Single event upset failure probability evaluation and periodic scrubbing techniques for hierarchical parallel vision processors

Qian Di\(^1,2\), Zhongxing Zhang\(^1,2\), Honglong Li\(^1,2\), Zhao Zhang\(^1,2\), Peng Feng\(^1,2,a\), and Nanjian Wu\(^1,2,3\)

Abstract This paper proposes novel single event upset (SEU) failure probability evaluation and periodic scrubbing techniques for hierarchical parallel vision processors. To automatically evaluate the SEU failure probability and identify all the critical elements in a processor, complementary fault injection methods based on logic circuit simulator and Perl script are proposed. These methods can be used to randomly inject faults into D flip-flops (DFFs) and various types of memory at the register transfer level (RTL) as well as to evaluate the vision processor performance. Based on the evaluation results, an accurate periodic scrubbing technique is proposed to increase the processor availability. The results denote that the peak availability of the processor over a period of one year can be improved from 18% to 99.9% after scrubbing the RISC program memory for a period of \(10^4\) s. Therefore, we can improve the fault-tolerance performance of a vision processor while avoiding unnecessary area and power costs using techniques ranging from evaluation to mitigation.

key words: single event upset, vision processor, fault injection, periodic scrubbing, availability
Classification: Integrated circuits

1. Introduction

A hierarchical parallel vision processor is a device that integrates multiple levels of processors exhibiting different parallelisms and complexities. Such processors can be extensively applied in areas, including industrial automation and security monitoring [1–6]. With the rapid growth of computation requirements in space image-processing missions [7–9], vision processors exhibit excellent prospects for performing various image-processing tasks. However, the single event upset (SEU) effect can change the content of the storage elements in advanced CMOS technology. A bit-flip in the storage elements, including D flip-flops (DFFs) and various types of memory, has different impacts on the vision processor performance. Directly hardening the vision processor via mitigation techniques [10–12] will lead to an extra chip area and power consumption. So we should initially evaluate the SEU sensitivity of the storage elements and identify the critical elements. Further, we should adopt mitigation techniques to reduce the SEU effect on these critical elements. To the best of our knowledge, similar studies that identify the critical elements and mitigate the SEU effect on vision processors have not yet been reported.

Various fault injection methods, including FPGA-based [13–15], software-based [16, 17], and simulation-based methods [18–20], have been proposed to evaluate the SEU sensitivity of the storage elements. Even though the FPGA-based methods exhibit fast emulation speeds, the scale of the evaluated circuit is limited by the FPGA resources, and the synthesis process is time consuming. The software-based methods have the advantage of evaluating multiple faults with high flexibility. The simulation-based methods have the advantages of better controllability and observability. However, with an increase in the chip scale and complexity, we cannot achieve full coverage using any of the aforementioned single methods. Therefore, this paper focuses on complementary software methods to achieve full-coverage fault injection into a vision processor.

A periodic scrubbing technique has been proposed to mitigate the SEU effect and improve the processor availability by periodically repairing the data in memory [21–23]. The availability of the Virtex device XC3V300 can reach 0.9963 at the scrub period of 60 s [22]. It was considered that a processor would fail once an SEU event occurred. However, not every SEU event will cause processor failure. That is, existing periodic scrubbing studies have not provided an accurate calculation model for optimizing the processor availability. This paper proposes novel SEU failure probability evaluation and periodic scrubbing techniques for hierarchical parallel vision processors. The methods for performing SEU failure probability evaluation include a non-intrusive fault injection method for DFF (NIFID), a non-intrusive and accumulated fault injection method for the long-term stored memory (NAFIM), and an automatic fault injection method for the temporarily stored memory (AFITM), which are used to evaluate different types of storage elements. These com-
plementary methods achieve full coverage and automatically identify all the critical elements in a vision processor. Based on the simulation results, an accurate periodic scrubbing technique is proposed to improve the processor availability. This accurate periodic scrubbing technique is suitable for the long-term stored memory and can increase the peak availability of the processor from 18% to 99.9%. Using techniques from evaluation to mitigation, we can improve the fault-tolerance performance of a vision processor while avoiding unnecessary area and power costs.

2. Vision processor

Fig. 1 depicts the architecture of a hierarchical parallel vision processor. It comprises a processing array (PA), RISC32 MPU, and peripheral modules. The PA consists of 4×4 patch processors, which contain a patch processing unit (PPU), a local data memory, and 32 pixel-parallel process elements (PEs). The RISC32 MPU manages the overall vision processor and is involved in some global computing tasks. The peripheral modules include RISC program memory for storing MPU instructions, VMIPS program memory for storing the algorithm instructions, input/output (I/O) memory for transferring data, a VMIPS controller, and so on.

The PA and RISC32 MPU are the main functional modules. The circuit of the patch processor in the PA is depicted in Fig. 2. In the patch processor, the PPU primarily comprises an instruction decoder, a register file, an ALU, and multiplexers. The decoder decodes the instruction and generates control signals for the PPU and PEs, such as the write enable signal called WriteReg and the destination register selection signal called RegDst. Each PE comprises a register file, an ALU, and multiplexers. The local data memory is used to store the image data, network weights, and intermediate results. The RISC32 MPU is a five-stage pipelined MIPS processor [24] which also primarily comprises an instruction decoder, a register file, an ALU, and multiplexers.

The vision processor supports various algorithms [25–28]. The utilization rates and behaviors of the modules are dependent on the algorithms. Thus, a Sobel-based classification algorithm and a convolutional neural network (CNN)-based classification algorithm [28] were selected to perform the fault injection experiment. In the Sobel-based classification algorithm, the total sum of the image gradient is calculated via a Sobel descriptor. Further, the classification is based on whether this sum is within the default threshold range. In the CNN-based classification algorithm, image features are extracted through alternating arranged convolutional and pooling layer, which mainly contains multiply-accumulation (MAC) operation. The classification is based on the confidence result given by the output layer.

3. Fault injection methods

3.1 NIFID for DFFs

The NIFID method is proposed to evaluate the SEU sensitivity of the DFFs in the vision processor. Based on the logic circuit simulator and the designed Perl script, random faults are automatically injected at the register transfer level (RTL) with non-intrusive of Verilog HDL code. Therefore, the circuit scale is not increased, and the time consumption on modifying and verifying the HDL code is saved. Furthermore, this method does not depend on a specific simulator; therefore, a high-speed simulator can be selected for the RTL simulation. Accordingly, this method achieves high-speed fault injection and evaluation.

Fig. 3 denotes the process of the proposed NIFID method. It contains a setup phase, a simulation phase, and an evaluation phase. In the setup phase, first, the name and the entire path of all the DFFs are initially extracted and stored in a file called the "register library". Second, the injection parameters, which include the start time of injection, the injected DFF, and the workload, should be determined. The start time of injection can be randomly selected during the
entire simulation process, and the fault of the injected DFF is removed at the next rising edge of clock. The injected DFF is randomly selected from the register library. The workload is the algorithm. Third, the Perl script for fault injection is designed to update the injection parameters and change the test-bench using these parameters. It controls the automatic experimental process.

In the simulation phase, two types of files called the golden run file and the faulty run files are created. The golden run file and the faulty run files store the simulation results before and after injecting random faults, respectively. In the evaluation phase, every faulty run file is compared with the golden run file and the comparison results are analyzed and recorded. The processor failure probability can be obtained via thousands of repeated simulations. In addition, critical DFFs can be identified using the statistical results.

3.2 NAFIM for long-term stored memories

The NAFIM method is proposed to inject accumulated faults and evaluate the SEU sensitivity of long-term stored memories, including the RISC program memory, the VMIPS program memory, and the data memory storing the weights. Based on the logic circuit simulator and the designed Perl scripts, randomly accumulated faults are successively injected into the data file prior to the simulation. The same as the NIFID, this method achieves a high-speed fault injection with non-intrusive of the HDL source code.

Fig. 4. Process of the proposed NAFIM method.

Fig. 4 denotes the process of the proposed NAFIM. It also contains a setup phase, a simulation phase, and an evaluation phase. In the setup phase, the data file (Mem.dat) stores the original binary instructions. The injection parameters include the injected bits and workload. The injected bits are randomly selected from Mem.dat. The Perl script for fault injection is responsible for updating the injection parameters and changing Mem.dat using the injected bits to generate an updated binary instruction file (updated mem.dat) prior to the simulation. It enables the fault injection experiment to run automatically.

In the simulation phase, the golden run file and the faulty run files are successively generated. In the evaluation phase, the faulty run file is compared with the golden run file after each simulation. If the result in the faulty run file is the same as that in the golden run file, the accumulated fault injection script continues to inject faults into the memory and repeat the simulation and evaluation phases until the comparison result is different. The total number of accumulated faults that cause a processor failure is recoded. The relation between the processor failure probability and the number of accumulated injection faults can be obtained using thousands of repetitions of the above process.

3.3 AFITM for temporarily stored memory

There are temporarily stored memories in the vision processor including the register files, the I/O memory, and data memory storing the image data and results. To achieve full coverage of the vision processor, we propose the AFITM method for such memories, as depicted in Fig. 5. Based on the designed Perl script and an extra monitor module, random faults are automatically injected.

The Perl script for fault injection randomly selects the start time of injection, the injected address (RandAddr[8:0]), and the injected bit (RandBit[7:0]). It also enables the fault injection experiment to run automatically. The injection module monitors the memory address. Once the data memory performs a read out operation and the address (MemAddr[8:0]) matches the injected address, the corresponding bit of the read data is flipped. The fault injection process is completed when the data stored in the injected address is rewritten. Via thousands of repeated simulations, the processor failure probability can be obtained.

4. Simulation results of fault injection

4.1 Simulation results and analysis of DFF

DFFs are distributed in the PPU, the PEs, the RISC32 MPU, and the peripheral modules. 5000 DFF fault injection experiments were performed for each function module, the number of which is much larger than the number of DFFs in any module. In each experiment, one random fault was injected into the DFF using the NIFID method. The algorithms of the Sobel- and CNN-based classifications were implemented on the vision processor. The failure probabilities and main critical DFFs that obtained from the experiments are presented in Table I.

In this table, the S_Ctrl, S_IF, S_ID, S_EX, and S_MA modules of the PPU are responsible for generating global control signals, program counting and fetching instructions, decoding instructions and accessing register files, executing, accessing data memory and writing data back, respectively. The functions of the modules in the PPU are the same as that in the RISC32 MPU. The naming rule and functions of the critical DFFs in the PPU and MPU are based on those
Table I. Fault injection results for the DFFs.

| Module | Num. DFFs | Failure Probability(%) | Critical DFFs | Failure Probability(%) | Critical DFFs |
|--------|-----------|------------------------|--------------|------------------------|--------------|
| PPU    |           |                        |              |                        |              |
| S_Ctrl | 10        | 8.1                    | Hold, Flush  | 37.1                   | Hold, Flush  |
| S_IF   | 114       | 1.5                    | IFID_Instr, PC[15:5] | 6.1                   | IFID_Instr, PC |
| S_MA   | 21        | 1.1                    | MAWB_DataToReg/_RegDst | 2.7                  | MAWB_WriteReg/_DataToReg/_RegDst |
| S_EX   | 97        | 0.5                    | EXMA_ALUResult, RsData | 1.7                | EXMA_ALUResult, RsData |
| S_ID   | 156       | 0.4                    | IDEX_Instr, IDEX_RsData | 1.3                | IDEX_Instr, IDEX_RsData |
| RISC32 MPU |       |                        |              |                        |              |
| R_CTRL | 12        | 28.0                   | Hold, Flush  | 24.3                   | Hold, Flush  |
| R_IF   | 97        | 7.5                    | IFID_Instr, PC | 13.1                  | IFID_Instr, PC |
| R_MA   | 159       | 1.4                    | MAWB_DataToReg/_RegDst | 0.8                | MAWB_DataToReg/_RegDst |
| R_EX   | 356       | 1.4                    | RsData, EX_NewPC | 2.9                 | RsData, EXMA_RegDst |
| R_ID   | 257       | 4.3                    | IDEX_RsData  | 3.9                    | IDEX_RsData  |
| PEs    | 6432      | 0                      | -            | 0                     | -            |
| PERIPHERAL MODULES |       |                        |              |                        |              |
| VMIPS_Ctrl | 118     | 0.6                    | PCConfigure  | 4.1                    | PCConfigure  |
| mAHB   | 48        | 1.6                    | SlaveSelect, MasterReadData | 3.4         | SlaveSelect, MasterReadData |

In the standard MIPS processor [24], the PEs also contain multiple function modules, of which the total number of DFFs is 6432. And the number of DFFs in the instruction execution module is the largest, which equals to 112. The failure probabilities caused by bit-flips in the PPU and RISC32 MPU are higher than that in PEs. The PPU is responsible for decoding and executing algorithm instructions; therefore, bit-flips of the corresponding DFFs will directly affect the algorithm flow. Further, the critical DFFs and failure probabilities in the PPU are different for the two algorithms because the SEU sensitivities of the DFFs in the PPU are dependent on the algorithm instructions. The RISC32 MPU manages the operation of a vision processor; therefore, bit-flips of the DFFs in the MPU are more likely to cause a processor failure. The bit-flips of the DFFs in the PEs do not introduce errors to a vision processor. This is because the PEs are only responsible for logical or arithmetic operations and one flip usually causes a calculation error of one pixel, which will barely affect the final result.

In peripheral modules, PCConfigure in the VMIPS_Ctrl module is responsible for configuring the program counter. A bit-flip will directly affect the algorithm. SlaveSelect in the mAHB module is responsible for selecting the other modules. A bit-flip will cause inaccurate operation. Therefore, some modules in the peripheral modules should also be protected.

From the simulation results, design guidance can be provided for a radiation-hardened vision processor. Critical DFFs should be protected via mitigation techniques such as triple modular redundancy (TMR). The other DFFs that have negligible influence on the vision processor do not need to be protected. Via the selective protection of DFFs, vision processor is expected to operate more reliably in space with a small increase in chip area and power.

4.2 Simulation results and analysis of memory

To compare the SEU sensitivities of all the different types of memory, fault injection experiments with a single fault were initially performed using the AFITM method. The failure probabilities are presented in Table II. There are three types of memory.

1) Program and weight memories: A bit-flip in the program memory has a greater impact on the vision processor function when compared with that in the weight memory. This is because the processor will fail once the key instructions are inaccurate.

2) Register files: The register files of the PPU and RISC32 MPU are critical. This is because the algorithm or control process may be inaccurate once the data in these register files are flipped. For critical register files, TMR can also be adopted to protect them against SEU.

3) Other memories: The impact of other memories is negligible.

Table II. Fault injection results for various types of memory.

| Memory             | Failure probability(%) | Sobel | CNN |
|--------------------|------------------------|-------|-----|
| RISC program memory| 23.7                   | 23.1  |     |
| VMIPS program memory| 32.5                  | 12.3  |     |
| Data memory-weights | -                      | 0.7   |     |
| Register File (PPU) | 0.7                    | 16.5  |     |
| Register File (MPU) | 22.3                   | 10.9  |     |
| Register File (PEs) | 0.2                    | 0     |     |
| Data memory-image data | 0                  | 0     |     |
| Data memory-results | 0                      | 0     |     |
| Input memory        | 0                      | 0.2   |     |
| Output memory       | 0                      | 0     |     |

In case of the program and weight memories, the instructions and weights are both stored for long term. The accumulated fault injection on such a type of memory must be performed using the NAFIM method. We chose the RISC program memory as the NAFIM injection target. Theoretically, each accumulated injection into the program memory can be approximated as an independent experiment. Therefore, the processor failure probability after the $k_{th}$ accumulated injection by NAFIM should satisfy the geometric distribution:
where \( p \) is the processor failure probability after the first fault is injected and \( k \) is the accumulated injection times, which equals to the number of SEU events.

To verify the theoretical analysis, 2000 fault injection experiments were performed on the RISC program memory using the NAFIM method. In each experiment, the accumulated faults were successively injected into the memory until the failure of the processor. The total number of times that the injected faults causing processor failure was recorded after each experiment. The statistical simulation results are in good agreement with the theoretical data calculated using Eq. (1).

Bit-flips in the program memories are more likely to cause a processor failure. Considering the large storage capacity characteristics of such memories, low-cost mitigation technique should be designed to reduce the SEU effect.

5. Periodic scrubbing to improve the availability

Periodic scrubbing is an effective mitigation technique to reduce the SEU effect on the long-term stored memories and improve the processor availability. In the periodic scrubbing technique, the scrub period is a key parameter. Therefore, the relation between the availability and the scrub period should be modeled. The processor failure probability obtained from NAFIM is introduced to ensure the accuracy of the model. The modeling process is introduced as follows.

The SEU events in space can be modeled as a Poisson process with an intensity \( \lambda \) [29], which is the expected SEU rate. The SEU events can be expressed as:

\[
P_k(t) = \frac{(\lambda t)^k e^{-\lambda t}}{k!}, \quad k = 0, 1, 2, ... \quad (2)
\]

where \( t \) is the operating time and \( k \) is the number of SEU events. According to Eq. (1), the total failure probability of the processor after \( k \) SEU events is:

\[
C(k) = \begin{cases} 
\sum_{i=1}^{k} p(1 - p)^{i-1}, & k = 1, 2, 3... \\
0, & k = 0
\end{cases} \quad (3)
\]

Reliability is defined as the probability that the processor accurately operates until a given \( t \). It can be calculated as:

\[
R(t) = \sum_{k=0}^{\infty} P_k(t)(1 - C(k)) = e^{-\lambda t} \quad (4)
\]

Availability is defined as the ratio of the accurate operating time to the total time. To quantitatively analyze the processor availability, we need to obtain the average continuous accurate operating time in a scrub period, described by the mean up time (MUT). MUT can be expressed as:

\[
MUT = \int_{0}^{T_1} tf(t) dt + \int_{T_1}^{\infty} T_1 f(t) dt = \frac{1 - e^{-\lambda T}}{p\lambda} \quad (5)
\]

where \( f(t) \) is the failure probability density function and \( T_1 \) is the operating time in a scrub period. The availability can be further expressed as the ratio of MUT to the scrub period \( T \) [22], which comprises \( T_1 \) and the mean time to repair (MTTR), as depicted in Fig. 7. Thus, the availability can be calculated as:

\[
A(\infty) = \frac{MUT}{T} = \frac{1 - e^{-\lambda(T-\text{MTTR})}}{p\lambda T} \quad (6)
\]

The quantitative relation between the availability and scrub period can be obtained after the parameters \( \lambda, p, \) and MTTR are provided. These parameters are determined by the algorithm, the scrubbed memory, and the orbital altitude. In this study, the Sobel-based classification algorithm, the RISC program memory, and an orbital altitude of 2000 km were selected as an example to calculate these parameters. \( \lambda \) equaled 0.06 upset/day at 2000 km and was estimated using a professional calculation tool called ForeCAST [30]. \( p \) equaled 0.25 and was obtained from NAFIM. MTTR equaled 0.05 ms with an operating frequency of 50 MHz. Fig. 8 denotes the relation between the availability and scrub period. The availability reaches 0.999 at a scrub period of \( 10^4 \) s, indicating that the accurate operating time is 364.6 days for an entire year. Conversely, the availability is only 0.18 for an entire year when there is no scrub, which can also be calculated using Eq. (6). Therefore, periodic scrubbing is an effective mitigation technique to improve the processor availability.

![Fig. 6. Failure probability distribution when the processor runs (a) the Sobel-based classification algorithm and (b) the CNN-based classification algorithm.](image)

![Fig. 7. Time distribution in a scrub period.](image)

![Fig. 8. Availability at different scrub periods.](image)
6. Conclusion

This paper proposed novel SEU failure probability evaluation and periodic scrubbing techniques for a vision processor. The NIFID, NAFIM, and AFITM methods were used to inject faults into the DFF, long-term stored memory, and temporarily stored memory, respectively, and to evaluate the processor failure probability. These complementary methods achieved high speed and full coverage. The simulation results denote that the failure probabilities caused by bit-flips in the PPU and RISC32 MPU are high and that the program memory is more critical than other types of memory. Based on the simulation results, the proposed periodic scrubbing technique was used to increase the processor availability from 18% to 99.9% after scrubbing the RISC program memory for period of 10^4 s. The periodic scrubbing technique can effectively improve the fault-tolerance performance of a vision processor with low area and power costs.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant Nos. 61434004, 61234003, 61504141) and the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDB32050200).

References

[1] W. Miao, et al.: “A programmable SIMD vision chip for real-time vision applications,” IEEE J. Solid-State Circuit 43 (2008) 1470 (DOI: 10.1109/jssc.2008.923621).
[2] Q. Lin, et al.: “A 1,000 frames/s programmable vision chip with variable resolution and row-pixel-mixed parallel image processors,” Sensors 9 (2009) 5933 (DOI: 10.3390/s90805933).
[3] W. Zhang, et al.: “A programmable vision chip based on multiple levels of parallel processors,” IEEE J. Solid-State Circuit 46 (2011) 2132 (DOI: 10.1109/JSSC.2011.2158024).
[4] C. Shi, et al.: “A 1000 fps vision chip based on a dynamically reconfigurable hybrid architecture comprising a PE array processor and self-organizing map neural network,” IEEE J. Solid-State Circuit 49 (2014) 2067 (DOI: 10.1109/jssc.2014.2332134).
[5] J. Yang, et al.: “A heterogeneous parallel processor for high-speed vision chip,” IEEE Trans. Circuits Syst. Video Technol. 28 (2018) 746 (DOI: 10.1109/tcsvt.2016.2618753).
[6] N. Wu: “Neuromorphic vision chips,” Sci. China-Inf. Sci. 61 (2018) 060421 (DOI: 10.1007/s11432-017-9303-0).
[7] M. Strube, et al.: “Raven: An on-orbit relative navigation demonstration using international space station visiting vehicles,” (2015).
[8] A. Lindoso, et al.: “Evaluation of the suitability of NEON SIMD macroprocessor extensions under proton irradiation,” IEEE Trans. Nucl. Sci. 65 (2018) 1835 (DOI: 10.1109/TNS.2018.2823540).
[9] W.A.Powell, et al.: “High-Performance Spaceflight Computing (HPSC) Project Overview,” Radiation Hardened Electronics Technology Conference (2018).
[10] V. Vargas, et al.: “Radiation experiments on a 28 nm single-chip many-core processor and SEU error-rate prediction,” IEEE Trans. Nucl. Sci. 64 (2017) 483 (DOI: 10.1109/TNS.2016.2638081).
[11] V. Vargas, et al.: “NMR-MPar: A fault-tolerance approach for multi-core and many-core processors,” IEEE Trans. Nucl. Sci. 8 (2018) 465 (DOI: 10.3390/app08030465).
[12] P. Ramos, et al.: “SEE error-rate evaluation of an application implemented in COTS multicore-many-core processors,” IEEE Trans. Nucl. Sci. 65 (2018) 1879 (DOI: 10.1109/TNS.2018.2838526).
[13] M. Alderighi, et al.: “Experimental validation of fault injection analyses by the FLIPPER tool,” IEEE Trans. Nucl. Sci. 57 (2010) 2129 (DOI: 10.1109/TNS.2010.2043855).
[14] L. A. Naviner, et al.: “FIFA: A fault-injection-fault-analysis-based tool for reliability assessment at RTL level,” Microelectron. Reliab. 51 (2011) 1459 (DOI: 10.1016/j.microrel.2011.06.017).
[15] W. Mansour, et al.: “An automated SEU fault-injection method and tool for HDL-based designs,” IEEE Trans. Nucl. Sci. 60 (2013) 2728 (DOI: 10.1109/TNS.2013.2267097).
[16] R. Velazco, et al.: “Predicting error rate for microprocessor-based digital architectures through CEU (Code Emulating Upsets) injection,” IEEE Trans. Nucl. Sci. 47 (2000) 2405 (DOI: 10.1109/23.903784).
[17] D. Costa: Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation (Springer, 2003) 125-139.
[18] E. Jenn, et al.: “Fault injection into VHDL models: the MEFISTO tool,” International Symposium on Fault Tolerant Computing (1994) 329 (DOI: 10.1109/FTCS.1994.315656).
[19] H. R. Zarandi, et al.: “Dependability analysis using a fault injection tool based on synthesizability of HDL models,” International Symposium on Defect and Fault-Tolerance in VLSI Systems (2003) 485 (DOI: 10.1109/DFTVS.2003.1250147).
[20] A. Rohani, et al.: “A technique for accelerating injection of transient faults in complex SoCs,” 14th European Conference on Digital System Designs (2011) 213 (DOI: 10.1109/DSD.2011.31).
[21] A. M. Keller, et al.: “Benefits of complementary SEU mitigation for the LEON3 soft processor on SRAM-based FPGAs,” IEEE Trans. Nucl. Sci. 64 (2017) 519 (DOI: 10.1109/ns.2016.2635028).
[22] Z. Wang, et al.: “The reliability and availability analysis of SEU mitigation techniques in SRAM-based FPGAs,” Radiation and Its Effects on Components and Systems (2009) 497 (DOI: 10.1109/RADEC.2009.5949702).
[23] P.S. Ostler, et al.: “SRAM FPGA reliability analysis for harsh radiation environments,” IEEE Trans. Nucl. Sci. 56 (2009) 3519 (DOI: 10.1109/tns.2009.2033381).
[24] D. Patterson: Computer organization and design: the hardware/software interface (Elsevier, 2015) 5th ed.
[25] T. Ojala, et al.: “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns,” IEEE Trans. Pattern Anal. Mach. Intell. 24 (2002) 971 (DOI: 10.1109/tpami.2002.1017623).
[26] R. Rojas, et al.: “Adaboost and the super bowl of classifiers a tutorial introduction to adaptive boosting,” Tech. Rep (Berlin, 2009).
[27] E. Rosten, et al.: “Faster and better: A machine learning approach to corner detection,” IEEE Trans. Pattern Anal. Mach. Intell. 32 (2010) 105 (DOI: 10.1109/TPAMI.2008.275).
[28] S. Luetkemeier, et al.: “Imagenet classification with deep convolutional neural networks,” Adv. Neural Inform. Process Syst. 65 (2012) 1097 (DOI: 10.1145/3065386).
[29] J. S. Browning, et al.: “Single event upset rate estimates for a 16-K CMOS SRAM,” IEEE Trans. Nucl. Sci. 32 (1985) 4133 (DOI: 10.1109/TNS.1985.4334081).
[30] Cogenda: ForeCAST https://forecast.cogenda.com.cn.