Valderas, M. G., Garcia, M. P., Lopez, C. & Entrena, L. (2010, agosto). Extensive SEU Impact Analysis of a PIC Microprocessor for Selective Hardening. IEEE Transactions on Nuclear Science, 57(4), 1986-1991.

DOI: 10.1109/tns.2009.2039581

Extensive SEU impact analysis of a PIC microprocessor for selective hardening

Mario García Valderas, Marta Portela García, Celia López, Luis Entrena

***Abstract*—In order to increase the robustness of a circuit against SEUs, fault injection is commonly used to locate weak areas. Autonomous Emulation is a very powerful tool to locate these areas by executing huge fault injection campaigns. In this work, fault injection has been extensively applied to a PIC18 microprocessor, while executing three different workloads. A 80 million fault campaign has been performed, and results show that a failure rate lower than 1% can be obtained by hardening a 24% of the circuit flip-flops, for the given applications.**

***Index Terms*—SEU, Fault Injection, FPGA-emulation**

I. Introduction

TECHNOLOGY scaling increases notably the digital circuits' sensitiveness to environmental effects [1]. In particular, soft errors due to cosmic radiation effects are a main concern, not only in safety-critical applications but also in mainstream applications, even at earth surface [2]. In order to ensure the reliable operation of a circuit under soft errors, adding fault tolerant mechanisms is required. These mechanisms add redundancy, producing area overhead, increasing the power consumption and reducing the circuit performance [3]. A trade-off between reliability, cost and performance must be reached [7].

In the literature there are recent approaches to deal with soft error rate (SER) reduction in a cost effective way, reducing area overhead, power and performance penalties [4][5][6][7][8]. A solution to reduce the soft error rate, improving the reliability, in a cost effective way consists in performing selective hardening in early design stages [5], i.e., applying mitigation techniques just in some parts of the circuit during the design stage.

Evaluating the susceptibility to soft errors in order to indentify critical parts of a circuit is an essential task to perform selective hardening efficiently. Soft error sensitivity of a circuit depends on technological factors, functional features and on the workload.

In [9], a methodology to estimate the SER is presented. Technological factors are enclosed in the concepts of Nominal FIT (Failure in Time), which is related to the electrical characteristics of the technology, and Time Derating, which involves the times when the circuit is susceptible to SEU. Functional aspects are considered under the term Logic Derating, which is the probability of a SEU to impact the behaviour of the circuit. It depends on the circuit architecture and the workload. Fault injection is used to estimate logic derating.

In [10], technological aspects are measured through a static test, in which the circuit does not run a real application. This test is performed through radiation testing. A dynamic test is performed by checking the circuit behaviour under faults while the circuit is running a specific workload. Fault injection techniques are used to perform the dynamic test.

Fault injection is the most accepted solution by the scientific community to perform the dynamic test or to estimate logic derating. This paper is oriented in the same way. The behaviour of the circuit under faults is analysed from a functional point of view. By using fault injection, we are able to locate critical circuit areas so that they can be hardened before manufacturing the circuit. Once the circuit is hardened and manufactured, radiation testing will be required to check the result. This work is focused on the sensitivity analysis, and hardening and radiation testing fall outside the scope of the paper.

A method to locate weak circuit areas for selective hardening in digital circuits is described. A PIC18 microcontroller macrocell has been used as case study since microprocessors are typical circuits where penalties introduced by hardening techniques must be minimized. Three different workloads, with different nature, have been used to demonstrate that the critical parts of a circuit depend on the running workload.

The Autonomous Emulation system [11] has been used. This tool profits from hardware emulation on platform FPGAs, and provides a very high fault injection rate, allowing massive fault injection campaigns. It profits from hardware speed since all fault injection tasks, like controlling the injection campaign, observing the circuit behaviour to classify the faults or applying the stimuli, are performed in hardware. One of the advantages of the fault injection system hardware implementation is that some tasks can be paralleled, specially, in the case of calculation-based tasks.

This technique exploits the embedded hardware resources available in the FPGA platforms, like RAM memory blocks in order to implement the complete fault injection environment. Furthermore, Autonomous Emulation provides more observability and controllability than other proposals. These

Manuscript received April 3, 2009.

Authors are with Electronic Technology Department, University Carlos III of Madrid, Av. Universidad 30, 28911 Leganes, Madrid, Spain. Phone: +34 916246018, Fax: +34 916249430. Email: {mgvalder, mportela, celia, entrena}@ing.uc3m.es.

features are exploited to optimise the injection process and making it faster. Most approaches [9][12] rely on Statistical Fault Injection (SFI) to reduce the huge amount of possible faults to a representative reduced fault set.

Autonomous Emulation provides the following capabilities in a cost-effective way:

- Obtaining the accurate fault dictionary for every memory element in the circuit, for a given workload. Every possible SEU in a memory element can be injected and evaluated for a workload, without the need of selecting statistical samples.
- Analysing the weak areas of the circuit at first design stage. This allows the designer to study the trade-off between the applied hardening techniques and introduced penalties (in area, performance and power consumption). This analysis can be performed iteratively since the evaluation process is very fast.

The paper is organized as follows. Section 2 describes the experimental setup, including the autonomous emulation system built over the PIC18 circuit and the different workloads. Section 3 shows the results obtained and a thorough discussion. Finally, section 4 states some conclusions.

II. Experimental setup

As a case study, an in-house developed PIC18 macrocell running three different workloads has been tested. An autonomous fault emulation system [11] has been built.

A. Autonomous Fault Emulation

Autonomous emulation is an automatic tool to perform fault injection analysis on circuit descriptions. Fault injection is done according to the commonly accepted fault model *bit-flip,* corresponding to SEU effect on memory elements into digital integrated circuits. This injection is executed directly on circuit memory elements prototyped on a programmable device (FPGA).

Although the final implementation will be an ASIC, memory elements are equivalent in an FPGA prototyped version. Little differences could be caused by the use of different synthesisers for FPGA and ASIC libraries. If an exact result is required, the circuit should be compiled for the ASIC technology and every gate and flip-flop replaced for a behavioural equivalent FPGA model.

In Autonomous emulation system, a modified (*instrumented*) version of the circuit is prototyped on a FPGA platform along with a module in charge of performing the whole fault injection campaign.

The main advantage of Autonomous Emulation approach is the capability to abort the emulation once a fault is classified as failure or silent, restoring the circuit state to the previous clock cycle to the injection of the fault classified. With this optimisation, few clock cycles are employed per fault emulation and high fault injection speeds are achieved.

The Autonomous Emulation system built for this work includes the synthesized and instrumented PIC18 macrocell, the program memory, a fault classification mechanism and the emulation control unit.

B. PIC18 microprocessor

PIC18 is an 8-bit, high performance microprocessor, well suited for embedded applications. It is a RISC processor with Harvard architecture and a 2-stage pipeline (fetch and execution) [13].

Commercial implementations offer up to 128kB of ROM memory for program and data, and up to 4kB of RAM memory.

The used PIC18 IP has been developed to build microprocessor systems with certain computational power and to be able to include new peripherals. Peripherals are attached to the data bus and can be controllers by mapping special function registers or by replacing a RAM memory bank and mapping the peripheral in RAM space address.

The developed IP can be configured in terms of ROM memory, RAM memory and the selection of peripherals.

The core blocks include:

- Instruction fetch and decode,
- Program counter and associated registers and the program counter stack (32 addresses).
- Program Memory (ROM)
- User Memory (RAM) and related logic, like addressing logic (block select, indirect addressing).
- ALU: arithmetic and logic functions, the working register (WREG) and a multiplier.

Developed peripherals include:

- Input-output ports (5): PortA to PortD.
- Timers (4): Timer0 to Timer3
- UART: asynchronous serial communications.

The setup chosen for the experiment include PortB and PortD, the timers and the UART.

In order to build the autonomous emulation system for fault injection, the PIC18 macrocell has been synthesized for a Virtex-4 FPGA family. Synthesis results are shown in Table 1~~Table 1~~, showing the number of flip-flops, LUTs (Look-Up Tables) and RAM memory blocks. The design includes RAM blocks for user RAM and PC stack. Program memory has been left out of the design, so that the program can be easily changed without the need of rebuilding the emulation system. In commercial PICs, program memory is FLASH, and thus not sensitive to SEU. If program memory is implemented using RAM, it should be protected using EDAC codes.

Table 1. PIC18 synthesis results

Virtex-4	Used
Flip-flops	596
LUTs	2,314
Block-RAMs	7

Results are written to port B and port D. Writing to a port a result which is different than expected or writing it in a not exact clock cycle will result on the fault being classified as a failure.

C. Workloads

Three different applications have been used to test the microprocessor: a matrix multiplication, a serial data transmission using the UART and a real time clock.

The matrix multiplication is a data intensive application, which spends most of the time moving data from memory to the ALU and back, and making arithmetical operations. In this application, the ALU is the most used component.

For the data transmission program, the microprocessor TX and RX signal are shorted, so that it receives through RX the same data sent through TX. The program consists in a continuous flow of characters going through the serial port. The UART is the most used component in this application.

The real time clock application uses a timer to maintain the system clock, which is a very important task in any space application. The timer is set to trigger a periodic interrupt. The interrupt service routine increments a tick counter, and several functions are used to update the system date and time. In this case, the most critical blocks are the timer used and the interrupt handler block.

The three applications write partial data to the microprocessor ports in order to check the correct executions of the programs.

III. Experimental Results and Discussion

An intensive fault injection campaign has been performed for three workloads. A single fault has been injected in every circuit flip-flop, and in every clock cycle of every workload. Only some clock cycles at the beginning have been left out. Table 2~~Table 2~~ shows the characteristics of the three fault injection campaigns. Tables 3 to 5 show the grouped fault classifications for the three workloads, and Table 6~~Table 6~~ shows a failure rate comparative between the three applications.

Table 2. Fault injection characteristics

	Workload clock cycles	Total injected faults
Matrix multiplication	55,784	33,414,616
Serial communication	28,963	17,213,076
Real Time Clock	51,103	29,693,316

Table 3. Grouped Fault classification: matrix multiplication

	#FF	Failure	Silent	Latent	RAM Latent
Pipeline	2	72.768	0	177	38.623
ALU	42	93.675	223.110	728.432	1.297.711
Data&Addr	52	163.073	0	1.440.199	1.297.496
Instructions	104	1.028.126	55.936	2.036.476	2.680.998
Ports	48	446.272	0	2.228.432	2.928
Interrupts	79	0	534	4.294.834	111.568
Timers	152	0	446.272	7.196.270	836.626
UART	117	0	892.544	5.522.616	111.568
Total	596	1.803.914	1.618.396	23.447.436	6.377.518
		5,43%	4,87%	70,52%	19,18%

Table 4. Grouped Fault classification: serial communication

	#FF	Failure	Silent	Latent	RAM Latent
Pipeline	2	39,327	17,507	472	456
ALU	42	5,753	369,365	837,690	194
Data&Addr	52	423,956	297,752	714,980	65,124
Instructions	104	347,010	1,068,999	1,332,515	255,100
Ports	48	231,048	0	932,680	222,560
Interrupts	79	4,134	52,702	2,223,603	1,160
Timers	152	0	635,226	3,754,686	0
UART	117	1,305,026	1,538,300	534,603	1,148
Total	596	2,356,254	3,979,851	10,331,229	545,742
		13.69%	23.12%	60.02%	3.17%

Table 5. Grouped Fault classification: real time clock

	#FF	Failure	Silent	Latent	RAM Latent
Pipeline	2	59,569	29,433	4,993	5,647
ALU	42	56,072	748,115	817,532	470,763
Data&Addr	52	1,107,135	576,611	210,094	696,852
Instructions	104	1,210,106	1,296,209	1,312,461	1,362,608
Ports	48	398,568	189,768	1,608,744	194,328
Interrupts	79	221,431	118,943	3,519,434	76,051
Timers	152	936,260	982,425	5,021,772	632,335
UART	117	0	844,578	4,932,279	52,200
Total	596	3,989,141	4,786,082	17,427,309	3,490,784
		13.43%	16.12%	58.69%	11.76%

Table 6. Failure rates comparative

	#FF	Matrix	UART	RTC
Pipeline	2	0.22%	0.23%	0.20%
ALU	42	0.28%	0.03%	0.19%
Data&Addr	52	0.49%	2.46%	3.73%
Instructions	104	3.09%	2.02%	4.08%
Ports	48	1.34%	1.34%	1.34%
Interrupts	79	0.00%	0.02%	0.75%
Timers	152	0.00%	0.00%	3.15%
UART	117	0.00%	7.58%	0.00%
Total	596	5.43%	13.69%	13.43%

In these tables, a fault is considered silent when the fault injection effect disappears completely from the circuit. A failure indicates the fault has been propagated to the microprocessor ports. Latent faults indicate that at the end of the program execution some flip-flop values are different than the ones in the fault free circuit, but the fault has not been propagated to the ports. RAM latent faults are like latent faults, but the fault has propagated into RAM blocks.

These results give a lot of significant information about the circuit. There are two flip-flops that control the pipeline flow, which produce a failure in a very high amount of faults, and should be protected.

The matrix multiplication program is data intensive, and produces a higher amount of failures in the ALU. The communication program produces more failures when faults are injected in the UART, and the real time clock application shows more failures in the used timer and interrupt treatment blocks.

Ports show a high failure rate in any application because they are used to check the results. The rate would be lower with the real applications.

The real time clock application fails quite more in the addressing (Data&Addr) and instruction blocks. The addressing block fails because in this application there are

frequent subroutine calls, and context storage and restoring force the use of indirect addressing. The instruction block fails more because the program is more complex and uses a higher number of different instructions.

However, the most powerful ability of autonomous emulation is to obtain the fault classification in a per flip-flop basis. It allows selecting the critical sections of the circuit for hardening. In the tables, it can be noticed that most of the faults producing failures are related with instructions (program counter and instruction register), with microprocessor ports, and with UART and timers in the applications which use them.

A deeper analysis is presented in Fig. 1, 2 and 3, which graphically shows the fault classification of every flip-flop. Every horizontal line represents a different flip-flop. The whole line is the total number of faults injected in the flip-flop (one for each clock cycle).

In these figures, failures are colored in red (dark), so that critical flip-flops can be easily identified. This way, it can be noticed that the most critical flip-flops in the design are those related with ports, the program counter and specific parts used by each application, like the ALU, UART, timers and interrupts.

Port related flip-flops always produce failures in the three applications, because values at ports are directly used to determine the classification of faults. In real applications, ports are usually critical parts of the circuit, given that they propagate faults very easily.

The program counter also produces a high amount of failures, but not all the bits. Only the least significant 12 bits (out of 20) produce failures. The rest produces latent faults. The explanation is the design of the program memory. Instead of reading zeros when accessing an empty memory position, the program has been replicated throughout all the memory space. This way, high bits on the program counter do not affect the program sequence when affected by SEUs.

There are also other flip-flops related to the program counter that throw a lot of failures, such as the Top Of Stack register (TOS). It stores the returning address of a subprogram or interrupt. Only the first 12 bits are critical, the same as the program counter and for the same reason.

ALU related registers are not very critical in these applications. These registers are the working register (WREG) the status register (STATUS) and two registers used to store multiplication results (PRODH and PRODL). In general, these registers do not produce many failures. In the matrix multiplication application (Fig. 1), the failure rate is higher because it is data intensive and the ALU is used more often. The working register is going to be critical, independently of the application, because it is used very often for all sorts of operations.

Other group of registers that produce many failures are indirect addressing registers, FSR0 to FSR2 (corresponding to Data&Addr section in tables). These are the pointers for indirect addressing, so when they fail, data are handled incorrectly. In the matrix multiplication application, there is very little indirect addressing, and the failure rate is lower.

In the communications and real time clock applications, the failure rate due to indirect addressing is higher. In the communications application, the variable that controls the characters sent through the serial port is accessed using FSR2, and produces failures when characters different than expected are sent. In the real time clock application there are a lot of subroutine calls and interrupts, which produce a lot of context saving and restoring. Context variables use indirect addressing, so a lot of failures are produced.

The UART and timers produce failures just on the applications that use them. The UART is a big block composed mainly by counters, and fault injection produces the timing to fail when the UART is in use (Fig. 2). Something similar happens to timers. The real time clock application uses one of the four microprocessor timers and produces a lot of failures when injecting in that timer (Fig. 3).

Figures also show the parts of the circuit that are not used. Unused parts produce latent faults. The fault is stored and it does not produce a failure because it is never used and it cannot disappear because it is never overwritten, so it remains latent forever. This is the case of the interrupt controller, PORTA and unused timers.

From this analysis, several conclusions can be obtained. On the one side, there are some microprocessor areas that should be protected with independence of the application, because they are critical for the microprocessor correct flow. On the other side, there are some parts of the circuit that should be hardened according to the workload. For the tested workloads, it can be deduced that hardening a 24% of the flip-flops (using TMR in a preliminary approach) the failure rate obtained is lower than 1%.

This analysis shows the possibilities that the Autonomous Emulation fault injection system can offer. The fault classification heavily depends on the application used as workload, and different hardening decisions can be taken for the same circuit but different applications. Autonomous Emulation is a very powerful tool used to locate weak areas of a circuit, taking into account the workload executed.

Selectively hardening a design, in relation to the particular application it is going to run, may result in a lower area overhead, compared with a full hardening approach, while maintaining a great reduction in the failure rate.

IV. Conclusions

Applying hardening techniques to a whole circuit can be a waste of resources, if we consider the particular use of the circuit in the final application. Microprocessor cores are general purpose circuits, and the usage of its resources depends heavily on the workload. There are usually some parts of the circuit that are seldom or not used at all and extra resources to harden them would be wasted.

In this paper, we have used the autonomous emulation fault injection system to perform a massive fault injection campaign in a PIC18 microprocessor, running three different

applications, a matrix multiplication, serial communications and a real time clock.

Results show that some parts of the circuit are critical for any application, and others that should be protected only if they are used. Given a workload, weak circuit areas can be easily located thanks to Autonomous Emulation.

REFERENCES

[1] International Technology Roadmap for Semiconductors, Editions 2001-2007.

[2] R. C. Baumann, "Radiation-Induced Soft Errors in Advanced Semiconductor Technologies", IEEE Trans. On Device and Materials Reliability, Vol. 5, No. 3, pp. 305-316, September 2005.

[3] M. Nicolaidis, "Design for soft error mitigation" IEEE Trans. On Device and Materials Reliability, Vol. 5, No. 3, pp. 405-418, 2005.

[4] D. Nowroth, I. Polian, B. Becker, "A Study of Cognitive Resilience in a JPEG Compressor" Int. Conference on Dependable Systems & Networks, pp. 32-41, June 2008.

[5] C.G. Zoellin, H. J. Wunderlich, I. Polian, B. Becker, "Selective Hardening in Early Design Steps" 13th European Test Symposium, pp. 185-190, 2008.

[6] M. Zhang, S. Mitra, T. M. Mak, N. Seifert, N. J. Wang, Q. Shi, K. S. Kim, N. R, Shanbhag, S. J. Patel, "Sequential Element Design with Built-In Soft Error Resilience" IEEE Trans. On Very Large Scale Integration Systems, Vol. 14, No. 12, December 2006.

[7] K. Mohanram, N. A. Touba, "Cost-Effective Approach for Reducing Soft Error Failure Rate in Logic Circuits" International Test Conference, pp. 893-901, 2003.

[8] Mojtaba Mehrara, Mona Attariyan, Smitha Shyam, Kypros Constantinides, Valeria Bertacco, Todd Austin, "Low-cost Protection for SER Upsets and Silicon Defects", Design Automation and Test in Europe (DATE), Nice, France, April 2007.

[9] Hang T. Nguyen, Yoad Yagil, Norbert Seifert, Mike Reitsma, "Chip-Level Soft Error Estimation Method", IEEE Transaction On Device And Materials Reliability, Vol. 5, No. 3, September 2005.

[10] S. Rezgui, R. Velazco, R. Ecoffet, S. Rodriguez, J. R. Mingo "Estimating error rates in processor-based architectures" IEEE Transactions on Nuclear Science, Vol. 48, Issue 5, pp. 1680-1687, Octubre, 2001.

[11] C. López-Ongil, M. García-Valderas, M. Portela-Garcia, L. Entrena, "Autonomous Fault Emulation: A New FPGA-Based Acceleration System for Hardness Evaluation", Trans. on Nuclear Science, Vol. 54, No. 1, pp. 252-261, February, 2001.

[12] Jean-Marc Daveau, Alexandre Blampey, Gilles Gasiot, Joseph Bulone, Philippe Roche, "An Industrial Fault Injection Platform for Soft-Error Dependability Analysis and Hardening of Complex System-On-a-Chip", IEEE CFP09RPS-CDR 47th Annual International Reliability Physics Symposium, Montreal, 2009.

[13] "PICmicro® 18C MCU Family. Reference Manual", Microchip Technology Inc., ref. DS39500A, 2000.

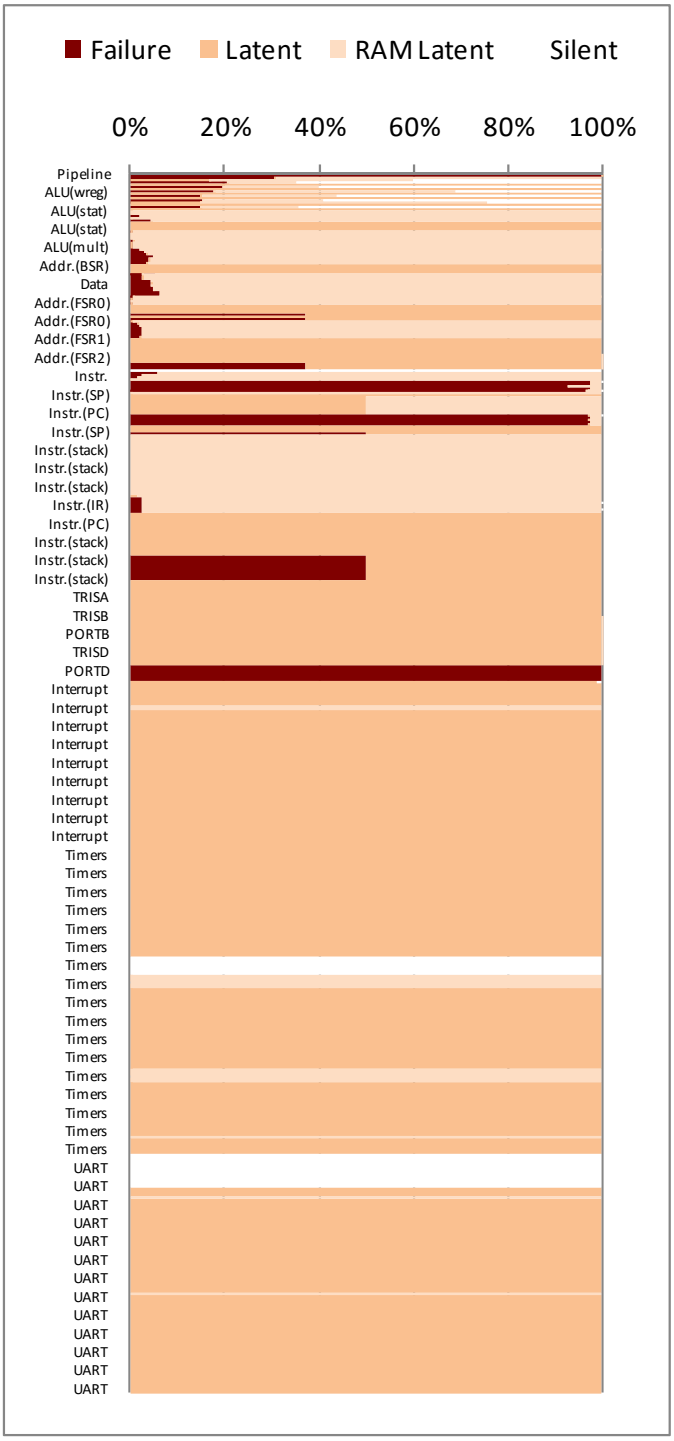

Fig. 1. Per flip-flop fault classification: matrix multiplication

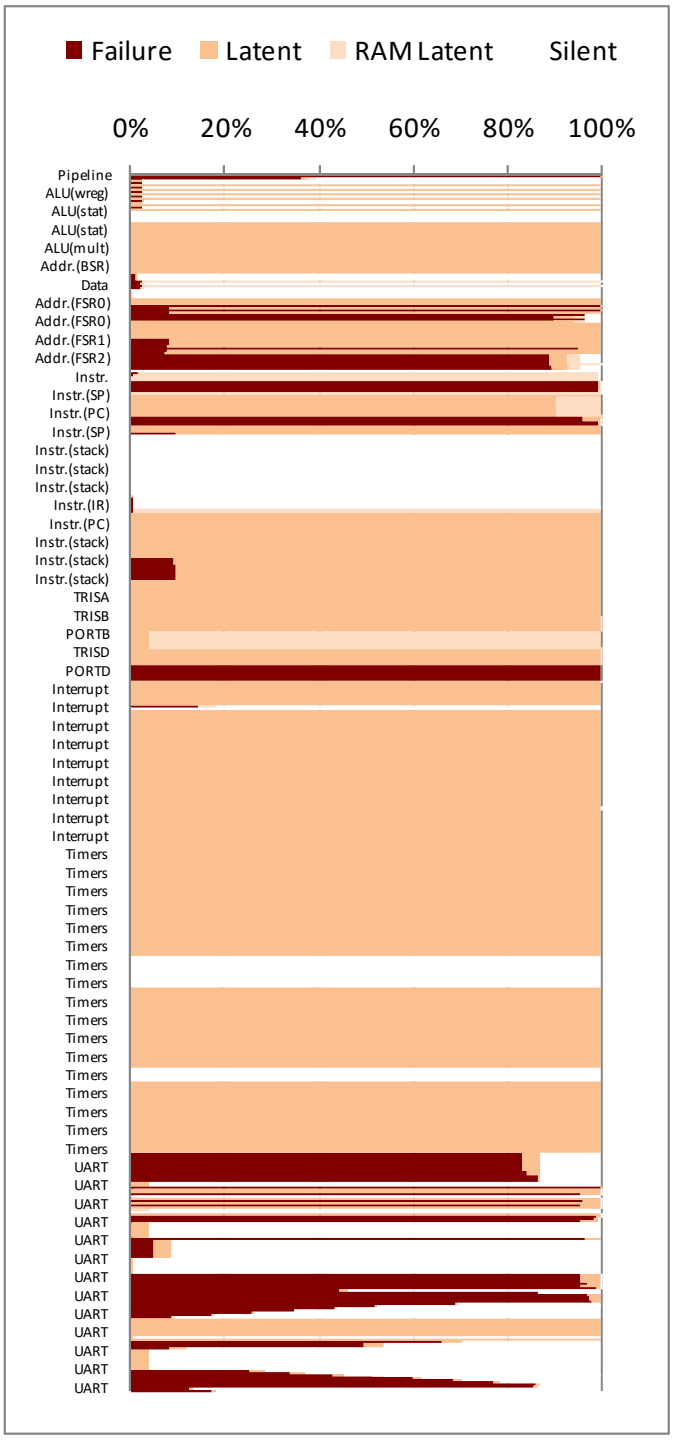

Fig. 2. Per flip-flop fault classification: serial communication

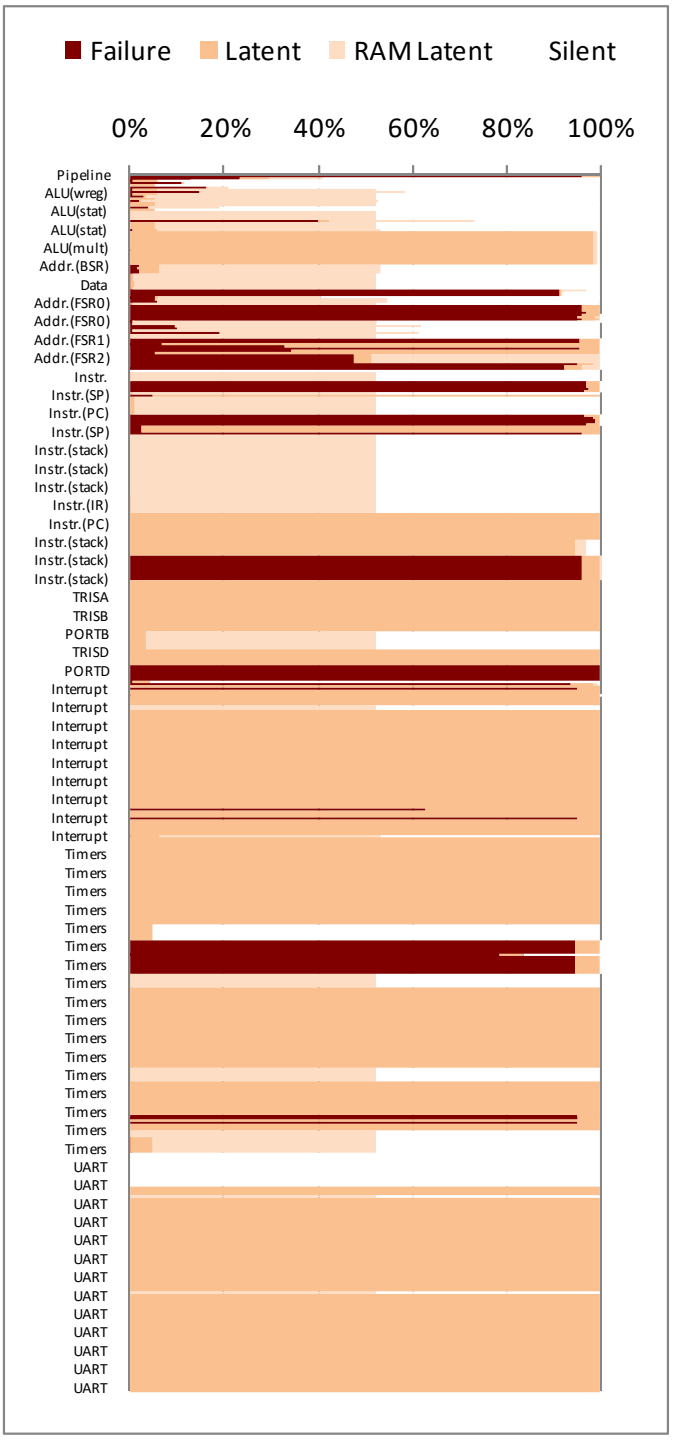

Fig. 3. Per flip-flop fault classification: real time clock