HHC: Hierarchical Hardware Checkpointing to Accelerate Fault Recovery for SRAM-based FPGAs

Enshan Yang[1,2], Keheng Huang[1,2], Yu Hu[1], Xiaowei Li[1]
[1.]State Key Laboratory of Computer Architecture,
Institute of Computing Technology, Chinese Academy of Sciences
[2.]University of Chinese Academy of Sciences
{yangenshan, huangkeheng, huyu, lxw}@ict.ac.cn

Jian Gong[3,4], Hongjin Liu[3,4], Bo Liu[3,4]
[3.]Beijing Institute of Control Engineering,
[4.]Science and Technology on Space Intelligent Control Laboratory
{gjxjtu1997, lhjbuaa}@hotmail.com, liub@bice.org.cn

***Abstract*—As the feature size shrinks to the nanometer scale, SRAM-based FPGAs are increasingly vulnerable to soft errors. Checkpointing is an effective fault recovery technique that can restore the faulty system to its previous fault free state. Since the function of the system needs to be suspended during checkpoint saving and checkpoint restoring, so the Mean Time to Repair (MTTR) of the system is critical to the system performance. In this work, we propose a hierarchical hardware checkpointing (HHC) technique that contains a high-speed on-chip checkpoint and a low-speed off-chip checkpoint to accelerate fault recovery for SRAM-based FPGAs. Most of single event effect (SEE) faults can be recovered by the high-speed on-chip checkpoint, which significantly reduces the MTTR of the system. The memory resource occupation of the on-chip checkpoint is low because HHC only stores the logic states of user bits and check information for configuration bits. Experimental results show that, compared with traditional off-chip checkpoint strategies, the proposed technique can reduce the MTTR of the system by 94.30%. In addition, the memory resource occupation is 11.11% of FPGAs, a little high but can be further optimized.**

***Index Terms*—SRAM-based FPGAs, fault recovery, hardware checkpoint, hierarchical, MTTR, ECC.**

I. Introduction

Field programmable gate arrays (FPGAs) provide an attractive design platform due to their short design cycle and low development cost. With exponential growth in performance and capacity, SRAM-based FPGAs are widely used in many application domains. In recent years, SRAM-based FPGAs are further applied in aerospace system [1] [2] and high performance supercomputers [3]–[5].

Although SRAM-based FPGAs have many advantages, they are more vulnerable to soft errors induced by high-energy particles than application-specific integrated circuits (ASICs) [6] [7], which limits their widespread usage in mission-critical applications. The function of SRAM-based FPGAs is determined by the bitstream stored in SRAM cells. When high-energy particles hit sensitive sections of the FPGA silicon, they may cause SRAM cells to flip its state, called single event effect (SEE) faults, which may affect the normal function of the system.

To mitigate soft errors in SRAM-based FPGA, there are a lot of error mitigation techniques, such as the work during logic synthesis [9], placement and routing [10]. However, these techniques can only mitigate the effect of soft errors, while cannot make SRAM-based FPGAs immune from soft errors. Consequently, fault recovery techniques are indispensable.

Checkpoint is an effective way to recover the faulty system. The procedure of checkpointing can be mainly divided into two steps. First, if the system is fault free, the running application is periodically suspended, then save the logic state of system as a checkpoint. Afterwards, if fault occurs, restore the checkpoint, so the system can resume from the fault free state. Since the checkpoint saving and restoring may affect the normal function of system, the MTTR of the system is critical to the performance. On the other hand, the controller and the storage of checkpointing strategies need additional hardware resources, so the area overhead is another concern.

Checkpoint can be established in either software or hardware. Traditional checkpoint techniques are mainly software checkpoint that are implemented with dedicated commands issued by the operating system [11]–[13]. In contrast, hardware checkpoint is introduced in recent years which utilizes hardware features to save and restore the system state [14]–[20]. According to the implementation methods, hardware checkpoint can be further divided into two categories.

The first category of hardware checkpointing is design modification. For a design, the work proposed in [14]–[17] adds specific data path to read and write state elements. In function mode, the state elements can be read and written by original functional circuit, while in checkpoint saving mode, the state elements can be accessed by the specific data path. There are several ways to implement the data path, such as establishing a scan chain for all state elements [14]–[16] or allocating an addressable RAM structure [17]. The main advantages of this category are the high data efficiency and chip independence. Taking the scan chain based data path as an example. The scanned out data exactly corresponds to the state of the system, thus its data efficiency is very high. In addition, as the scan chain is constructed at the RTL level, there is no need to know the detailed architecture of the bitstream. However, the biggest disadvantage is the high area overhead. To establish the data path, an additional hardware structure is essential for each state element, which may significantly increase the area occupation of the design. Additionally, the added data path may increase the delay of the original design, which leads to unexpected performance degradation.

The second category of hardware checkpointing is bitstream

978-1-4799-0664-2/13/$31.00

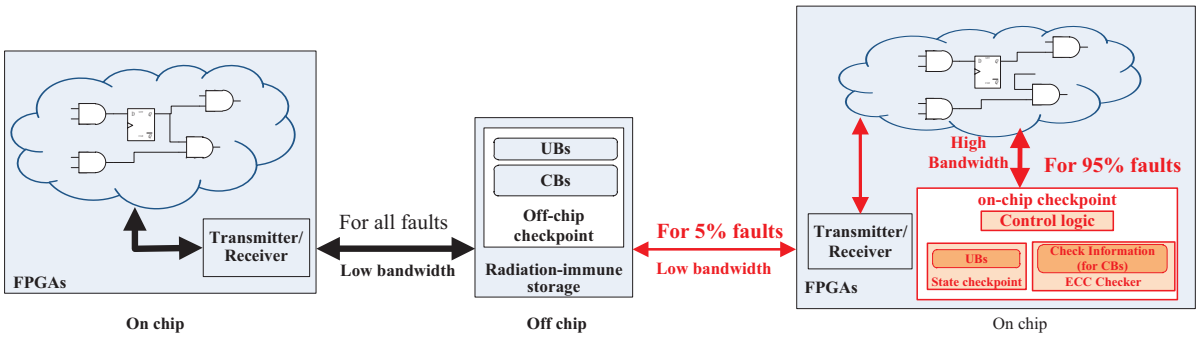


Fig. 1. Framework of the off-chip checkpointing technique and the proposed HHC technique.

readback [18]–[20], which utilizes the readback feature provided by FPGA vendors, such as Xilinx and Altera. By capturing the system state into bitstream first, and then read the bitstream back to retrieve the checkpoint at a given time. Compared to the design modification method, the main advantage of bitstream readback is no modification of the original design. Bitstream readback methods use the inherent state access structure and configuration port to readback bitstream. On the other hand, these works need to filter the state information from the bitstream, which indicates the data efficiency is relatively low. Meanwhile, the detailed knowledge of the bitstream is essential.

In mission-critical system, the saved checkpoint is generally taken as a golden image of the system, so the checkpoint should be stored in a radiation-immune storage. Since FPGAs are vulnerable to radiation-induced soft errors, existing checkpoint-based fault recovery methods store the checkpoint in off-chip storage [18]–[22], called off-chip checkpointing, as Fig. 1(a) shows. For all kinds of faults, the checkpoint needs to be transmitted to FPGAs from the off-chip storage, then restored into the system. Note that, due to the low transmission bandwidth between FPGAs and the off-chip storage, the data transmission becomes the time bottleneck of checkpointing, which severely affects the MTTR of the system. Analytical result shows that, for off-chip storage, the typical bandwidth of checkpoint transmission is 33Mbps [26]. In contrast, the bandwidth of checkpoint storing inside the FPGAs can achieve as high as 3.2Gbps [23]. There is a nearly 100 times gap between checkpoint transmission and checkpoint storage, which motivates us to increase the bandwidth of checkpoint transmission to reduce MTTR.

In this work, we focus on the readback configuration based hardware checkpointing technique, and propose a hierarchical checkpointing technique to accelerate fault recovery for SRAM-based FPGAs, as Fig. 1(b) shows. Compared to traditional SEE mitigation solutions [28], [29], Not only can the HHC technique restore the circuit function and the circuit's state based on hardware implementation, but also it achieves acceleration of system recovery by introducing on-chip checkpoint. The proposed HHC technique can reduce the MTTR of the system, while has low area overhead in FPGAs. The main contributions of this paper are:

1) The proposed HHC technique consists of high-speed on-chip checkpoint and low-speed off-chip checkpoint. Most faults can be recovered by the on-chip checkpoint, which significantly reduces the MTTR and increases the availability of system.
2) As the configuration bits (CBs) protected by error correction codes (ECC) online, the on-chip checkpoint only stores the logic state of user bits (UBs) and check information for CBs, which significantly reduces the memory resource occupation in FPGAs. On the other hand, the HHC technique stores both CBs and UBs in the off-chip checkpoint in case of multi-bit fault in CBs.
3) Some suggestions are given to Xilinx to modify the design of FrameECC. FrameECC is a hard IP core inside Xilinx FPGAs, which provides dedicated, built-in ECC for the configuration memory of FPGAs. The modified FrameECC is compatible with the proposed HHC technique, while has no adverse effect on other application of the core.

The following discussions are based on Xilinx FPGAs. But the HHC technique are widely portable as long as the FPGAs are in support of configuration readback. The rest of this paper is organized as follows. Section 2 introduces the background and motivation. Section 3 describes the proposed HHC technique. Section 4 gives the experimental results. The conclusion and future work are given in Section 5.

II. Background and Motivation

A. Architecture of SRAM-based FPGAs

The architecture of SRAM-based FPGAs can be clearly described from two levels. From logic level, SRAM-based FPGAs have fixed number of configurable logic blocks (CLBs), switch boxes and wire segments. The CLB is a multi-input, multi-output digital circuits, the switch box and wire segments are used to implement connections among CLBs. In addition, there are some hard IP cores, such as Block RAM (BRAM) and Internal Configuration Access Port (ICAP), as Fig. 2 shows. From configuration level, the function of SRAM-based FPGAs is determined by the bitstream stored in SRAM cells. The bitstream is composed of two parts, the CBs that determine the functionality of the design, and the UBs that determine the state of the running system. In Xilinx FPGAs, the bitstream is organized as a network of frames [23]. The Xilinx FPGA

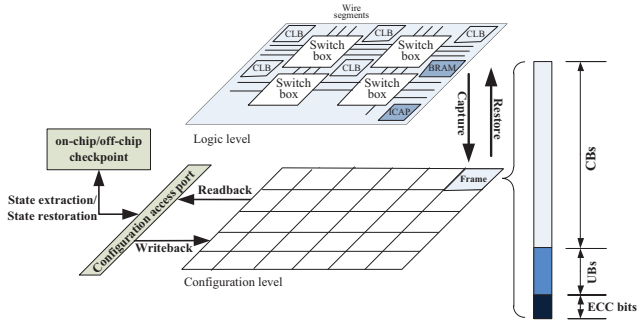


Fig. 2. Architecture of SRAM-based FPGAs.

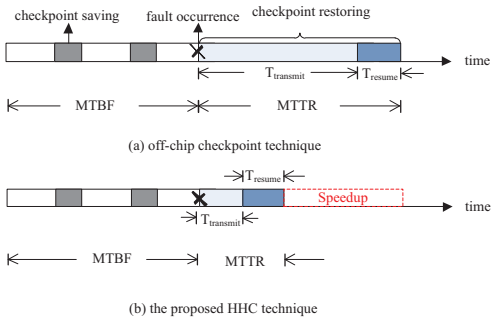


Fig. 3. Time information of checkpointing techniques.

provides such hardware feature that can be used to perform actions on configuration bitstream. The capture feature can generate a snapshot of current system state, then writes it from logic level to configuration level, which changes UBs only. This procedure takes few clock cycles. The readback feature helps getting back configuration frame, so analysis of the frame can be made thereafter. The writeback feature helps us write specific data into configuration level of FPGAs. The restore feature achieves implementation of writing data from configuration level to logic level, so the system can resume working from a given state, which is reverse process of capture. In each frame, apart from CBs and UBs, there are also ECC bits that can correct single-bit and detect double-bit fault in the frame, as Fig. 2 shows. The Primary ECC bits are calculated based on UBs and CBs. The ECC bits become invalid once UBs changed [23]. Analytical result shows that, CBs take up more than 98% of SRAM cells, while both UBs and ECC bits take up less than 2%. According to [8], more than 96% of the faults are single-bit fault, then the single-bit fault in CBs takes up nearly 98%*96%=95% of the faults in FPGAs, which is a big concern for reliability.

B. System availability and MTTR

Availability represents the percentage of time that the system is ready to serve end users, which is an important issue for mission-critical system. Availability can be calculated as follows:

$$Availability = \frac{MTBF}{(MTBF + MTTR)} \quad (1)$$

where MTBF represents the Mean Time Between Failures. To improve the availability of the system, a methodology can either increase MTBF or reduce MTTR of the system. As SRAM-based FPGAs are vulnerable to effects of Single Event Upset, the system breaks down inevitably. it's hard to improve MTBF. So in this paper, we set the goal to reduce MTTR.

Based on the Fail-Stop fault model [24] [25], there is a system failure after the fault occurs immediately. During the normal operation of the system, checkpoint is saved for a specified period. For readback configuration based hardware checkpoint technique, the procedure of checkpoint saving will not affect the normal function of system. Then, if a fault occurs, checkpoint restoring is performed to resume the system, as Fig. 3 shows. The MTTR is composed of two parts, which can be computed as

$$MTTR = T_{transmit} + T_{resume} \quad (2)$$

where $T_{transmit}$ represents the time that transmits the checkpoint from the storage to FPGAs. T_{resume} represents the time that restores checkpoint to system. For all faults, traditional off-chip checkpointing techniques need to call the low-speed off-chip checkpoint to recover the system. The low transmission bandwidth between FPGAs and the off-chip storage becomes the bottleneck of fault recovery, as Fig. 3(a) shows. In contrast, the proposed HHC can recover 95% of the faults by calling the high-speed on-chip checkpoint, which can significantly accelerate system recovery, as Fig. 3(b) shows.

III. Hierarchical Hardware Checkpointing Technique

In this work, we propose a hierarchical hardware checkpointing technique. Besides the checkpoint stored in the off-chip storage, we establish an additional on-chip checkpoint in FPGAs. By using ECC to on-the-fly protect CBs, the on-chip checkpoint only contains the state of UBs (called state checkpoint) and check information for CBs, which results in low memory resource occupation in FPGAs. Fig. 1(b) shows the overview of the proposed HHC technique. Note here state checkpoint is reliable by ECC protection.

A. Operation of the proposed HHC technique

The operation flow of the proposed HHC technique is shown in Fig. 4. When the system is power on, FPGAs is configured by the bitstream. Meanwhile, the ECC check bits of each configuration frame are computed and stored in FPGAs. Here only the check bits for CBs are computed, while the logic state of UBs are ignored.

During normal operation, we take a snapshot of state for running system into configuration bitstream. Then every used configuration frame is sequentially readback. In this phase, the ECC checker is used to examine if there are faults in CBs, and other fault detection schemes, like DMR or TMR, are used to check whether faults occur in UBs. If there is no fault for all used frames, the UBs are filtered out as both the on-chip checkpoint and the off-chip checkpoint, as procedure ① in Fig. 4 shows. On the other hand, if there is a fault detected, depending on the type of the fault, the following checkpoint recovery scenarios are possible:

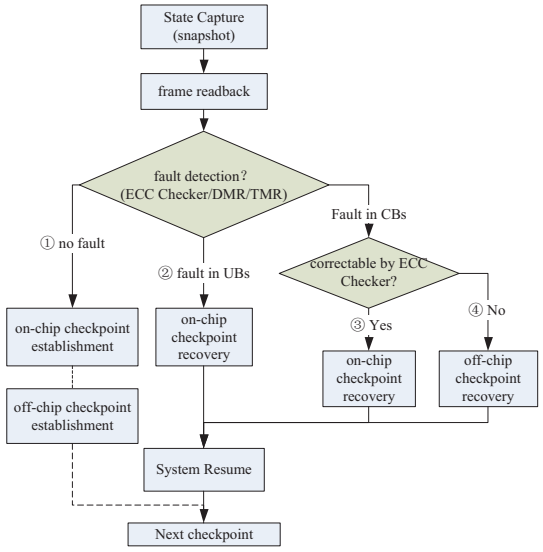


Fig. 4. Operation of the hierarchical checkpointing technique.

a) Scenario 1: fault recovery by on-chip checkpoint. If there is **a single-bit fault in CBs**, the faulty site is reported by the ECC checker. To recover the system, the faulty frame is corrected by the ECC checker first. Then, the on-chip checkpoint can be directly used to restore the correct system state, as procedure ② in Fig. 4 shows. In this case, due to the high bandwidth of data transmission in FPGAs, the time overhead of fault recovery is quite low. If there is **a fault in UBs**, additional fault detection algorithms, such as DMR or TMR, are needed to pound the alarm, then use the on-chip checkpoint to restore the system, as procedure ③ in Fig. 4 shows. Note the accuracy of state checkpoint is the basic premise every time we invoke on-chip checkpoint recovery procedure. But once the state checkpoint becomes unreliable due to rigorous environment, off-chip checkpoint is necessary to recover the system.

b) Scenario 2: fault recovery by off-chip checkpoint. If there is **a multi-bit fault in CBs** of the frame, it's beyond the error correcting capability of the ECC mechanism. During fault recovery, since the on-chip checkpoint that stores only UBs is insufficient to correct the fault, the off-chip checkpoint is used to recover the system, which is the same as traditional off-chip checkpointing strategies, as procedure ④ in Fig. 4 shows.

After fault recovery, system resumes working normally.

B. Implementation of the proposed checkpointing technique

Compared to the off-chip checkpointing techniques shown in Fig. 1(a), the proposed HHC technique additionally adds four parts, state checkpoint for UBs, ECC checker for CBs, DMR for UBs and control logic, as Fig. 1(b) shows. Here DMR is incorporated into user design, which is not shown.

Firstly, extracts the system state, then stores it in the BRAM of FPGAs and Flash. Here BRAM is protected by ECC. If a fault occurs, the system state can be recovered by directly calling the on-chip checkpoint, rather than the off-chip checkpoint as existing work does. Additionally, ECC checker is introduced as a substitution of Xilinx FrameECC. That's

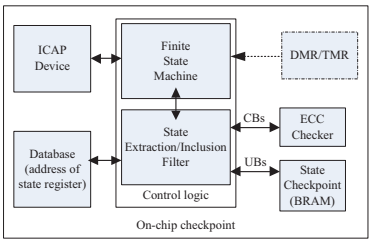


Fig. 5. Architecture of on-chip checkpoint.

because the frameECC coding algorithm generates the check bits for constant CBs and UBs [23], while UBs are volatile in the proposed HHC technique. When system state is captured into configuration level, UBs are altered, then frameECC is not applicable. To make the Xilinx ECC coding algorithm compatible, the proposed HHC filters the UBs out, and then generates check bits only for CBs. Under the circumstances, the state variation of UBs will not affect the ECC checking for CBs. The control logic manages the execution of fault detection and fault recovery. The detailed architecture of the on-chip checkpoint is shown in Fig. 5.

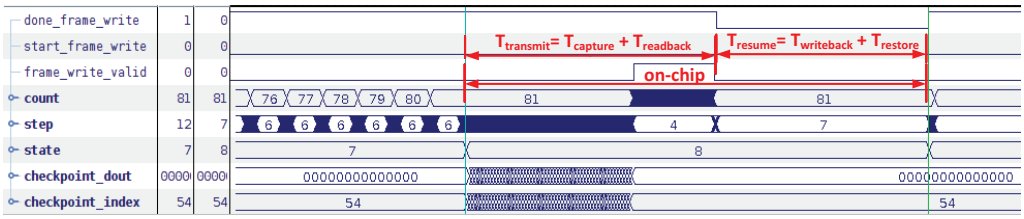
In this work, the on-chip checkpoint is established by bitstream readback. The hard IP core ICAP is used to access the bitstream of FPGAs. The database stores the address of state registers. The ECC checker generates the ECC bits for each configuration frame when system is configured for the first time, and detects the faults in CBs when the configuration frames are readback. The BRAM stores the UBs that represent the system state as the on-chip checkpoint. Also check information is stored in BRAM. The control logic manages the read/write sequence of the ICAP device. In addition, the control logic filters out UBs from the configuration frame during checkpoint saving, and composes UBs in the state checkpoint and the readbacked CBs to recover the system state during checkpoint restoring.

C. Comparison of MTTR

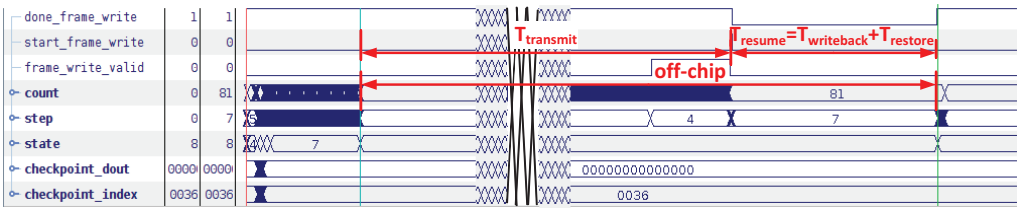
The main steps of checkpointing are checkpoint saving and checkpoint restoring. During checkpoint saving, the bitstream readback will not affect the normal function of the system, so there is no time overhead in this phase. During checkpoint restoring, if there is a single-bit fault in CBs or UBs, the proposed HHC technique first corrects the fault by ECC checker for CBs. Then the proposed HHC technique reads remaining CBs by ICAP device, and composes CBs with UBs in the on-chip state checkpoint to recover the system, which can avoid the bottleneck of data transmission between FPGAs and the off-chip storage. In this case, the time of checkpoint restoring $T_{on-chip}$ can be computed as follows

$$T_{on-chip} = T_{transmit} + T_{resume} = \frac{S_{CBs} + S_{UBs}}{BW_{on-chip}} + T_{resume} \quad (3)$$

where S_{CBs} represents the size of CBs, S_{UBs} represents the size of UBs stored in the on-chip checkpoint, and $BW_{on-chip}$ represents the transmission bandwidth of BRAM in FPGAs. Here $T_{transmit}$ corresponds to the execution time of procedure



(a) Scenario 1: a single-bit fault in CBs or fault in UBs



(b) Scenario 2: a multi-bit fault in CBs

Fig. 6. Fault recovery scenarios of the proposed HHC technique.

capture and readback, while T_{resume} corresponds to the execution time of procedure writeback and restore in Fig.2. On the other hand, if there is a multi-bit fault in CBs, the proposed HHC calls the off-chip checkpoint to recover the system. In these cases, the time of checkpoint restoring $T_{off-chip}$ can be computed as follows

$$T_{off-chip} = T_{transmit} + T_{resume} = \frac{S_{CBs} + S_{UBs}}{BW_{off-chip}} + T_{resume} \tag{4}$$

where $BW_{off-chip}$ represents the transmission bandwidth between FPGAs and the off-chip storage. Here $T_{transmit}$ corresponds to the exection time of transmitting data from off-chip storage to FPGAs, while T_{resume} corresponds to the execution time of procedure writeback and restore in 2. Since $BW_{on-chip}$ is much larger than $BW_{off-chip}$ ($BW_{on-chip}$=3.2Gbps for Xilinx XC7K325T FPGAs [23], and $BW_{off-chip}$=33Mbps for XCF32P storage device [26]), so $T_{on-chip}$ is much shorter than $T_{off-chip}$.

In terms of MTTR, traditional off-chip checkpointing technique needs to call the off-chip checkpoint no matter what kind of fault occurs. So the MTTR of off-chip checkpoint technology can be computed as

$$MTTR_{off-chip} = T_{off-chip} = \frac{S_{CBs} + S_{UBs}}{BW_{off-chip}} + T_{resume} \tag{5}$$

In contrast, the proposed HHC separates the fault recovery mechanism in different fault cases. Assuming the occurrence probability of a single-bit fault in CBs or a fault in UBs is $P_{on-chip}$, and the occurrence probability of a multi-bit fault in CBs is $P_{off-chip}$. The MTTR of the proposed HHC technique can be computed as

$$\begin{aligned} MTTR_{HHC} &= P_{on-chip} * T_{on-chip} + P_{off-chip} * T_{off-chip} \\ &= P_{on-chip} * (\frac{S_{CBs} + S_{UBs}}{BW_{on-chip}} + T_{reume}) \\ &+ P_{off-chip} * (\frac{S_{CBs} + S_{UBs}}{BW_{off-chip}} + T_{resume}) \end{aligned} \tag{6}$$

In SRAM-based FPGAs, the single-bit fault in CBs takes up about 95% of the faults ($P_{on-chip} = 95\%$), which indicates that the proposed HHC can recovery about 95% of the faults by directly calling the on-chip checkpoint, successfully avoiding the bottleneck of data transmission between FPGAs and the off-chip storage. As a result, the MTTR of the system is significantly reduced.

IV. Experimental Results

In order to validate the effectiveness of the proposed HHC technique, we implement it in Xilinx Kintex XC7K325T FPGAs. The off-chip storage is an XCF32P flash with the capacity of 32Mbit. The user design is a LEON3 SPARC processor [27]. In our experiments, all the algorithms are described in Verilog HDL language.

To establish the database shown in Fig. 5, at the stage of bitstream generation, $-l$ option is used to generate the logic location file (.ll file), which identifies the FFs of the design and their positions in the bitstream. In our experiments, we store it in the BRAM of FPGAs.

Assuming the simplest case that only a configuration frame needs to be restored. We use ChipScope to conduct analysis of the fault-recovery time in both on-chip and off-chip checkpoint cases. According to the number of clock cycles monitored by Chipscope, we can get following information(clock frequency at 5Mhz):

the time of data transmission is $T_{transmit} = 4.61\mu s$, and the time to resume the checkpoint into the system is $T_{resume} = 2.94\mu s$. So the time of fault recovery by the on-chip checkpoint in Equation (3) is $T_{on-chip} = 7.55\mu s$, as Fig. 6(a) shows. In scenario 2, the off-chip checkpoint is used to recover the system. In this case, the time of data transmission from the off-chip storage is $T_{transmit} = 1026\mu s$, and the time to store the checkpoint into the system is $T_{resume} = 2.94\mu s$. So the time of fault recovery by the off-chip checkpoint in Equation (4) is $T_{off-chip} = 1028.94\mu s$, as Fig. 6(b) shows. As mentioned before, the single-bit fault takes up 95% of faults in FPGAs, the MTTR of the proposed HHC technique in Equation

TABLE I
AREA OCCUPATION OF THE PROPOSED HHC TECHNIQUE

Module	(Used / Total)	Ratio
State checkpoint	16.4 Kbits / 16020 Kbits	0.10%
ECC check bit	842.4 Kbits / 16020 Kbits	5.26%
Frame address	921.1 Kbits / 16020 Kbits	5.75%
Control logic+ECC checker	3743 LUTs / 343680 LUTs	1.09%
Total LUTs	3743 LUTs / 343680 LUTs	1.09%
Total BRAMs	1779.9 Kbits / 16020 Kbits	11.11%

(6) is $MTTR_{HHC} = 58.62\mu s$. In contrast, the off-chip checkpointing technique always uses the off-chip checkpoint to recovery the system. The MTTR of the off-chip checkpointing technique in Equation (5) is $MTTR_{off-chip} = 1028.94\mu s$. So the proposed HHC can reduce the MTTR by 94.30%.

In terms of area overhead, Table I shows the area occupation of the proposed HHC technique. As we can see, the proposed HHC takes up 1.09% of the LUTs and less than 12% of the BRAMs in FPGAs. Furthermore, the BRAM usage and the LUTs usage can be further decreased. As described in Section II, there are ECC bits in a configuration frame, which can be used to store the ECC check bits. In addition, FrameECC also can be modified to check ECC bits [23]. Traditional FrameECC checks ECC bits for both CBs and initial UBs. But ECC bits can be invalid as we capture system state into configuration frame, which changes UBs. This way frameECC results in false error detection. So we suggest Xilinx to modify the FrameECC that only check ECC bits for CBs to make FrameECC compatible with checkpointing based fault recovery scheme. Note that, the modification has no adverse effect on other application of the FrameECC core.

V. CONCLUSION AND FUTURE WORK

In this work, we observe the bottleneck of the traditional off-chip checkpointing techniques, and proposed a hierarchical hardware checkpointing technique for fault recovery of SRAM-based FPGAs. The proposed HHC technique contains a high-speed on-chip checkpoint and a low-speed off-chip checkpoint, which can reduce the MTTR of the system by 94.30%. Furthermore, the on-chip checkpoint only stores the UBs check information for CBs, so the memory resource occupation is about 11% in FPGAs. Moreover, we can pipleline procedure of transmission and resumption of checkpoint to accelerate system recovery. In the future, we will extend the proposed HHC technique to other mission-critical applications.

REFERENCES

[1] Kenneth A. Label, Rich Katz, "NASA FPGA Needs and Activities," in Proc. of Military Aerospace Applications of Programmable Logic Devices (MAPLD), 2004.

[2] David Merodio Codinachs, "Overview of FPGA activities in ESA," MAFA2007, European Space Agency, 2007.

[3] M. Awad, "FPGA SUPERCOMPUTING PLATFORMS: A SURVEY," in Proc. of IEEE Field Programmable Logic and Applications (FPL), 2009, pp.564-568.

[4] K. Compton, S. Hauck, "Reconfigurable computing: a survey of systems and software," ACM Computing Surveys(CSUR), vol. 34, no. 2, 2002, pp.171-210.

[5] T. El-Ghazawi, E. El-Araby, M. Huang, and et al, "The Promise of High-Performance Reconfigurable Computing," IEEE Computing & Processing (Hardware/Software), 2009, pp.69-76.

[6] C. Carmichael, E.Fuller, J. Fabula and et al, "Proton Testing of SEU Mitigation Methods for the Virtex FPGA," in Proc. of Military and Aerospace Applications of Programmable Logic Devices(MAPLD), 1999. pp. 23-25.

[7] E. Normand, "Single Event Upset at Ground Level," IEEE Transactions on Nuclear Science, vol. 43, no. 6, 1996, pp. 2742-2750.

[8] H. Quinn, P. Graham, J. Krone, and et al, "Radiation-induced Multi-Bit Upsets in SRAM-based FPGAs," IEEE Transactions on Nuclear Science, vol. 52, no. 6, 2005, pp. 2455-2461.

[9] Y. Hu, Z. Feng, L. He, and R. Majumdar, "Robust FPGA resynthesis based on fault-tolerant Boolean matching," in Proc. IEEE/ACM Int. Conf. Comput.-Aided Design, Nov. 2008, pp. 706-713.

[10] K. Huang, Y. Hu, and X. Li, "Cross-layer optimized placement and routing for FPGA soft error mitigation," in Proc. IEEE/ACM Design, Automation and Test in Europe(DATE), Mar. 2011, pp. 58-63.

[11] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and et al, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," ACM Computing Surveys(CSUR), vol.34, no. 3, 2002, pp. 375-408.

[12] R. Gioiosa, J. Sancho, and et al, "Transparent, Incremental Checkpointing at Kernel Level: a Foundation for Fault Tolerance for Parallel Computers," in Proc. IEEE/ACM Super Computing Conf. (SC), 2005, pp. 1-9.

[13] N. Elmootazbellah, S. James, "Checkpointing for Peta-Scale Systems: A Look into the Future of Practical Rollback-Recovery," IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 2, 2004, pp. 97-108.

[14] D. Koch, C. Haubelt and J. Teich, "Efficient Hardware Checkpointing-Concepts, Overhead Analysis, and Implementation," in Proc. of ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays(FPGA), 2007, pp.188-196.

[15] A. Tiwari and K. A. Tomko, "Scan-chain based watch-points for efficient run-time debugging and verification of FPGA designs," in Proc. of Asia and South Pacific Design Automation Conf.(ASP-DAC), 2003, pp. 705-711.

[16] T. Wheeler, P. Graham, B. Nelson, and et al, "Using Design-Level Scan to Improve FPGA Design Observability and Controllability for Functional Verification," in Proc. of Field-Programmable Logic and Applications(FPL), 2001, pp. 483-492.

[17] J.-Y. Mignolet, V. Nollet, P. Coene, and et al, "Infrastructure for Design and Management of Relocatable Tasks in a Heterogeneous Reconfigurable System-on-Chip," In Proc. of IEEE/ACM Design, Automation and Test in Europe(DATE), 2003, pp.986-991.

[18] H. Kalte, M. Porrmann, "Context Saving and Restoring for Multitasking in Reconfigurable Systems," in Proc. of IEEE/ACM Field Programmable Logic and Applications(FPL), 2005, pp. 223-228.

[19] H. Simmler, L. Levinson, and R. Manner, "Multitasking on FPGA Coprocessors," in Proc. of IEEE/ACM Field Programmable Logic and Applications(FPL), 2000, pp. 121-130.

[20] M.A. Khan, R.N. Pittman, and A. Forin, "gNOSIS: A Board-Level Debugging and Verification Tool," in Proc. of IEEE Int. Conf. on Reconfigurable Computing and FPGAs(ReConFig), 2010, pp. 43-48.

[21] C. Carmichael, M. Caffrey, A. Salazar, "Correcting Single-Event Upsets Through Virtex Partial Configuration," Xilinx Application Notes 216, 2000.

[22] W. J. Huang, E. J. McCluskey, "A memory coherence technique for online transient error recovery of FPGA configurations," in Proc. of ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays(FPGA), 2001, pp. 183-192.

[23] Xilinx, "7 Series FPGAs Configuration User Guide," Xilinx User Guide (UG470), 2013.

[24] R. D. Schlichting and F. B. Schneider, "Fail-Stop processors: An approach to designing fault-tolerant computing systems," ACM Transactions on Computer Systems, 1, 3, 1983.

[25] M. Hiller, A. Jhumka, N. Suri, "EPIC: Profiling the propagation and effect of data errors in software," IEEE Transactions on Computers, 2004, 53(5):512-530

[26] Xilinx, "Platform Flash In-System Programmable Configuration PROMS," Xilinx Datasheet (DS123), 2006.

[27] Aeroflex Gaisler, "GRLIB IP Library User's Manual," www.gaisler.com.

[28] F. Abate, L. Sterpone, and M. Violante, "A new mitigation approach for soft errors in embedded processors," IEEE Trans. Nucl. Sci., vol. 55, no. 4, Pt. 1, pp. 2063-2069, Aug. 2008.

[29] M. Sonza Reorda, M. Violante, C. Meinhardt, and R. Reis, "A low-cost SEE mitigation solution for soft-processors embedded in Systems on Programmable Chips," in Proc. IEEE DATE, 2009, pp. 352-357.