An online fault injection method for the dynamic partial reconfiguration system based on a lightweight ICAP controller

Wang Guohua^{a)}, Tian Congsheng, and Luo Dongming

School of Instrumentation Science & Opto-electronics Engineering,

Beihang University, Beijing 100191, China

a) wgh@buaa.edu.cn

Abstract: Nowadays most FPGAs use SRAM-based technology, which are sensitive to radiation-induced Single Event Upsets (SEUs). To validate the reliability of FPGA, fault injection is widely used to simulate the SEUs. However, current fault injection researches mostly focus on offline. With the emerging of dynamic partial reconfiguration (DPR) technology, the requirement of online fault injection is becoming more and more urgent. This paper proposed an ICAP controller-based online fault injection method and developed an ICAP controller for DPR system. The designed ICAP controller injects SEUs by firstly reading back one frame from the specific address to local RAM, and then reconfiguring the fault frame to the original position. Moreover, it consumes half resources used by the Xilinx' AXI_LHWICAP, a quarter of those used by AC_ICAP and one tenth of those used by FT_ICAP. The experiments were conducted on a Xilinx Artix7 and validated the effectiveness of the proposed method.

Keywords: SRAM FPGAs, ICAP controller, fault injection, DPR

Classification: Integrated circuits

References

- [1] 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](https://doi.org/10.1109/TNS.2009.2033381)).
- [2] V. Chandra and R. Aitken: "Impact of technology and voltage scaling on the soft error susceptibility in nanoscale cmos," *Proc. of IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems (DFT)* (2008) 114 (DOI: [10.1109/DFT.2008.50](https://doi.org/10.1109/DFT.2008.50)).
- [3] N. N. Mahatme, *et al.*: "Comparison of combinational and sequential error rates for a deep submicron process," *IEEE Trans. Nucl. Sci.* **58** (2011) 2719 (DOI: [10.1109/TNS.2011.2171993](https://doi.org/10.1109/TNS.2011.2171993)).
- [4] M. Ebrahimi, *et al.*: "A fast, flexible, and easy-to-develop FPGA-based fault injection technique," *Microelectron. Reliab.* **54** (2014) 1000 (DOI: [10.1016/j.microrel.2014.01.002](https://doi.org/10.1016/j.microrel.2014.01.002)).
- [5] D. Radaelli, *et al.*: "Investigation of multi-bit upsets in a 150 nm technology

- SRAM device,” IEEE Trans. Nucl. Sci. **52** (2005) 2433 (DOI: [10.1109/TNS.2005.860675](https://doi.org/10.1109/TNS.2005.860675)).
- [6] J. Karlsson, *et al.*: “Application of three physical fault injection techniques to the experimental assessment of the MARS architecture,” Proc. of Dependable Computing for Critical Applications (1995) 150.
 - [7] J. Pontes, *et al.*: “An accurate single event effect digital design flow for reliable system level design,” Proc. of Design, Automation and Test in Europe Conference (DATE) (2012) (DOI: [10.1109/DATE.2012.6176466](https://doi.org/10.1109/DATE.2012.6176466)).
 - [8] S.-A. Hwang, *et al.*: “Sequential circuit fault simulation using logic emulation,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **17** (1998) 724 (DOI: [10.1109/43.712103](https://doi.org/10.1109/43.712103)).
 - [9] H. R. Zarandi, *et al.*: “Dependability analysis using a fault injection tool based on synthesizability of HDL models,” Proc. of International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT) (2003) 485 (DOI: [10.1109/DFTVS.2003.1250147](https://doi.org/10.1109/DFTVS.2003.1250147)).
 - [10] J. Su, *et al.*: “In-place LUT polarity inversion to mitigate soft error for FPGAs,” IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (2016) 81 (DOI: [10.1109/DFT.2016.7684074](https://doi.org/10.1109/DFT.2016.7684074)).
 - [11] M. Alderighi, *et al.*: “Evaluation of single event upset mitigation schemes for SRAM based FPGAs using the FLIPPER fault injection platform,” Defect and Fault-Tolerance in VLSI Systems (DFT) (2007) 105 (DOI: [10.1109/DFT.2007.45](https://doi.org/10.1109/DFT.2007.45)).
 - [12] K.-T. Cheng, *et al.*: “Fault emulation: A new methodology for fault grading,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **18** (1999) 1487 (DOI: [10.1109/43.790625](https://doi.org/10.1109/43.790625)).
 - [13] K. Vipin and S. A. Fahmy: “Enabling high level design of adaptive systems with partial reconfiguration,” The 2011 International Conference on Field-Programmable Technology (2011) 1 (DOI: [10.1109/FPT.2011.6132688](https://doi.org/10.1109/FPT.2011.6132688)).
 - [14] C. Hong, *et al.*: “A run-time reconfigurable system for adaptive high performance eddicient computing,” International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (2013) 113 (DOI: [10.1145/2641361.2641380](https://doi.org/10.1145/2641361.2641380)).
 - [15] A. Ebrahim, *et al.*: “A novel high performance fault-tolerant ICAP controller,” Adaptive Hardware and Systems (AHS) (2012) 259 (DOI: [10.1109/AHS.2012.6268660](https://doi.org/10.1109/AHS.2012.6268660)).
 - [16] J. Heiner, *et al.*: “Fault tolerant ICAP controller for high-reliable internal scrubbing,” IEEE Aerospace Conference (2008) 1 (DOI: [10.1109/AERO.2008.4526471](https://doi.org/10.1109/AERO.2008.4526471)).
 - [17] L. A. Cardona and C. Ferrer: “AC_ICAP: A flexible high speed ICAP controller,” Int. J. Reconfigurable Comput. **2015** (2015) 314358 (DOI: [10.1155/2015/314358](https://doi.org/10.1155/2015/314358)).
 - [18] Xilinx: User guide. LogiCORE IP AXI HWICAP.
 - [19] Xilinx: User guide. Soft Error Mitigation Controller.
 - [20] E. Lübbers and M. Platzner: “Multithreaded programming and execution models for reconfigurable hardware,” ACM Trans. Embed. Comput. Syst. **9** (2009) 8 (DOI: [10.1145/1596532.1596540](https://doi.org/10.1145/1596532.1596540)).
 - [21] R. Zhang, *et al.*: “A fast fault injection platform of multiple SEUs for SRAM-based FPGAs,” Microelectron. Reliab. **82** (2018) 147 (DOI: [10.1016/j.microrel.2018.01.014](https://doi.org/10.1016/j.microrel.2018.01.014)).

1 Introduction

Field Programmable Gate Arrays (FPGAs) are becoming an attractive choice for the aerospace industry due to their low non-recurring engineering costs, high computational efficiency and the feature of reconfigurability [1]. However, size shrinkage in FPGAs results in reduction of capacitance per transistor. As a consequence, particles with lower energy can generate sufficient charge to cause soft errors [2, 3]. Since soft errors are a major reliability concern in deep-sub-micron technologies, safety and mission-critical systems that works in harsh environment require resilient designs to avoid these errors. In this regard, it is important to validate the reliability of the designed system by fault injecting. Consequently, an efficient and accurate fault injection technique is needed to test and evaluate the reliability of the FPGAs design.

Fault injection techniques are divided into physical, simulation-based and FPGA-based types [4]. In physical fault injection techniques [5, 6], a specific SRAM device is exposed to radiations. This technique is able to produce the closest result to the real environment, but it requires dedicated and expensive equipments. Additionally, physical fault injection cannot control the injecting location. In contrast, simulation-based fault injection technique [7] aim to evaluate dependability features in earlier design stage and to fix any possible problems of the design. However, this technique cannot be used to perform fault injection in the FPGAs running process.

FPGA-based fault injection technique is either instrumentation-based or reconfiguration-based. Instrumentation-based technique is based on a fault injector circuit named saboteur. In [8, 9], the researchers use a non-commercial tool to add saboteur for design evaluation and create a modified design with built-in saboteur for fault injection. However, this technique needs a dedicated tool to place the saboteur, and it is time-consuming to develop such tools. Reconfiguration-based technique relies on configuration port of Xilinx FPGAs. In [10], the researchers injected an SEU into the FPGAs configuration memory via JTAG of FPGAs. It randomly reads a frame from CRAM, modifies it by flipping a bit and writes back the modified frame. In [11], the FLIPPER fault injection platform was used to evaluate SEUs in SRAM-based FPGAs. Such platforms are developed for specific FPGAs. In [12], for each fault, design is recompiled and the bitstreams need to be reprogrammed.

In modern FPGAs, the flexibility of the device is furtherly enhanced by the dynamic partial reconfiguration (DPR). DPR allows the FGPAs to change the functionality of certain blocks at run-time without interrupting the operation of the system. Many researches have explored the benefits of DPR through dynamic partial reconfiguration system (DPRS) in space applications [13]. In the soft fault injection, the functionality of fault injection needs to take system call of DPRS into account, and the consumed resources also need to be reduced. Thus we attempt to enable part of DPRS to perform the fault injection task, which shall be possessed by every reconfiguration system and equipped with required hardware for the soft fault injection. For all the fault injection, the modified configuration information shall be reconfigured to FPGAs, so we try to enable the internal configuration access port (ICAP) controller for DPRS to carry out the fault injection.

All the DPRS of FPGAs depend on the ICAP to access the configuration memory, thus an ICAP controller or ICAP manager is essential. The Xilinx' AXLICAP IP core is an off-the-shelf solution for systems which require access to the configuration memory of the FPGAs through the ICAP [14]. It provides flexible access to the ICAP combining with the PowerPC processor or the MicroBlaze processor [15]. Unfortunately, it does not provide the solution for systems that require efficient use of the FPGAs resources. Put forward by Luis Andres Cardona et al., AC_ICAP depends on other bus models to perform its function and coordinates with other micro-controllers. Therefore, it consumes more resources of FPGAs. ICAP controllers in [1, 16, 17] don't have the fault-injection function. To meet this special need of DPR and reduce the resources consumed by the specialized fault injector and the complexity of DPR system connection, we add in the fault injection function to ICAP controller.

In this paper, a novel ICAP controller-based fault injection approach is presented. The proposed ICAP controller allows both the reconfiguration of DPRS' tasks and the online fault injection of SRAM FPGAs. It features in less source utilization and the support of the DPRS commands. For fault injection, the ICAP controller receives commands from DPRS, modifies the configuration data which is read from the target location and takes advantage of the dynamic reconfiguration feature of FPGAs to perform the fault injection.

The rest of the paper is organized as follows. Section II first outlines the ICAP controller architecture. Then, the operation of ICAP controller is presented in Section III. In Section IV, we discussed the different ICAP Controllers. The experiments and results are then demonstrated in Section V. Finally, conclusions are drawn in Section VI.

2 The lightweight ICAP controller architecture

The ICAPE2 is a primitive found in Xilinx FPGAs. The ICAPE2 gives access to the FPGAs configuration memory through the configuration register. The ICAPE2 interface is shown in Fig. 1.

Fig. 1. The ICAPE2

The ICAP controller is responsible for interacting with the FPGAs' ICAPE2, it sends appropriate configuration commands to ICAPE2 to perform the operations.

The ICAP controller mainly consists of a Finite State Machine (FSM) which controls the ICAPE2 signals and Read/Write FIFOs which stream data from/to the ICAPE2. The top level block diagram for AXI HWICAP is shown in Fig. 2. The AXI-Lite interface module provides the bidirectional interface between the

Fig. 2. Top level block diagram for the AXI HWICAP core

HWICAP core and the AXI. The HWICAP module provides the interface to the ICAPE2 [16].

The basic building blocks of a reconfiguration system are the ICAP, the ICAP controller, a block RAM Controller and a Processor (Fig. 3). The target FPGA was then filled with logic to drive the LUT and block RAM utilization to approximately 70% and the I/O utilization to approximately 80%. The statistic data above shows that AXI HWICAP consumes many resources at run-time [18].

Fig. 3. Simplified block diagram of a system based on AXI HWICAP

AC_ICAP controller's application architecture under the control of PLB bus is shown in Fig. 4. Through the MicroBlaze processor controlling the reconfiguration process, AC_ICAP needs to be combined with PLB module to be PLB_AC_ICAP. Thus AC_ICAP can be controlled by PLB bus.

Fig. 4. Architecture of a system based on AC_ICAP

Fig. 5. Simplified block diagram of a system based on FTICAP.

Ali Ebrahim et al. proposed a Novel High-Performance Fault-Tolerant (FT) ICAP Controller with the usage of Triple Modular Redundancy (TMR). The application of Fault-Tolerant ICAP Controller is shown in Fig. 5. It has been shown that FSM and External Memory Interface mainly occupy the resources for the space. Without the consideration of ICAP Controller's Fault-Tolerance, FSM will occupy 609 slices, while External Memory Interface will take up 72 slices.

Fig. 6. The architecture of the proposed ICAP controller

The Fig. 6 shows the connection of the proposed ICAP controller. As the principal control unit, PicoBlaze receives the commands to control ICAP and FRAME_ECCE2. For PicoBlaze to read from ICAP it writes to a constant optimized output port which sets 'RSWRB = 1' and generates a single clock cycle active low pulse to 'CSIB'. Three clock cycles later, the value read is presented at the 'O' output of ICAP and this is captured in a register as it only remains valid for one clock cycle. Having allowed adequate clock cycles for the data to be captured, PicoBlaze can then read the 32-bit value via four input ports.

To write to ICAP, PicoBlaze prepares the 32-bit word by writing to the four output ports shown above and then writes to a constantly optimized output port to set 'RSWRB = 0' and generate a single clock cycle active low pulse to 'CSIB'.

To read from memory, PicoBlaze prepares the 12-bit address by writing to two output ports. The 8-bit data sorted at that address is then read via an input port. To write to memory, PicoBlaze prepares the 12-bit address by writing to the two output

ports shown above and then writes 8-bit data directly into the memory using a third output port that generates a write strobe ('ram_we').

Fig. 7. The FRAME_ECCE2

FRAME_ECCE2 (Fig. 7) enables the dedicated, built-in Error Correction Code (ECC) for the configuration memory of the FPGAs. This element contains outputs that allow monitoring the status of the ECC circuitry and the status of the readback CRC circuitry. When readback CRC scanning is active, SYNDROMEVALID pulses high for 1 in every 101 clock cycles as each configuration frame is read and its contents are being checked.

Table I. Resources used for different ICAP controllers

Controller	Slices	Flip Flops	LUTs
AXI-HWICAP	233	633	618
AC_ICAP	-	1193	1286
FT_ICAP	681	-	-
The proposed ICAP	64	313	320

The hardware resource usages of the AXI HWICAP, AC_ICAP, FT_ICAP and the proposed ICAP controller are presented in Table I. In Table I, it's clearly shown that the proposed ICAP controller consumes only about half of the resources used by AXI HWICAP and a quarter of the resources used by AC_ICAP.

3 The function of ICAP controller

The proposed ICAP controller can perform different operations at low-cost resources. These operations are: reading information (ICAPE2 Registers and CRCERROR status), exercising and observing the error detection and correction capabilities of the device, looking up physical frame address corresponding with linear address, reading the specified frame configuration data out of configuration memory via ICAP and stores it into the RAM buffer, writing the frame into the configuration memory and injecting SEU. Table II shows the commands of function. The performance of the ICAP controller is demonstrated in the later sections of this paper.

A. Reading configuration registers information.

The 'C' command reads and displays the contents of seven of the configuration registers including IDCODE, CTL0, STAT, COR0, COR1, WBSTAT and

Table II. Command and function.

index	command	Function
1	C	Read the configuration registers
2	E	Enable detection and correction
3	W	Write a frame
4	R	Read a frame
5	A	Look up physical frame address correspond with linear address
6	P	Specify physical address of target frame
7	F	Fault injection
8	D	Disable detection and correction

BOOTSTS. As the basic command of ICAP controller, it can decide what status the current ICAP is in.

B. Enable detection and correction

The proposed ICAP controller allows users to make the CRC verification of the configuration frame to ensure the accuracy. This command sets the COR1 register to 00810100 which will enable the built-in readback CRC scanning mechanism with error correction capability. The majority of naturally occurring SEUs will result in a single bit error within a frame. The syndrome calculated as each frame is being scanned will reveal the exact bit that has been flipped and the built-in correction mechanism will perform the read, modify and write operations to correct the frame to its original value. Due to the frame level detection of an error and its almost immediate correction the device level CRC only acts as a back-up for more serious errors.

C. Write command

This command communicates with ICAP and implements the sequence required to write one complete frame of configuration memory to the device. The frame to be written is defined by setting the Frame Address Register (FAR) to the target address currently held in scratch pad memory (previously defined using ‘F’ command). The act of writing a frame will automatically increment the value held in FAR and Readback CRC will modify the value of FAR during operation so it is always necessary to set FAR as part of the frame writing sequence.

D. Reading specific frame and storing configuration data into the RAM buffer

This command informs PicoBlaze which frame the user is interested in and it remembers this value in its scratch pad memory. PicoBlaze will then load this value into the frame address register (FAR).

This command communicates with ICAP and implements the sequence required to read one complete frame of configuration memory from the device. The frame to be read is defined by setting the Frame Address Register (FAR) to the target address currently held in scratch pad memory (previously defined using ‘P’ command). Note that the act of reading a frame will automatically increment the value held in FAR and readback CRC will modify the value of FAR during operation so it is always necessary to set FAR as part of the frame reading sequence.

E. Disable detection and correction

This command sets the COR1 register to 000000000 which turns off the built-in Readback CRC scanning mechanism. Without scanning taking place no errors will be detected and therefore no errors will be corrected.

F. Look up physical frame address corresponding with linear address

The Linear Frame Address (LA) is considered to be the position of the frame in the Readback CRC scan of the device starting with the first frame being LA = 00000000. The Physical Frame Address (PA) is the actual 32-bit address of each frame within the configuration memory map of the device. At the very start of the device the LA and PA addresses match but very soon they start to become quite different. There are two main reasons for these differences. Firstly, the memory map of the device is non-contiguous as it relates to the physical layout of the device and its various block types. Secondly, the readback CRC scan only scans the configuration frames containing static information frames associated with BRAM contents are skipped as they probably contain variable data.

Fig. 8. Artix 7 frame addressing scheme (xc7a35tcbg236 part).

Fig. 8 depicts an exemplary frame addressing scheme. In this figure the correlation between the configuration data and the underlying physical structure of the fabric can be easily seen. For instance, the global clocking resources located in the center of the fabric are mapped to a dedicated VCLK column in the middle of the bitstream. It must be noted that two pad frames are added at the end of each row.

G. Specify physical address of target frame

This command specifies a physical frame address which is then stored in scratch pad memory for use with the frame read and frame write commands. This command does not interact with ICAPE2 to set the frame address register (FAR). The value stored in scratch pad memory will be used to set FAR during every frame read and frame write operation. This is necessary because readback CRC scanning of the device modifies the value of FAR during operation. Likewise, the each act of reading or writing a frame causes the value held in FAR to be automatically incremented.

H. Fault injection

In practice, single event upsets (SEU) rarely flip more than one bit at a time and therefore most of users' experiments would be expected to emulate similar event [19].

This command enables users to flip the state of any bit of the frame data held in the RAM buffer, and prompts users to specify which one of the 32-bits in which one of the 101 words is to be flipped. After flipping the specified bit, the modified frame can be configured in the FPGAs.

Fig. 9. The procedure of frame fault injection.

Fault injection is performed to specify the addresses and then write the modified configuration frame into the original position. In the process of fault injection, firstly read the configuration frames to local RAM, then users designate the word position of soft errors and bits of words. The Fig. 9 shows the detailed process of fault injection. Users designate the injected configuration frame position based on the table above, then choose the words to be injected based on the configuration frame information in the table below. Each of words consists of 32-bits binary numbers, so the position of the injected word shall be specified (i.e. one of the 32-bits). As is shown in the Fig. 9, WORD 49 is chosen to be fault injected and the 11th bit is to be flipped, then flipped WORD 49 replaces the original one, constituting the configuration frame with fault information.

4 Experiments and results

To verify the validity of the proposed method, we conducted two experiments to test if ICAP-controller we designed is able to inject SEU fault into the SRAM-base FPGAs. In order to inject the fault automatically, the ICAP controller was added to the ReconOS [20] for the system call and reconfiguration function test. The experiments were implemented on a Xilinx Artix7 FPGA.

A. Experiment 1:

In this experiment, we inject a SEU fault into a specific position of the FPGA and then readback the configuration frame to check if the very position is converted. Firstly use the 'P' command to specify physical address of target frame. Then execute 'R' command to read target configuration frame to RAM buffer, and follow the 'F' command to flip a bit. Since the flipped configuration frame can't be verified, 'N' command shall be used to disable the automatic calibration of the ECC

and CRC values. Finally use ‘W’ command to reconfigure the modified frame. To calibrate the configuration of modified frame in the FPGAs, ‘R’ command is used to read the configured frame and display in the upper computer. We repeated the experiment plenty times and one of the results is shown as in the Fig. 10.

In Fig. 10 we can find that after the injection, one bit of the frame is flipped from 0 to 1, which represents a SEU fault in FPGA. Based on this experiment, we could simulate SEU faults in any position of the FPGA, so we conducted experiment 2 for further research.

Normal Frame Frame address = 00000010										Fault Frame Frame address = 00000010 Flipped word = 1 Flipped bit = 0									
00:00000000	01:00000000	02:00000000	03:00000000	04:00000000	05:00000000	06:00000000	07:00000000	08:00000000	09:00000000	00:00000000	01:00000000	02:00000000	03:00000000	04:00000000	05:00000000	06:00000000	07:00000000	08:00000000	09:00000000
0A:00000000	0B:00000000	0C:00000000	0D:00000000	0E:00000000	0F:00000000	10:00000000	11:00000000	12:00000000	13:00000000	0A:00000000	0B:00000000	0C:00000000	0D:00000000	0E:00000000	0F:00000000	10:00000000	11:00000000	12:00000000	13:00000000
14:00000000	15:00000000	16:00000000	17:00000000	18:00000000	19:00000000	1A:00000000	1B:00000000	1C:00000000	1D:00000000	14:00000000	15:00000000	16:00000000	17:00000000	18:00000000	19:00000000	1A:00000000	1B:00000000	1C:00000000	1D:00000000
1E:00000000	1F:00000000	20:00000000	21:00000000	22:00000000	23:00000000	24:00000000	25:00000000	26:00000000	27:00000000	1E:00000000	1F:00000000	20:00000000	21:00000000	22:00000000	23:00000000	24:00000000	25:00000000	26:00000000	27:00000000
28:00000000	29:00000000	2A:00000000	2B:00000000	2C:00000000	2D:00000000	2E:00000000	2F:00000000	30:00000000	31:00000000	28:00000000	29:00000000	2A:00000000	2B:00000000	2C:00000000	2D:00000000	2E:00000000	2F:00000000	30:00000000	31:00000000
32:00000000	33:00000000	34:00000000	35:00000000	36:00000000	37:00000000	38:00000000	39:00000000	3A:00000000	3B:00000000	32:00000000	33:00000000	34:00000000	35:00000000	36:00000000	37:00000000	38:00000000	39:00000000	3A:00000000	3B:00000000
3C:00000000	3D:00000000	3E:00000000	3F:00000000	40:00000000	41:00000000	42:00000000	43:00000000	44:00000000	45:00000000	3C:00000000	3D:00000000	3E:00000000	3F:00000000	40:00000000	41:00000000	42:00000000	43:00000000	44:00000000	45:00000000
46:00000000	47:00000000	48:00000000	49:00000000	4A:00000000	4B:00000000	4C:00000000	4D:00000000	4E:00000000	4F:00000000	46:00000000	47:00000000	48:00000000	49:00000000	4A:00000000	4B:00000000	4C:00000000	4D:00000000	4E:00000000	4F:00000000
50:00000000	51:00000000	52:00000000	53:00000000	54:00000000	55:00000000	56:00000000	57:00000000	58:00000000	59:00000000	50:00000000	51:00000000	52:00000000	53:00000000	54:00000000	55:00000000	56:00000000	57:00000000	58:00000000	59:00000000
5A:00000000	5B:00000000	5C:00000000	5D:00000000	5E:00000000	5F:00000000	60:00000000	61:00000000	62:00000000	63:00000000	5A:00000000	5B:00000000	5C:00000000	5D:00000000	5E:00000000	5F:00000000	60:00000000	61:00000000	62:00000000	63:00000000

Fig. 10. The result of flipping one bit in one frame

B. Experiment 2:

Accumulation of multiple SEUs is a widespread index to indicate the sensitivity of SRAM-based FPGAs [21], it means the average number of SEUs faults that a SRAM-based FPGA circuit can endure. In experiment 1, we injected a single SEU fault but it usually causes no damage to the whole function of the FPGA. However, if the SEU accumulates, the results are doubtful. In this case, we conducted experiment 2 to test the sensitivity of some circuits and to confirm our method furtherly. We chose five circuits of ISCAS85 benchmark for test, including c880, c2670, c3540, c5315 and c6288, and a large number of repeating experiments were conducted to get a comprehensive conclusion.

In this experiment, the procedure of injecting SEU faults is divided into four steps: readback, fault injecting, comparing and reconfiguration. As mentioned in Section III, readback combined with fault injecting will inject a SEU at a specific position of the tested circuit. After the fault is injected, the procedure comparing will check whether the function of tested circuit is right by comparing the output vectors with the golden output vectors. However, because of the redundancy of the circuits, one single SEU may not cause fatal error, therefore the output maybe normal with one or more fault injections. Thus, if the comparing procedure shows that actual output is identical with the golden output, the readback and fault injecting procedure will run again and inject another SEU at another position. The loop will not end until the comparing procedure detects a difference between two outputs. Once the difference is detected, the number of accumulated SEUs is recorded and start reconfiguration procedure to replace the damaged circuit with intact one and execute another fault injection test.

We tested every circuit 100 times and recorded the accumulated SEUs every time. At last, we calculated the average number of accumulated SEUs and the final result is shown in Table III. In the table, the nets and primitives indicate the resource utilization of the circuits. Nets indicates the place and the route, and primitives indicates the hardware resources such as CLBs.

Table III. Resource utilization and accumulated SEUs

Circuits	Nets	Primitives	Accumuted SEUs
c880	138	81	19854
c2670	263	118	9490
c5315	466	301	2165
c3540	376	323	1080
c6288	1232	1200	563

In order to validate the proposed method, we compared the accumulated SEUs of circuits with different resource utilization. As seen in Table III, the circuits with more resource utilization consume less accumulated SEUs, which means they are more sensitive to SEU faults. In fact, as the circuit becomes larger and more complex, the faults in one part of the circuit is more likely to influence the function of other parts, thus less accumulated SEUs may cause functional faults to the circuits.

5 Conclusion

To meet the requirement of online fault injection and validate FPGAs system in DPRS, an ICAP controller with fault injection interfaces is proposed in this paper. The proposed ICAP controller injects SEUs by firstly reading back one frame to local RAM and then configuring the modified frame to the fault injection position. Moreover, the ICAP controller does not rely on dedicated software or hardware, requires no regeneration of design and consumes less system resource due to its configuration by frame. Two experiments are carried out based on open-source reconfiguration system ReconOS on a Xilinx Artix7 to demonstrate the effectiveness of the proposed method. Moreover, the proposed ICAP controller consumes less resources compared with other controllers such as AXL_HWICAP, fault-tolerance ICAP, AC_ICAP and ZYCAP.