Heterogeneous Configuration Memory Scrubbing for Soft Error Mitigation in FPGAs

Ju-Yueh Lee[1], Cheng-Ru Chang[1], Naifeng Jing[2], Juexiao Su[1], Shijie Wen[3], Rich Wong[3], Lei He[1]
[1] Electrical Engineering Department, University of California, Los Angeles, USA
[2]School of Microelectronics, Shanghai Jiao Tong University, Shanghai, China
[3]Cisco System Inc., San Jose, CA, USA

***Abstract*—In this paper, we present HCS – Heterogeneous CRAM Scrubbing – for FPGAs. By utilizing stochastic fault modeling for SEUs in CRAM, we present a quantitative estimate of system MTTF improvement through CRAM scrubbing. HCS then leverages the fact that different SEUs have unequal effects on the circuit system operation, and thus the CRAM bits can be scrubbed at different rates based on the sensitivity of the bits to the circuit system failures. To maximize the improvement on system MTTF for a given circuit system, we present a dynamic programming algorithm which solves the problem efficiently and effectively. Through a detailed case study on system level study by an H.264/AVC decoder implemented on a Xilinx Virtex-5 FPGA, we show an estimation of 60% MTTF improvement by HCS over the existing homogeneous CRAM scrubbing method, while contributing virtually no area, performance and power overhead to the system.**

Keywords-SRAM-based FPGA; Soft Errors; SER; Fault Tolerant; Memory Scrubbing; Single Event Upset

I. INTRODUCTION

State-of-the-art SRAM-based FPGAs utilize a huge amount of configuration RAM (CRAM) to implement logic and interconnect configurability for complex circuit functionality [1]. Reliability has become an issue of increasing importance for SRAM-based FPGAs over the last decade due to soft errors caused by Single Event Upsets (SEU), which can result from cosmic radiation or the circuit's internal noise. Advances in manufacturing process technology and the extensive use of SRAM have resulted in a higher susceptibility to SEUs.

Previous researches have studied the impact of SEU induced soft errors on the FPGA devices. The work in [2] and [3] used simulation based fault injection to calculate the failure rate and [4] proposed an emulation approach for SEU evaluation for FPGAs. A wide variety of techniques have also been proposed to provide more reliable FPGA devices, e.g. radiation hardened FPGA to prevent CRAM from upset [5][6], explicit module redundancy techniques such as Triple Module Redundancy (TMR) [7], and in-place mitigation techniques like IPR [8], IPD [9] and IPV [10]. However, radiation hardened FPGAs have low logic density and TMR causes excessive and unacceptable overhead in power consumption and area. In contrast, the in-place techniques leverage logic masking with minimum overhead, but do not correct faults in CRAM, where the latent faults - faults at internal circuit nodes but not reaching primary outputs - may be accumulated. Hence, an SEU correction mechanism becomes essential to avoid accumulation of latent faults and ensure correct operation of FPGA devices.

To prevent accumulation of SEUs, it is now common to use CRAM Cyclic Redundancy Check (CRC), Error Correction Code (ECC), or scrubbing in a wide range of applications. [11] evaluates two different memory scrubbing schemes including internal CRAM scrubbing solution using CRC and ECC [12] and an external scrubbing solution designed by NASA/GSFC REAG. Without frame by frame readback and ECC checking, REAG scrubber has improved performance over the solution developed by Xilinx. [13] provides a way to perform partial reconfiguration with memory scrubbing and improves the configuration time for different designs. However, none of the techniques have demonstrated quantitatively how effective or how much improvement they can make with respect to the Mean-Time-To-Failure (MTTF).

In addition, SEUs have varying impacts and sensitivities toward circuit behaviors. Rather than mitigating all the logic errors in the circuit, preventing critical failures in the system becomes a new challenge. Some SEUs may cause system failures, while others may not have a significant impact and are considered system don't cares. For example, an SEU on circuit control path is likely to cause unrecoverable system failures. In contrast, a single error on the circuit data outputs such as video or voice streaming rarely affects the system operation and only lowers the quality of the streaming content, which is usually insignificant. However, all of the existing CRAM scrubbing methods are homogeneous. That is, all of the CRAM bits are considered equally critical, which limits the improvement on system MTTF. Therefore, tuning fault tolerant techniques to consider critical system failure and system don't care becomes particularly important for FPGA applications.

In this paper, we first demonstrate for the first time how much CRAM scrubbing can quantitatively improve the system MTTF measured by the stochastic system vulnerability factor of CRAM bits. Then, we propose a Heterogeneous CRAM Scrubbing (HCS) technique, to make an intelligent use of application information to mitigate system failures to the greatest extent. To fully leverage the HCS technique, we develop a dynamic programming approach that solves the HCS problem efficiently and effectively. Our experiments first use the 10 largest combinational MCNC benchmark circuits to show that the optimized HCS technique achieves average system MTTF improvement of approximately 20%. Then, a system level study on a H.264/AVC decoder implemented on a Xilinx Virtex-5 FPGA shows an estimation of roughly 60% system MTTF improvement compared to the existing homogeneous CRAM scrubbing method. Note that the proposed HCS technique can be applied to most of the CRAM

978-1-4673-2845-6/12/$31.00 ©2012 IEEE

scrubbing solutions, such as the external CRAM scrubbing method in [11], by replacing the golden bitstream with the HCS result. Because modern FPGAs feature dynamic and partial reconfiguration, HCS does not require a stall and can be performed without interrupting the circuit operation. In addition, HCS virtually does not lead to any overhead on performance, power, and cost.

The rest of this paper is organized as follows. Section II introduces the stochastic modeling of SEU induced soft errors and its evaluation metrics. Section III presents the idea of how CRAM scrubbing improves circuit system robustness and the problem formulation of the proposed HCS technique. Section IV presents the dynamic programming approach to solve the HCS problem. Section V shows the experimental results on MCNC benchmark for module level study and a H.264/AVC decoder for system level study. Finally, Section VI concludes this paper.

II. PRELIMINARIES

FPGAs use CRAM to implement circuit functions. In general, the utilization of configuration RAM in an FPGA can be divided into four different categories: LUT, interconnect, block RAM (BRAM), and user flip-flop (FF). In terms of their quantities, LUTs and interconnect dominate BRAM and user flip-flop. Moreover, unlike soft errors on user memory (BRAM or FF), which might be self-recoverable after circuit running for several clock cycles, soft errors on CRAM have permanent impact until it is rewritten with the golden value. Therefore, in this paper, we do not consider soft errors on BRAM and user flip-flop, instead we focus on soft errors in LUTs and interconnect CRAM.

MTTF and Failures in Time (FIT) are two most commonly used design metrics to quantitatively evaluate the robustness of circuit systems. FIT is defined by the number of failures occurred in one billion hours while the circuit system is operating. On the other hand, MTTF is reversely proportional to the summation of the FIT values for every component in the system according to [14], which can be calculated by:

$$\text{MTTF (in hours)} = 10^9 \,/\, \text{FIT}_{\text{total}} \tag{1}$$

The FIT value of an FPGA can be further decomposed into two factors. First is the probability that an SEU occurs on a CRAM bit, which is the bit upset rate measured by upsets per billion hours and denoted by $R_{intrinsic\_error}$ that depends on operating environment. The other factor is the Architecture Vulnerability Factor (AVF) which is the probability of a fault leading to a system failure, i.e., a visible error at the circuit primary outputs. As a result, the FIT value of an FPGA circuit can be given by

$$\text{FIT}_{\text{total}} = \sum \text{AVF} \cdot R_{instrinsic\_error} \tag{2}$$

In fact, SEUs have various impacts on FPGAs. To quantitatively measure the sensitivity of a CRAM bit to a system failure, we define System Failure Rate (SFR). Under the single fault assumption, where at most one fault exists in the circuit at a time, the SFR of a CRAM bit evaluates the probability of the SEU on the bit that lead to system failure,

$$\text{SFR}_b = Pr(\mathbf{C}_b(x) \neq \mathbf{C}_{\bar{b}}(x) | b \xrightarrow{\text{SEU}} \bar{b}) \tag{3}$$

where $x \in (0,1)^n$ is the exhaustive set of input vectors and $\mathbf{C}_b(x)$ is the circuit status without SEU on b under x, and $\mathbf{C}_{\bar{b}}(x)$ is the circuit operation status when bit b is flipped due to an SEU. In general, the circuit operation status is usually defined by the register values of the main finite state machine or the circuit outputs that significantly affect the system operation. For theoretical study, SFR_b can be obtained by an exhaustive fault simulation on 2^n input vectors. For practical use, we can use Monte Carlo fault simulation to estimate SFR values efficiently.

While the metric of SFR_b is used in this paper as our evaluating measurement to estimate the probability of system failures due to an SEU on a specific CRAM bit, we will show in Sec. IV that it is the key factor and can be converted to AVF, FIT and MTTF on system failures when CRAM scrubbing is applied.

III. PROBLEM FORMULATION

In this section, we show how memory scrubbing can improve MTTF, and how heterogeneously rescheduling the memory scrubbing can further improve MTTF. Then, we present the formulation of HCS problem for MTTF optimization.

A. SEU Manifesting

It is known that not every SEU impacts the circuit functionality equally. In fact, an SEU takes time to manifest itself at the output and seldom raises a failure immediately. The amount of time required above has been defined as SEU Time to Manifest (TTM):

$$\text{TTM} = t_2 - t_1 \tag{4}$$

where t_1 is the time when an SEU occurs in the circuit and t_2 is the time when the circuit fails.

B. Motivation and Problem Formulation

Based on the fact above, if the SEU on a critical bit can be corrected before it induces errors, the system can stay healthy and failures can be avoided. Next, we consider memory scrubbing, and propose a heterogeneous CRAM scrubbing technique and its problem formulation.

We first define time to scrub of a CRAM bit as the scrubbing interval N of a CRAM bit (i.e., the number of clock cycles to rewrite the bit). For a traditional homogeneous memory scrubbing, which rewrites the entire CRAM memory sequentially from the first bit to the last bit, the time to scrub each bit is equivalent to the time for the scrubbing unit to go through the entire CRAM memory. Hence, it can be calculated by the total number of CRAM bits to be scrubbed divided by the number of CRAM bits that can be scrubbed in one clock cycle. When the circuit size increases by occupying more critical bits, the time required between scrubbings of a specific bit becomes longer, and this degrades the AVF. A straightforward solution to improve the AVF is to increase the speed of the CRAM scrubbing. However, the scrubbing speed is limited by the FPGA architecture.

Based on the previous CRAM bit vulnerability analysis such as [15] and [16], the vulnerability varies dramatically from node to node in the circuit, which highly depends on the circuit structure and application functionality. Intuitively, it helps to scrub critical components with higher vulnerability

more aggressively. In other words, more vulnerable bits should be scrubbed with a higher rate, and those robust bits should be scrubbed less frequently. Therefore, memory scrubbing intervals for different CRAM bits could be adjusted according to their vulnerability for higher AVF reduction. In this paper, we define the memory scrubbing schedule as a sequence of CRAM bits to be scrubbed, and the sequence is applied periodically to refresh the entire CRAM. Note that each CRAM bit is appeared at least once in the sequence to prevent the accumulation of SEUs. By adjusting the schedule, CRAM bits can be scrubbed at different rates, and the problem formulation to minimize circuit AVF is as follows.

FORMULATION 1: *Given a circuit C placed and routed onto an FPGA, the SFR values for each CRAM bit in C, the operating frequency F_c(Mhz) of C, and a memory scrubbing unit operating at F_m(Mhz) which refreshes W CRAM bits in each clock cycle, schedule the memory scrubbing units to refresh each CRAM bit (or a group of CRAM bits in sequence) in C in terms of its ordering and rate such that the AVF of C can be minimized.*

IV. HETEROGENEOUS MEMORY SCRUBBING SCHEDULING ALGORITHM

In this section, we present the reliability improvement with CRAM scrubbing based on the stochastic fault modeling presented in Section II. Then, we propose a dynamic programming based algorithm that can solve the heterogeneous memory scrubbing scheduling problem optimally with a given length of the schedule.

A. AVF update with CRAM scrubbing

Many scrubbing techniques and error correction methods by ECC have been proposed in the literature. However, no quantitative metric has been presented to show how much improvement it can provide. In this section, we present a stochastic method to estimate the improvement of CRAM scrubbing.

According to definition of Eq. (3), SFR is the probability that an error is observed at the circuit's primary output by a random input vector. For each clock cycle, we apply different random input vectors to the primary inputs. Assuming that an SEU occurs at a CRAM bit *b*, and the time to scrub the bit is *N* clock cycles, the probability that the circuit does not produce any error after *N* clock cycles is

$$\mathrm{AVF}_b = 1 - (1 - \mathrm{SFR}_b)^N \quad (5)$$

Note that *N* depends on the status of the scrubbing, i.e., at which bit the CRAM scrubbing unit is rewriting. In this paper, we assume a conservative and worst case condition that an SEU always occurs right after the CRAM bit has just been rewritten. In other words, *N* is the worst case time to scrub. Therefore, for the existing homogeneous CRAM scrubbing technique, *N* is the time to scrub the entire CRAM of the design.

B. Dynamic Programming Algorithm for Heterogeneous Memory Scrubbing Scheduling

According to Eq. (5), reducing the time to scrub *N* of one CRAM bit can reduce its AVF. However, scrubbing a CRAM bit more frequently generally result in scrubbing other CRAM bits less frequently assuming only one CRAM bit can be scrubbed at a time. In other words, reducing *N*o f a CRAM bit results in increasing *N* for other CRAM bits. Hence, a careful schedule of CRAM scrubbing process becomes essential for AVF improvement of the circuit. In this section, we propose a dynamic programming based algorithm that assigns the memory scrubbing rates (how frequently the CRAM bits are scrubbed) heterogeneously for CRAM bits based on the AVF update in the previous section to minimize the average AVF of the circuit.

In fact, the number of CRAM bits that can be written into CRAM at a time is determined by the CRAM and the FPGA architectures. Therefore, we define the CRAM granularity as the following.

DEFINITION1: We define the minimal number of CRAM bits that can be accessed at a time as **CRAM granularity**. As a result, the atomic operation of the memory scrubbing unit is to rewrite a CRAM bit or a group of CRAM bit according to the CRAM granularity.

In order to obtain a feasible solution and solve the problem efficiently in the proposed algorithm, we fix the length of the memory scrubbing schedule, which is defined as *L* as follows.

L: the number of atomic operations in the scrubbing sequence for given CRAM granularity

Then, the HCS algorithm assigns scrubbing rates for CRAM bits, i.e., how many times a CRAM bit or a group of CRAM bits are scrubbed in the given length of the schedule. Furthermore, we define

SR: the number of CRAM bits scrubbed in a clock cycle of circuit operation, which is calculated by the memory scrubbing through put divided by the circuit operating frequency, where $SR = (W \times F_m) / F_c$. Note that the circuit application on the FPGA and the memory scrubbing unit can operate under different clock domains, and *SR* is calculated under the circuit application clock domain.

S_b: how many times the CRAM bit or a group of CRAM bits *b* is rewritten in the memory scrubbing schedule, i.e., the scrubbing rate of bit *b*.

Given *L, SR*, and the SFR for each CRAM bit (group) in the following we present the mathematical formulation of the memory scrubbing scheduling problem as follows.

$$\begin{aligned} &\text{Minimize} && average\_avf = \textstyle\sum_{i=1}^{n} \mathrm{AVF}_i \,/n \\ &\text{Subject to} && \mathrm{AVF}_i = 1 - (1 - \mathrm{SFR}_i)^{Qi} \\ &\text{where} && Q_i = {}^{L}/_{S_i \cdot SR} \end{aligned} \quad (6)$$

Note that AVF is additive and can be calculated independently for each bit. Therefore, the problem above can be recursively divided into sub problems and solved optimally. Therefore, we develop a dynamic programming algorithm for a circuit given *L*, *SR*, and SFR_c, where SFR_c contains the CRAM bits (group) and their SFR in the circuit *C*, and the optimal AVF values can be achieved.

V. EXPERIMENTAL RESULTS

The proposed dynamic programming algorithm for HCS optimization is implemented in C++ on a Ubuntu server with Xeon 2.4GHZ CPU and 24GB memory. In this section, we evaluate the proposed HCS technique on two study cases: (a) a module level study using the 10 largest MCNC combinational benchmark circuits and (b) a system level study using an H.264/AVC video decoder implemented on a Xilinx Virtex-5 xc5vlx110t FPGA. For the module level, an error is recorded when a logic output is different from the right value and all logic outputs are treated equally. For the system level, an error is recorded considering so called system level don't cares. Specifically, errors for control modules (such as entropy coding and the main finite-state machine in the decoder) are recorded in the same way as that for the aforementioned module level study. But errors for data path modules are recorded only when the logic output errors are larger than the given thresholds. Clearly, the system level case study is similar to the practice and is therefore more relevant.

A. Module Level Case Study: MCNC benchmark circuits

The 10 largest combinational MCNC benchmark circuits are first optimized and mapped to 6-input LUTs using Berkeley ABC technology mapper [17]. The mapped circuits are packed using a cluster size of 8 using T-VPack tool, and placed and routed by the Versatile Place and Route (VPR) tool set [18], with a minimum dimension setting generating an FPGA as compact as possible. On the placed and routed circuit, similar to [3] and [10], we apply Monte Carlo simulation for SEUs on CRAM bits to calculate the SFR for each CRAM bits with a conservative assumption that any error at the primary outputs induces a system failure.

To evaluate the effectiveness of the proposed dynamic programming approach, we first apply the traditional homogeneous memory scrubbing and calculate the average AVF value ($AVF_{typical}$) as the baseline for comparison purpose. Then, we apply the dynamic programming algorithm to generate an HCS solution and calculate the optimized AVF value ($AVF_{optimized}$). Due to the fact that MTTF is inversely proportional to AVF, we show the MTTF improvement ratio of HCS over the baseline by calculating $AVF_{typical}/AVF_{optimized}$. We assume in the experiment that SR=100, where the memory scrubbing unit rewrites 100 CRAM bits when a circuit runs a clock cycle (a rate similar to that in Xilinx Virtex-5 assuming the circuit is running at 50Mhz), and $L=10\times|B|$, i.e., the length of HCS is 10 times the total number of CRAM bits used by the circuit.

Table I summarizes the MTTF improvement of the HCS solutions under various CRAM granularities. Some of the circuits have the same dimension but use different numbers of CRAM bits because only CRAM bits actually used are counted. The results show that on average the HCS achieves roughly 20% improvement over homogeneous memory scrubbing when CRAM granularity is equal to 16. A general trend shows that as CRAM granularity increases, the improvement by HCS decreases. This is because larger CRAM granularity typically results in lower design freedom for seeking a better HCS solution. For granularity equal to 1024, the proposed HCS can still provide an average improvement of approximately 9%.

TABLE I. MTTF IMPROVEMENT ON 10 LARGEST COMBINATIONAL MCNC BENCHMARK CIRCUIT

Circuit	Circuit Properties			MTTF improvement				Runtime (s)			
	# of CRAM bits	Dimension		Granularity				Granularity			
		x,y	w	16	64	256	1024	16	64	256	1024
alu4	103425	12,12	32	24.53%	29.70%	20.81%	12.21%	2659	455	94	13
apex2	165888	14,14	26	25.64%	23.70%	21.41%	20.39%	5344	1052	244	45
apex4	149505	14,14	26	18.75%	12.35%	8.44%	4.86%	4466	960	203	35
des	688128	15,15	36	10.16%	10.13%	6.61%	6.54%	20053	2699	581	136
ex1010	193536	17,17	32	17.50%	14.60%	8.63%	6.54%	5261	1622	344	65
ex5p	88064	17,17	32	18.75%	16.06%	13.80%	8.26%	1829	324	65	8
misex3	97208	17,17	34	22.83%	21.86%	18.26%	13.65%	2577	425	83	11
pdc	522240	42,42	18	18.53%	15.51%	6.83%	3.24%	17506	3036	655	143
seq	171008	24,24	48	24.44%	19.36%	14.78%	10.13%	5012	325	67	12
spla	466944	25,25	46	20.54%	15.13%	6.68%	3.20%	12350	2439	522	116
Average	-	-	-	20.17%	17.84%	12.62%	8.90%	7705.7	1333.7	285.8	58.4

We also list the runtime of the dynamic programming approach in Table I. On average the proposed algorithm returns an optimal solution in approximately two hours for the smallest CRAM granularity in our experiment. The runtime decreases dramatically with larger CRAM granularity because it also reduces the solution search space in the dynamic programming algorithm. When the CRAM granularity is 1024, the proposed technique can solve the circuits in a minute while providing a good improvement over the baseline.

The results show that even a circuit module has high sensitivity to system failures on its CRAM, the proposed HCS technique and the dynamic programming algorithm can still provide a good HCS solution with reliability enhancement by 6%~26% over the existing homogeneous CRAM scrubbing.

B. System Level Case Study: H.264/AVC decoder

For complex circuit systems and applications, not every circuit output error should be considered as a system failure. Instead, only errors occurred at the circuit critical outputs or registers cause system failures, which require a system reset to bring the system back to operation. For system level study, we use a H.264/AVC decoder, which is one of the most popular applications in FPGAs. H.264/AVC is a video coding standard which provides high compression efficiency via complex functionality and feature, such as entropy coding, transformation, filtering, and estimation. For example, an error in a single video frame can be considered a system don't care since the human vision system ignores such error during watching video [20]. However, a soft error in video decoder could degrade the quality in the successive video frame or even worse, lose the control of a system. For this reason, a good mitigation strategy is crucial to improve system MTTF

Unlike the module level SFR estimation, SFR estimation at the system level is much more challenging due its complexity. To solve those issues, we provide an approach of bottom-up SFR estimation. We assume control path has higher SFR than data path. Especially in video coding, an error in video output may be an acceptable error while an error in FSM output is a critical fault. Meanwhile, some modules with lossy compression functionality have error-tolerance capability like filter and IDCT. So we assume their SFR is lower than control-based module like main controller. Based on the assumptions, we define the critical registers in H.264/AVC decoder such as register in controller and entropy coding since the error in those registers will accumulate in time domain and make system enter an unknown state. Specifically, the proposed SFR estimation method can be divided into the following two steps: (i) divide the whole system into smaller modules; (ii) transfer SFR in nodes into SFR in CRAM bits for the modules. The details of the two steps are as follows.

First, we divide the system into modules by functionality illustrated in Fig. 1, and the modules are further divided into submodules. This strategy refers to actual placement and routing to define dimension for each module. We assume the CRAM bits in each module have high spatial locality. So the difference in SFR on CRAM bits for one module is relatively small, and we use the average SFR of the module as the SFR of the CRAM bits that belong to the module.

In the second step, we estimate the SFR of each module via Monte-Carlo fault simulation. For each module, the faults are injected to LUTs in the post-mapped simulation model. Then, we perform post-mapped simulation and capture the simulation result of the critical registers and the critical outputs. By comparing with the golden circuit result, we can calculate the average SFR for each module simulated. Lastly, the average SFR is assigned to the CRAM bits that belong to the module.

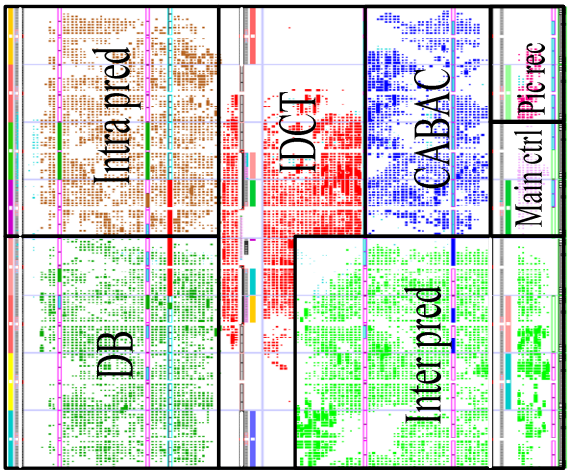

Figure 1. AVC/H.264 decoder on Xilinx Virtex-5 XC5VLX110 FPG

TABLE II. MTTF IMPROVEMENT ON THE H.264/AVC DECODER

Functional block	FF	LUT	# of CRAM bits	MTTF Improvement			Runtime (s)		
				Granularity			Granularity		
				64	256	1312	64	256	1312
CABAC	2278	9027	1093462	37.91%	35.32%	21.19%	75012	20850	5398
Intra prediction	5094	14634	3636129	60.43%	40.54%	35.24%			
IDCT	2471	10405	2971154	-1.97%	-1.87%	-2.09%			
Picture reconstruction	547	416	165051	28.27%	15.64%	8.27%			
Deblocking Filter	6648	7613	3638842	-0.84%	-0.75%	-1.07%			
Main Controller	418	466	205257	55.53%	36.73%	21.66%			
Inter prediction	8157	16133	5856249	71.73%	57.65%	48.57%			
System	-	-	-	60.01%	54.71%	40.84%			

TABLE II shows the experimental result for resource usage and improvement. For system-level analysis, the allocation in CRAM is proportional to FPGA resource although there are still some unused CRAM bits in each module. The system MTTF improvement is the weighting summation for each module improvement. The experiment in granularity = 1312 which is as the same setup as Xilinx Virtex-5 default condition [19] shows nearly 40.84% MTTF improvement. If we change granularity to 256 and 64, the improvements can even achieve 54.71% and 60.01%. The reason for such large improvement is that high percentages of CRAM bits have low SFR. This gives the rescheduling algorithm room to improve MTTF for other modules with high SFR. Negative MTTF improvement in the IDCT and de-blocking filter shows that the rescheduling algorithm sacrifices MTTF in those two modules. The bad module MTTF in the IDCT and de-blocking filter don't degrade system MTTF. In fact, the SFR in IDCT and de-blocking filter are rarely low so there is no significant drawback even if we increase their MTTF via low scrubbing frequency. However, system MTTF improves significantly since other modules have more time slots to enhance MTTF via memory scrubbing in higher refresh frequency.

VI. CONCLUSIONS AND FUTURE WORK

A Heterogeneous CRAM Scrubbing technique is proposed in this paper to improve the robustness of FPGAs against SEU induced system failures. To estimate the improvement from CRAM scrubbing, we use a stochastic fault modeling technique which allows us to calculate the MTTF improvement from CRAM scrubbing. Considering different levels of impact from SEU induced soft errors, we present a dynamic programming algorithm providing a HCS solution that effectively improves the system MTTF based on the system failure sensitivities of the CRAM bits. Our system-level estimation using a H.264/AVC decoder implemented on a Xilinx Virtex-5 FPGA shows that the proposed HCS method improves MTTF by 60% compared to the existing homogeneous CRAM scrubbing. Such improvement virtually does not have any overhead, i.e., virtually no change on area, performance and power at the system. In the future, we will develop a more efficient algorithm that solves the HCS problem. In addition, we will investigate the impact of HCS in different application domains.

REFERENCES

[1] "Virtex-7 FPGA Family Product Table," *http://www.xilinx.com/publications/prod_mktg/Virtex7-Product-Table.pdf*.

[2] C. Bernardeschi, L. Cassano, A. Domenici, "Failure probability of SRAM-FPGA systems with Stochastic Activity Networks," *in DDECS 2011*, pp. 293-296

[3] Naifeng Jing, Ju-Yueh Lee, Zhe Feng, Weifeng He, Zhigang Mao, Shi-Jie Wen, Rick Won, Lei He, "Quantitative SEU Fault Evaluation for SRAM-Based FPGA Architectures and Synthesis Algorithms", in FPL 2011, pp. 282–285.

[4] U. Legat, S. Ljubljana, A. Biasizzo, F. Novak, "Automated SEU fault emulation using partial FPGA reconfiguration," *in DDECS 2011*, pp. 24-27

[5] L. Rockett, D. Patel, S. Danziger, B. Cronquist, J.J. Wang, "Radiation Hardened FPGA Technology for Space Applications," *in Aerospace conference 2007*, pp. 1-7

[6] J. McCollum, "ASIC versus antifuse FPGA reliability," *in Aerospace conference* 2009, pp. 1-11

[7] R. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," IBM Journal of Research and Developmen, 1962

[8] Zhe Feng, Yu Hu, Lei He, Rupak Majumdar, "IPR: in-place reconfiguration for FPGA fault tolerance", in *Proceedings of the 2009 International Conference on Computer-Aided Design*, 2009, pp. 105-108.

[9] Ju-Yueh Lee, Zhe Feng, Lei He, "In-Place Decomposition for Robustness in FPGA", in *Proceedings of the International Conference on Computer-Aided Design*, 2010.

[10] Naifeng Jing, Ju-Yueh Lee, Weifeng He, Zhigang Mao, Lei He, "Mitigating FPGA Interconnect Soft Errors by In-Place LUT Inversion," in *ICCAD 2011*, pp.582-586

[11] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K.A. LaBel, M. Friendlich, H. Kim, A. Phan, "Effectiveness of Internal Versus External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis," *in IEEE transactions on nuclear science 2008*, Vol. 55, Issue 4, pp. 2259-2266

[12] Ken Chapman, "SEU Strategies for Virtex-5 Devices", 2009

[13] J. Heiner, B. Sellers, M. Wirthlin, J. Kalb, "FPGA partial reconfiguration via configuration scrubbing," *in FPL 2009*, pp. 99-104

[14] S. Mukherjee, J. Emer, and S. K. Reinhardt, "Radiation-induced soft errors: An architectural perspective," in HPCA, 2005.

[15] Samuel Luckenbill, Ju-Yueh Lee, Yu Hu, Rupak Majumdar, Lei He, "RALF: reliability analysis for logic faults: an exact algorithm and its applications", in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2010, pp. 783–788.

[16] J. Cong, K. Minkovich, "LUT-based FPGA Technology Mapping for Reliability," in DAC 2010, pp. 517-522

[17] *ABC: A system for sequential synthesis and verification*. Berkeley Logic Synthesis and Verification Group.

[18] Jonathan Rose, "VPR: A new packing, placement and routing tool for FPGA research", in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, 1997, pp. 213-222.

[19] "Virtex-5 FPGA Family Configuration User Guide", Xilinx

[20] I.S. Chong, A. Ortega, "Hardware testing for error tolerant multimedia compression based on linear transforms" , in DFT 2005.