules: as an example, in fig. 5 we did not apply Rules #1 and #2 to the variable named *condition*, which should be duplicated and checked for consistency after the test.

Modified Code:

```
/* Basic Block beginning */
...
/* Basic Block end */
```

Original Code:

```
/* Basic Block beginning #371 */
ecf = 371;
...
if (ecf != 371)
            error ();
/* Basic Block end */
```

Fig. 4. Example of application of Rules #4 and #5 to detect error types *Error-A* and *Error-B*.

Modified Code:

```
If (condition)
{           /* Block A */
            ...
}
else
{           /* Block B */
            ...
}
```

Original Code:

```
If (condition)
{           /* Block A */
                        if (!condition)
                        error();
            ...
}
{           /* Block B */
            if (condition)
                        error();
            ...
}
```

Fig. 5. Code transformation for a test statement (Rule #6).

4. Practical Experiments

This section synthesizes the experimental results that we have carried out in order to verify the effectiveness of the SW-based fault-tolerant techniques presented in the previous section. Having this goal in mind, we modeled and injected spurious electromagnetic noise into the power lines of a commercial off-the-shelf (COTS) microcontroller-based system. The noise modeling and injection procedure was oriented by the *International Standard IEC 61.000-4-29 Normative*, which rules testing proceedings for *voltage dips*, *voltage interruptions*, and *voltage variations*. Table I summarizes the types and durations of the noise injected into the power lines.

For each of the tests to be described hereafter, we have classified the system under test according one of the following performance criteria:

Class A: no performance degradation is verified. In this case, either no erroneous output was detected, or the error-detection techniques implemented in the system under test were able to indicate the faulty output.

Class B: performance degradation is verified only during testing. A change in state or loss of data is permissible, however, a self-recovery is expected during normal operation (i.e., after testing). In this case, one or more erroneous output was generated, but the implemented error-detection techniques were not able to detect their occurrence.

Class C: performance degradation may occur and it remains in this condition until the user/operator resets the system. In this case, one or more erroneous output was generated, and the implemented error-detection techniques were not able to detect their occurrence. As consequence, the system was set out of operation, which required the user intervention to recover from this situation.

TYPE OF DISRUPTION	Description: waveform and time duration	
Voltage Dips	a negative pulse of -30% of the Vcc power line	10ms
		100ms
		300ms
		X*
	a negative pulse of -60% of the Vcc power line	10ms
		100ms
		300ms
		X*
Short Interruptions	a negative pulse of -100% of the Vcc power line	10ms
		100ms
		300ms
		X*
Voltage Variations	a variation of –20% to +20% of the Vcc power line	continuous variation in the Vcc power line

***** *Open value(s) to be defined by the specific application or device to be tested.*

Table I. Summary of the *IEC 61.000-4-29 Normative* for voltage dips, short interruptions, and voltage variations [5].

Fig. 6 shows an oscilloscope printscreen indicating the *-30% voltage dips* injected into the microcontroller Vcc pin according to the IEC 61.000-4-29 Normative. These pulses were randomly generated in time, with a minimum of 3 pulses at every 10ms.

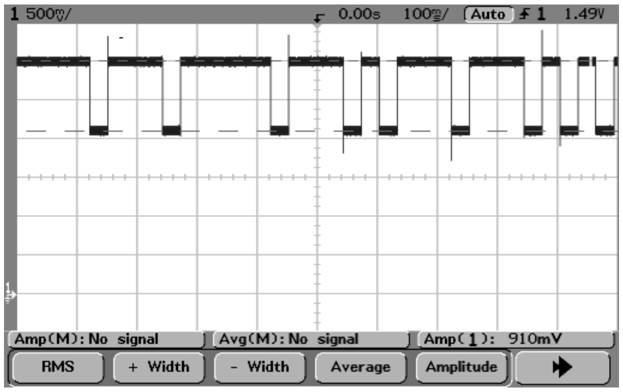

Fig. 6. Oscilloscope printscreen of voltage dips injected into the microcontroller Vcc pin. Negative pulse of -30% (-0.9V) and width of 30ms. (Nominal Vcc = 3.0V.)

Fig. 7 shows the test setup used to inject electromagnetic-induced noise into the processor Vcc power line. This setup is organized around a *PC-based host*, whose goal is to serve as interface between the user and the device under test (*MSP430-1 Board*). This server communicates with the MSP430-1 via RS232 and JTAG (optic fiber) by one side and with the fault injection controller (or *EMI Generator*): *MSP430-2 Board*, by the other side. As can be seen in this figure, the program stored in the flash memory of the *EMI Generator* deals to control the *Driver Board* by means of sending control signals to the digital-to-analog converter (DAC). Based on a reference voltage (REF), this converter translates the digital command into an analog signal which controls the operational amplifier (Op Amp, in the Driver Board). Then, the Op Amp output generates the corresponding analog "noisy Vcc signal" which is injected in the form of voltage dips into the MSP430-1 processor Vcc power pin (i.e., the device under test – DUT).

The DUT in the MSP430-1 Board is a *Texas Microcontroller MSP430F149* [9] which contains 60KB flash memory to store control program, and 2KB of RAM to store data as well as control state variable. This RAM also serves as stack area for the control part. Similarly, the MSP430-2 Board is also based on the same microcontroller device.

Table II shows results for the DUT executing as application an image linearization processing. In this example, the image consisted of 238 data bytes whose average value had to be computed (i.e., linearized) for every group of 9 pixels in order to filter noise represented by large discrepancies present in neighbor pixels values.

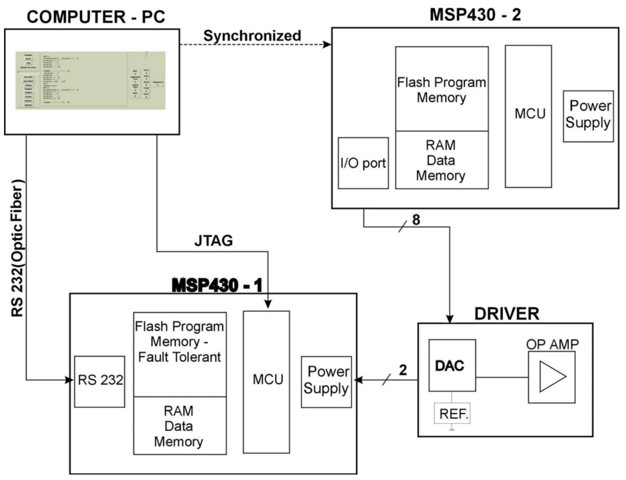

Fig. 7. Test setup used for electromagnetic-induced noise injection[2].

For this table, we have injected 85 voltage dips as seen in Fig. 6, for 5 different pulse time durations (10ms, 30ms, 100ms, 300ms, and 1s), in a total of 425 voltage dips injected in the DUT. The results displayed in this Table II can be interpreted as follows: for instance, for the 85 *0.01ms time duration voltage dips*, *36 functional errors* were verified. From this number, 63.89% were detected by the SW-based techniques described in the previous section, and XXX by the watch-dog-timer (WDT), i.e., by hardware. From the 63.89% of the faults detected, 82.61% were errors in *data*, while 17.39% were faults in the *control flow* of the processor. Figs. 8, 9, and 10 display individual results extracted from Table II.

Table III and Figs. 11, 12, and 13 summarize the same type of information shown in Table II and Figs. 8, 9, and 10. However, these results were obtained for a *Bubble Sort* Program running as the application in the DUT in the MSP430-1 Board. This Bubble Sort Program performs data reordering in a 238-databyte array.

Discussions:

As the reader can observe, Table II and Figs. 8, 9, and 10 (say *Test Set I*) indicate clearly the results pointing to a larger number of errors in the data used by the application than the number of errors in the control flow of the processor. This situation is exactly the opposite, when compared with the results presented in Table III and Figs. 11, 12 and 13 (say *Test Set II*). In this latter case, the number of control flow errors is dominant with respect to the one of data errors. This occurrence of data- or control flow-error dominance seen in *Test Sets I* and *II* can be explained by the profile of the application programs: while the *Image Processing* Program is data dominant in comparison to the number of instructions in the code, the *Bubble Sort* Program is control-dominant in the sense of several instructions executed on a compared reduced set of data.

Another conclusion taken from Figs. 10 and 13 is that the effectiveness of the control-flow error detection techniques is probably better than those used to protect data when conducted electromagnetic interference (Conduced-EMI)-induced disruptions are considered. However, this conclusion needs a deeper analysis with complementary practical tasks to clear some not yet answered questions: *Was there any fault generated in memory that could affect control-flow or data and that this error was generated after the processor had executed that part of the code*? If so, note that this (or these) fault(s) was (were) not detected by the fault detection strategies, but it (they) did account to increase the number of (latent) errors since it (they) would be propagated to primary outputs (PO) sometime in the future program runs. (Note that errors are detected by reading out the whole 2k-RAM memory containing data and control flow variables for the processor and by comparing them with a reference (*gold*) memory setup obtained by a fault-free simulation run).

Another important conclusion should be taken by observing that the average fault detection rate for the SW-based techniques is 70.63% (*Image Processing* Program) and 84.23% (*Bubble Sort* Program), as seen in Figs. 10 and 13, respectively. As consequence, the proposed SW-based techniques (and probably several other SW-based fault detection approaches found in the literature, since they present roughly similar principles [10,11]) do offer a *marginal protection level* to Conducted-EMI such as specified for *voltage dips* in the *IEC 61.000-4-29*.

On the other hand, after coupling the SW-based approaches with watch-dog-timer, the resulting average fault detection rate rises from 70.63% to 87.89% (Table II, Fig. 10) and from 84.23% to 90.10% (Table III, Fig. 13), which in both cases represent a considerable improvement of the detection rate.

Thus, having in mind these probability numbers, it is mandatory to carry out a more dedicated study about another complementary SW-based techniques to those presented in this work, if no HW-based fault detection is desired to be used. The possibility of proposing new fault-handling techniques, more adequate to detect conducted EMI-induced faults should also be strongly considered for future work. This solution is very important because the

[2] There are several professional EMI generators commercially available in the international market [11]. However, it should be noted that their typical prices could vary from two or three tens of thousands of dollars up to one hundred thousand dollars a unit. On the other hand, the test setup presented here is a simple, customized and economical conducted-EMI generator whose price turns around three thousand dollars a unit.

decision for using HW-based fault detection techniques could increase the whole system cost beyond acceptable values, or maybe because most of the existing COTS-based systems are not at all designed by considering fault detection based on dedicated HW structures.

		Detected Faults (%)			Not Detected Faults(%)
Duration (s)	Observed Errors	SW-Based Techniques (%)		HW-Based Technique (WDT)	
		% Data	% Control		
0,01	36	63,89		27,79	8,32
		82,61	17,39		
0,03	25	48		24	28
		100	0		
0,1	1	100		0	0
		100	0		
0,3	0	0		0	0
		0	0		
1	0	0		0	0
		0	0		

Table II. Fault Detection Summary for the 238-databyte Image Processing Program and Voltage Dips of -30% of the Nominal Vcc. (Nominal Vcc = 3.0V.)

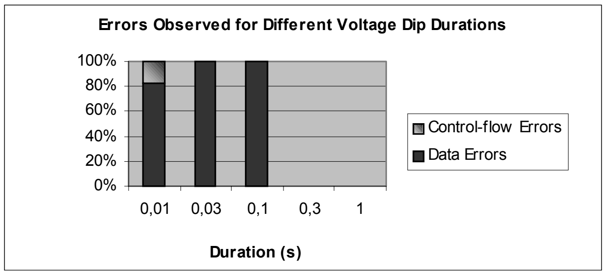

Fig. 8. "Data" *versus* "Control" Error Detection Summary. (238-databyte Image Processing Program. Voltage Dips of –30%.)

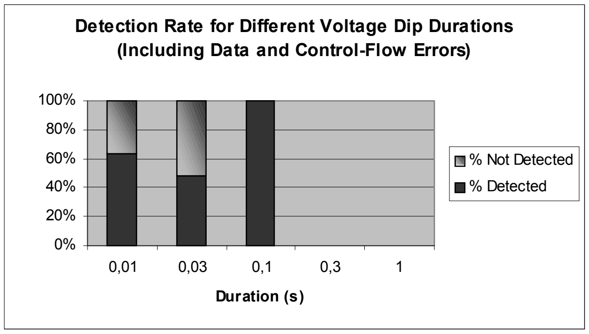

Fig. 9. Fault-Detection Capability Summary for the SW-Based Techniques. (238-databyte Image Processing Program. Voltage Dips of –30%.)

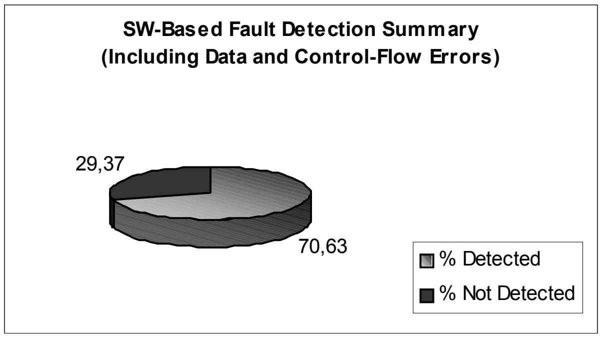

Fig. 10. SW-Based Fault Detection Summary. (238-databyte Image Processing Program. Voltage dips of –30%.)

Duration (s)	Observed Errors	Detected Faults (%)			Not Detected Faults(%)
		SW-Based Techniques (%)		HW-Based Technique (WDT)	
		% Data	% Control		
0,01	85	83,05		16,95	0
		39,8	60,2		
0,03	70	82,86		4,285	12,855
		32,76	67,24		
0,1	53	86,79		1,88	11,32
		21,74	78,26		
0,3	39	87,18		0	12,82
		20,59	79,41		
1	16	81,25		6,25	12,5
		15,38	84,62		

Table III. Fault-Detection Summary for the 238-databyte Bubble Sort Program and Voltage Dips of -30% of the Nominal Vcc. (Nominal Vcc = 3.0V.)

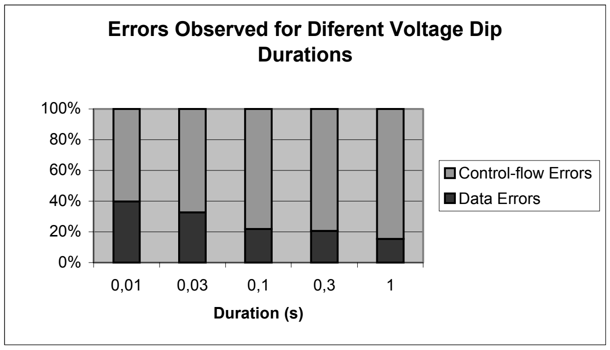

Fig. 11. “Data” *versus* “Control” Error Detection Summary. (238-databyte Bubble Sort Program. Voltage Dips of –30%.)

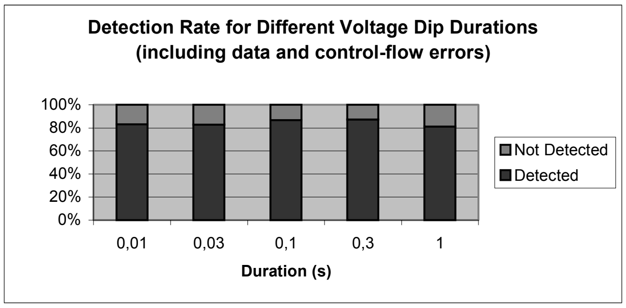

Fig. 12. Fault-Detection Capability Summary for the SW-Based Techniques. (238-databyte Bubble Sort Program. Voltage Dips of –30%.)

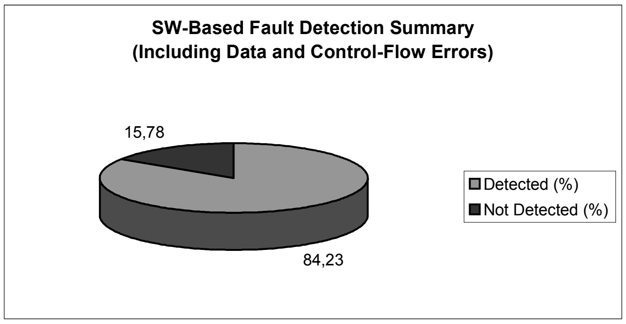

Fig. 13. SW-Based Fault Detection Summary. (238-databyte Bubble Sort Program. Voltage dips of –30%.)

5. Final Considerations & Conclusions

We have presented in this work a preliminary study to verify the capability of conventional fault-detection techniques to detect faults induced by conducted electromagnetic interference (Conducted-EMI). So far, these techniques have been commonly used to detect radiation-induced single-event upsets (SEUs) in memory elements and transient faults in logic.

The obtained results from practical experiments indicate to the direction of *marginal fault detection capability* due probably to:

a) the multiple generation nature as the primary effect of conducted-EMI on the processor power-bus;

b) the not compiler-visible control-flow variables used by the processor architecture, which cannot be accessed by the programmer and thus, cannot be modified in order to become EMI-hardened.

In this sense, more adequate or adapted fault coverage based uniquely on the use of SW-implemented fault detection techniques must be studied from the literature. Alternatively, new ones must be proposed in order to reach higher levels of fault coverage uniquely based on the use of SW-implemented fault detection techniques.

As the present work was involved only with voltage dips, voltage interruptions and voltage transients must also be studied in the future in order to attend the whole *IEC 61.000-4-29 International Standard Normative*. During this procedure, the processor must be exercised with different clock frequencies than the fixed 8MHz we have used in the present work.

References

[1] Steinecke, T. Design-In for EMC on CMOS large-Scale Integrated Circuits. International Symposium on Electromagnetic Compatibility – EMC'2001. Vol. 2 , 2001. pp. 910 –915.

[2] Perez, R. Signal Integrity Issues in ASIC and FPGA Design. International Symposium on Electromagnetic Compatibility – EMC'1997. 1997. pp. 334 -339.

[3] Whyman, N. L.; Dawson, J. F. Modelling RF Interference Effects in Integrated Circuits. International Symposium on Electromagnetic Compatibility – EMC'2001. Vol.2, 2001. pp. 1203 –1208.

[4] Tzimenakis, J.; Holland, D. Understanding the EMC Directive: Everything Made Clear. Gainspeed Ltd, UK, 2000. (jimtz@gainspeed.freeserve.com.uk)

[5] International Electrotechnical Commission - International Standard IEC 61000-4-29 Normative. (www.iec.ch)

[6] Fiori, F.; Benelli, Gaidano, G.; Pozzolo, V. Investigation on VLSIs' Input Ports Susceptibility to Conduct RF Interference. IEEE Transactions on Electromagnetic Compatibility, 1997. pp.326-329.

[7] www.polito.it/~sonza

[8] Rebaudengo, M.; Sonza Reorda, M.; Torchiano, M.; Violante, M. Soft-Error Detection Through Software Fault-Tolerance Techniques. IEEE Design for Testability Workshop (DFT'99), 1999.

[9] Microcontroller Texas MSP430F149. (www.ti.com)

[10] Miremadi, G.; Karlsson, J.; Gunneflo, U.; Torin, J. Two Software Techniques for On-Line Error Detection. IEEE Transactions on Electromagnetic Compatibility, 1992. pp.328-335.

[11] Miremadi, G.; Torin, J. Evaluating Processor-Behavior and Three Error-Detection Mechanisms Using Physical Fault-Injection. IEEE Transactions on Reliability. Vol. 44, No. 3, September 1995. pp. 441-454.

[12] Chillarege, R.; Bowen, N. S. Understanding Large Systems Failures – A Fault Injection Experiment. 19th International Symposium on Fault-Tolerant Computing - FTCS-19. 1989. pp. 356-363.

[13] Oh, N.; Shirvani, P. P.; McCluskey, E. J. Control-Flow Checking by Software Signatures. IEEE Transactions on Reliability. Vol. 51, No. 2, March 2002. pp. 111-122.

[14] www.cad.polito.it/cooperations/TOSCA/TOSCA.htm