Fast Bit Inversion Vulnerability Pre-estimation using Tcl and UPF in RTL Simulation Runtime

Myeongjin Kang, Nayoung Kwon, Seungmin Lee, Daejin Park*
School of Electronic and Electrical Engineering
Kyungpook National University
Daegu, Korea
audwls3158@knu.ac.kr, rnjsskdud999@knu.ac.kr, lsm1106@knu.ac.kr, boltanut@knu.ac.kr*

***Abstract*—As more of the world's information is digitized, it becomes increasingly important for the bits constituting the data to have integrity. Because bit errors are corrected and detected by the error correcting code module that can burden the entire system, it must be positioned only where it is needed. Therefore, it is necessary to detect places vulnerable to bit inversion error, when encoding and decoding ECC. However, the reproduction of bit inversion error cannot be deterministically described at the register transfer level (RTL) design stage, and it is difficult to find it at the RTL simulation stage. In this paper, we propose a structure that uses Tcl and unified power format (UPF) together to solve the above problem at the RTL design stage and verify the structure using a tiny processing unit (TPU). In order to find out vulnerabilities in RTL design based on the module-specific power which is derived from inserted bit inversion, we utilized the Tcl file to insert inversion error into simulation runtime and UPF to identify the power of each module. Also to validate vulnerabilities at simulation runtime, we accelerated the simulation. We branch the simulation by saving and loading the snapshot, which reduces unnecessary repetitive motions during multiple simulation times. Through this process, we verified 40% time reduction of simulation time.**

***Index Terms*—robust design, fast RTL simulation, error tolerant, error correcting**

I. Introduction

As more of the world's information is digitized, it becomes increasingly important for the bits constituting the data to have integrity. Logical data consisting of 0 and 1 do not simply mean 0 and 1, but represent information with a lot of high value. Data loss, as well as corrupted or compromised data without integrity, can cause significant damage to business or research. Data are not static. Processors process data by adding or multiplying a lot of data according to instructions. If integrity is not guaranteed in this process, the resulting data will be very far from the desired result. An unintentional inversion of 1 bit produces unpredictable results. Inversion of 1 bit can cause unpredictable and serious errors as it is combined and processed with other data. The bit inversion phenomenon occurs when the signal, including the data tier, does not reach a value sufficient for representing 0 and 1 due to external noise. In addition, microprocessing and multi-level cell (MLC), used to increase directivity in memory or processor chips lower the reliability of processing and memory storage. As the line width of the circuit and the interval between levels are reduced, the error between data transmission and reception increases [1].

To solve errors through the error correcting code (ECC) of the input/output terminal of the processor or memory, errors are managed and corrected inside the memory or processor. An ECC module generally uses hamming code to create a syndrome through the exclusive OR (XOR) operation of bits and then compares it with parity bit to detect and correct errors [2]. However, to check d_n bits, the inequality (1) must be satisfied. p_n means the number of parity bits and d_n means the number of data bits. According to inequality (1), p_n also needs to increase as d_n increases. It is installed inside the memory and processor to ensure bit integrity. However, this soon becomes the overhead of memory and processor. All data pass through the ECC module during memory input/output (I/O), and this becomes a bottleneck at the instruction fetch stage inside the processor. It strengthens the bottleneck and delays the overall execution time by increasing the execution time of each stage. Therefore, data integrity must be guaranteed by installing the minimum ECC module in the exact location required. A step is required to identify a vulnerability where bits can be inverted inside the design [3].

$$2^{p_n} \geq d_n + p_n + 1 \tag{1}$$

Several steps are taken to make a chip, define behavior and verify functionality through RTL design, and test functionality by iteratively simulating multiple scenarios. After that, based on RTL design, gate level design, layout, and chip production are performed. Bit inversion errors are caused by external noise or the characteristics of a process step. Therefore, it is difficult to verify bit inversion errors in the RTL stage of implementing and verifying functions. The possibility of bit inversion due

This study was supported by the BK21 FOUR project (4199990113966), the Basic Science Research Program (NRF2018R1A6A1A03025109), (NRF2022R1I1A3069260)through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, and (2020M3H2A1078119) byMinistry of Science and ICT. This work was partly supported by an Institute of Information and communications Technology Planning and Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2021-0-00944, Metamorphic approach of unstructured validation/verification for analyzingbinary code, 10%) and (No. 2022-0-01170, PIM Semiconductor Design Research Center, 20%) and (No. RS-2023-00228970, Development of Flexible SW-HW Conjunctive Solution for On-edge Self-supervised Learning, 20%) and (No. RS-2022-00156389, Innovative Human Resource Development for Local Intellectualization support program, 50%). The EDA tool was supported by the IC Design Education Center (IDEC), Korea.

to signal change or process noise comes after actual chip fabrication. However, as bit inversion occurs after the chip is manufactured, if an ECC module suitable for the bit is added in the RTL step, the cost of performing the previous step again occurs. In this paper, we pre-estimated and determined the vulnerability of bit inversion errors that can occur in the chip at the RTL stage.

UPF is an acronym for unified power format which is an Institute of Electrical and Electronics Engineers (IEEE) standard for specifying power intent [4]. UPF is used throughout the design flow such as design specifications, logic synthesis, physical synthesis, analysis, and verification. UPFs can be used during RTL development and implement power and ground for cells created during synthesis [5]. This means that power consumption for each module can be measured differently by supplying different power to each module in the RTL step. In the case of a processor, power consumption data are obtained through different power supplies for each module, such as arithmetic logic unit (ALU) and controller, and through this, it is possible to know how each module operates. If bit inversion is injected through the Tcl file, the operation of each module can be identified through UPF, as well as which module is vulnerable to which bit inversion [6].

In this paper, Section I introduces the problem presented in this paper, Section II describes the background knowledge used in this paper and motivation for solving the problem, Section III describes proposed architecture to solve the presented problem and Section IV verifies the proposed architecture, with processing the simulation proves its validity, and finally concludes in Section V.

II. Background and Motivation

When a bit is inverted, a wide variety of outcomes can ensue. If an ECC block is encountered after a bit inversion error occurs, the bits are inverted again to easily display the original data and operate normally. However, when processed with other data, the error spreads to multiple modules. As a result, a variety of errors occur, from software soft errors that simply change the result value to serious errors that touch the hardware that needs to be locked. As seen in Fig. 1, alternatively, an operation error may occur after an unexpected time as the stored value is later used for operation by a processor. Memory can correct errors through the ECC module during input/output, but when manufacturing a chip that includes a processor, it is necessary to identify vulnerable points against multiple bit inversions and decide whether to insert the ECC module [7].

For many chips, understanding the power consumption is a good indicator of the chip's operation. In particular, in the case of designing a small chip such as an embedded system, the operation of the chip leads to power consumption. Analysis of power consumption data are used for actual fault detection and diagnosis, as well as in the case of side channel attack to determine the operation of the chip from the outside. When a chip malfunctions, and a hardware error occurs, it does not go to the next flow, so the bit toggles decrease or it enters an unwanted

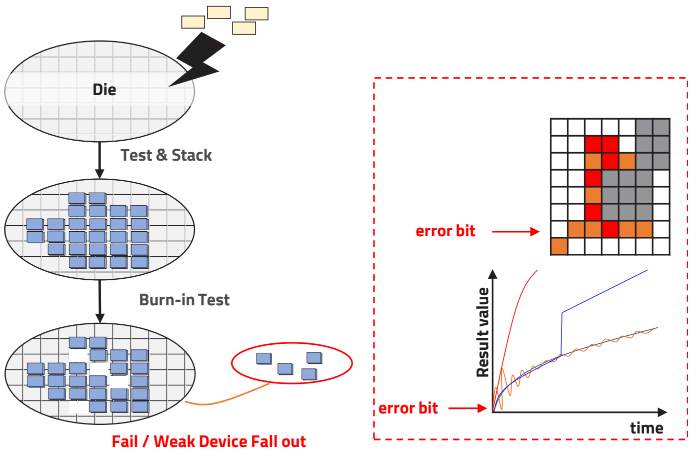

Fig. 1. Bit inversion effect

loop and the number of toggles increases abnormally. When bit inversion occurs inside the chip, it is possible to identify which position of each module is vulnerable to bit inversion by identifying the power consumption of each module.In this paper, we tested the power consumption of each module for the bit inversion situation in the simulation stage, and based on this, we tried to identify the weak points of the bit inversion of each module [8].

Using Tcl and UPF, it is possible to insert bit inversion errors into multiple modules in the RTL simulation runtime and receive the results as power consumption data. However, as the module grows, the number of places where bit inversion can occur due to noise continues to increase. RTL design is carried out by combining several modules together, and bit inversion occurs during communication between RTL modules or inside the module. Eventually, as the number of modules increases, the number of simulations increases by multiplying the existing number by the number of modules added and the number of lines to which the module is assigned. As the size of RTL design increases, the simulation execution time increases exponentially. This is the biggest obstacle to predicting and analyzing the possibility of errors for complex modules [9].

A lot of time is consumed to verify the robustness of the model designed in the RTL simulation runtime. When bit inversion is inserted for the inter-module communication part where bit inversion can occur, the RTL design model does not change, and most of the time, normal operation is performed by the premade testbench. Most of the various simulations perform the same operation, and this operation occupies most of the simulations. This paper uses Tcl to save and branch intermediate processes of simulation. To simulate multiple bit inversions, it is important to use the ”checkpoint” and ”restore” functions to reduce the number of simulation iterations for common intervals. This replaces the repetitive compilation time for RTL with the simulation save and restore times. As many intermediate process snapshots are stored, memory usage increases and the tradeoff results in a decrease in the overall execution time [10].

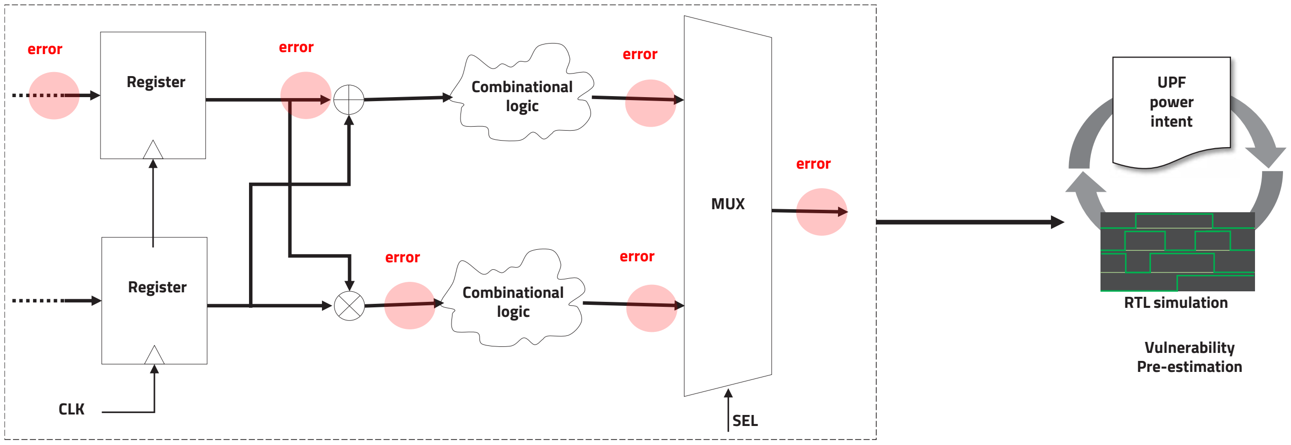

Fig. 2. Module toggle counting with each power

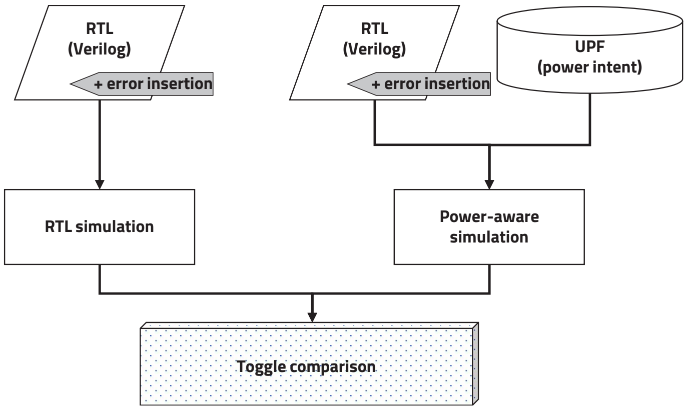

Fig. 3. Proposed flow

III. Proposed Architecture

In the existing RTL simulation stage, it was not possible to present a robust design for the bit inversion phenomenon. The bit inversion error is not revealed in the RTL simulation, and even if the bit inversion is forcibly injected using the testbench, the result of one error can be seen in the simulation runtime, and the result due to the bit inversion error cannot be explained.

Therefore, as seen in Fig. 2 we used Tcl to insert several types of bit inversion errors into the RTL simulation runtime, UPF to receive the result, and, use the number of toggles and input Vdd in RTL design to receive the vulnerability of bit inversion errors for each module, as well as to determine its cause. This paper provides information on robustness at the RTL stage so as to enable a robust system design by pre-estimating the vulnerability to bit reversal errors at RTL simulation runtime using Tcl and UPF.

In this paper, we performed RTL simulation after RTL design to prevent errors due to bit inversion, as seen in Fig. 3. When normal operation is guaranteed after RTL design, the presented simulation environment assumes communication between modules or bit inversion that can occur inside the module. Based on this, it is important to proceed by inserting bit inversion into the simulation runtime several times using the Tcl file. At this time, it is assumed that the error inserted into each module is only a 1-bit inversion. Inversion errors of 1-bit or more in most cases result in simulation results that cannot proceed any further, or because most modules show a lot of differences from the existing reference, it is not appropriate to find a module that is vulnerable to bit inversion.

By using the Tcl file, it was possible to insert bit inversion between modules or between modules at runtime. We used UPF to analyze and feedback the results for each module. UPF is an IEEE standard that defines power intent in power optimization for electronic design automation. The standard was made public through the donation of Accellera equipment. We used UPF is used to apply various voltages to each module so that it can operate at different power sources. We derived the results of each module by applying different voltages to it, and then we combined these with the simulation results.

Energy consumption per module is defined by the Eq (2). The energy consumption of each module is the integral of power, and power is composed of dynamic power consumption when data are toggled in a module made of CMOS and power consumption when 1 and 0 are displayed. The structure proposed in this paper analyzes energy consumption data by calculating the number of toggles through bit inversion insertion as well as the time of data values indicated by each module.

$$E_{module} = N_{toggle} * \int P_{toggle} dt + \int_0^{t1} P_1 dt + \int_0^{t0} P_0 dt \tag{2}$$

We used the customized TPU based on MIPs to verify the proposed structure. The structure of TPU is as follows. The TPU has five pipeline sections, instruction fetch, decode, execute, memory access, and register write. To implement this, we designed the TPU by dividing it into five modules in the RTL design. Each consists of ROM, RAM, Controller, Decoder, and ALU. Each module interacts with one another for the operation of the TPU and operates to process 16-bit

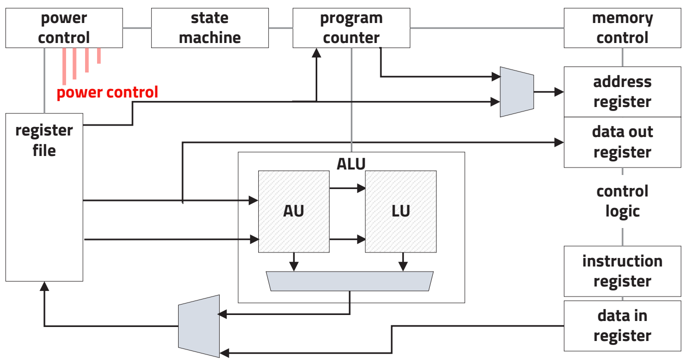

Fig. 4. Processor architecture

commands by passing through five pipelines such as a Mips-based processor. In the instruction fetch stage, instructions are requested and received from RAM, and if a hit does not occur, ROM is accessed and instructions are fetched. After that, the command is transmitted to the decode module to proceed with the process of decoding the command. Based on the decoded command, the ALU operates in the execute stage, and accesses memory or updates registered based on decoded command.

When each module transmits/receives data to perform the operation of the processor or when data are toggled within each module, bit inversion error may occur. As a result of the bit inversion that occurs at this time, after parity check through the ECC checker, it can converge to normal operation or enter an unused module and then converge to normal operation. However, when corrupted data are stored in memory and read again, or when they operate through several modules, a 1-bit error passes through to another module, causing a software or hardware error.

The model that injects bit errors and examines the results proposed in this paper requires a long time in the RTL simulation process. Bit inversion is forced during the procedure of processing multiple commands in the TPU, and the code and preprocessing method that have been verified for normal operation are all consumed for one simulation. For one simulation, it is crucial to run it according to the process of compiling RTL and the written testbench. The structure proposed uses Tcl and UPF to implement bit errors at simulation runtime in Fig. 4 and Fig. 5. After it is compiled, the simulation will inject errors, so there is no need for repetitive compilation.

We present the structure shown in Fig. 3 to execute the proposed structure in RTL simulation runtime. The proposed simulation process simulates normal operation once through one compilation time t_c and simulation t_s. Afterwards, snapshots are saved for each location where bit errors are to be inserted. After this, all simulations proceed with the remaining simulations through the process of loading snapshots rather than compilation. As shown in Eq (3), in the case of the original simulation, simulation time $T_{original}$is proportion to the product of each module and the node between the modules.

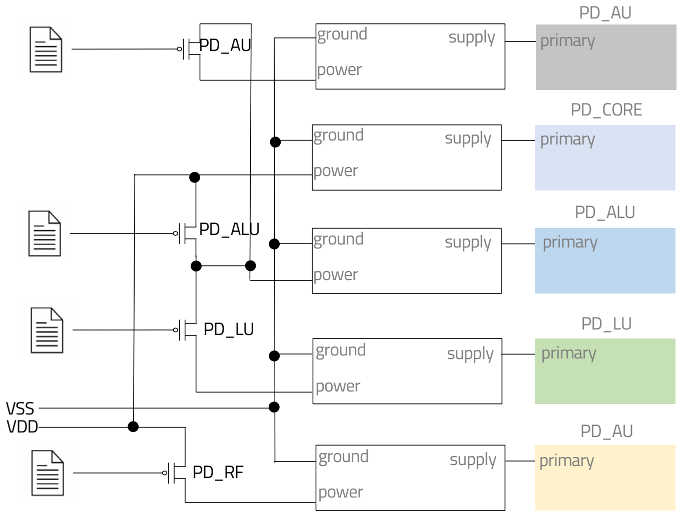

Fig. 5. Processor UPF connection

The structure proposed in this paper proceeds with simulation as much as the product of the number of modules and nodes between them, as previously demonstrated, but compilation proceeds once during the initial simulation. After that, a snapshot is called and processed, resulting $T_{proposed}$ of Eq (4).

$$T_{original} = (t_c + t_s) * module_n * node_n \tag{3}$$

$$T_{proposed} = (t_c + t_s) * t_s + t_l * module_n * node_n \tag{4}$$

IV. Experimental Results

A. Experimental setup

This paper proposes simulation runtime bit inversion error insertion through Tcl and vulnerability identification based on power consumption data through UPF. To verify this, we utilized TPU as an experimental setup. TPU consists of top, memory, decoder, ALU, registers, and the controller unit as shown in Fig. 4. The TPU is connected with a total of six modules and 30 nodes, and the simulation examines the results of bit inversion of 1 node and tje bit inversion of 2 nodes.

B. Bit inversion vulnerability

In this paper, we inserted bit inversion into the simulation runtime to identify the bit inversion vulnerability of each module. In simulation runtime, the result of bit inversion insertion is expressed as power consumption data through UPF. Power consumption data are shown by measuring the number of toggles and the time representing 1 and 0, respectively, as shown in the Eq (2). The TPU executes 16 instructions. As a result, errors per module occur as shown in Fig. 6, or converge to normal operation in case of small errors. Power consumption data are indicated through the measurement of time representing 0 and 1 as well as the number of simple

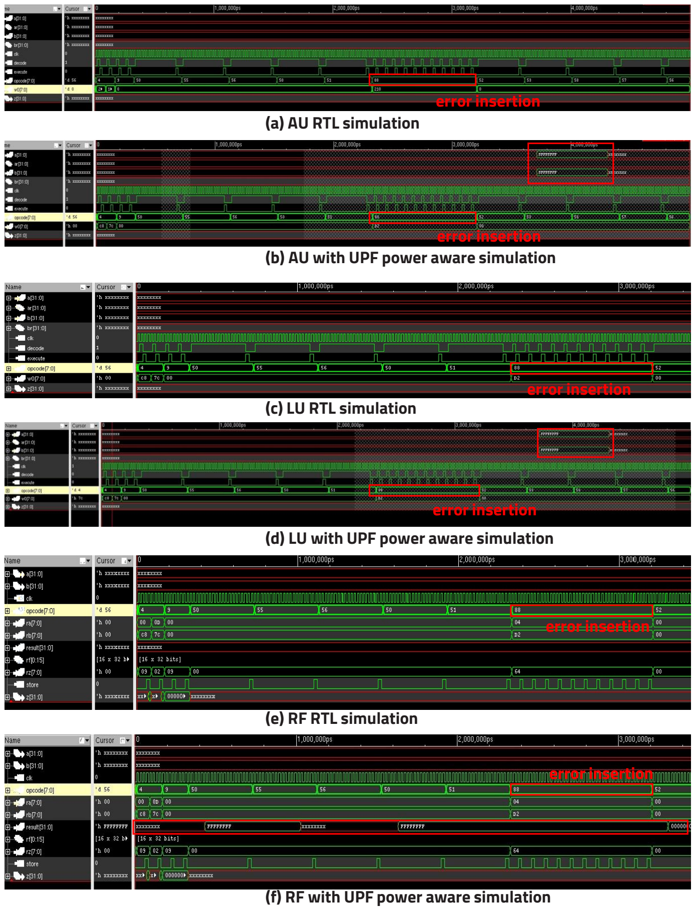
error insertion
(a) AU RTL simulation
(b) AU with UPF power aware simulation
(c) LU RTL simulation
(d) LU with UPF power aware simulation
(e) RF RTL simulation
(f) RF with UPF power aware simulation

Fig. 6. Toggle comparison

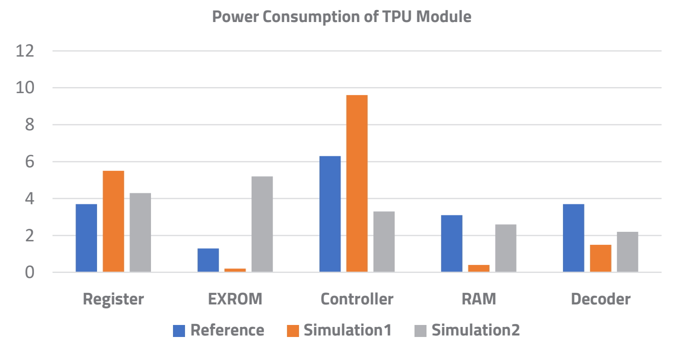

Fig. 7. Power consumption of modules

toggles as seen in Fig 7, so it is possible to identify which module is vulnerable to which error insertion location.

C. Simulation time

We simulated 16 instructions on a TPU consisting of six modules and 30 nodes. In the case of one simulation, it takes 14.03 sec for compilation, elaboration, waveform generation, and data parsing. About 3,840 simulation iterations are required to simulate the proposed structure, and the entire

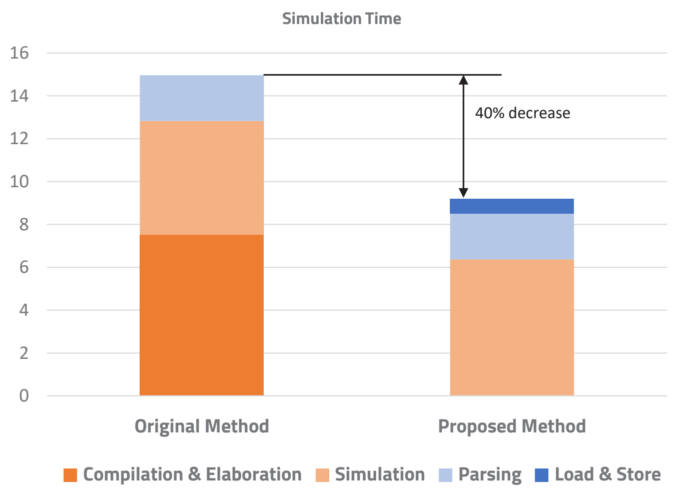

Fig. 8. Simulation elapsed time

	Repetition	Compilation & Elaboration (hour)	Simulation (hour)	Parsing (hour)	Load& Store (hour)
Original	3,840	7.52	5.31	2.14	0
Proposed	3,840	0	6.38	2.14	0.51

TABLE I
SIMULATION ELAPSED TIME TABLE

simulation takes about 15 hours. This is the result of a simple TPU, and as the module being tested becomes more complex, the simulation time increases exponentially with complexity and size. Through the structure proposed in this paper, we created a snapshot through one compilation and simulation. After that, for each simulation, a snapshot is called and error bit insertion and data parsing were performed. As a result, the simulation time was reduced by about 40% from 15 hours to 9 hours as seen in Fig. 8 and Table IV-C. Although the simulation time increased with the use of Tcl, the result of the omission of repetitive compilation and elaborate processes took up the most time.

V. Conclusion

This paper pre-estimates the risk of bit inversion, an error that occurs during chip design, at the RTL simulation stage. At this time, to verify errors that cannot occur at the RTL stage, Tcl and UPF were used together to forcibly insert a bit inversion error into the model and log the result. Through the proposed model, we analyzed the result value through the power consumption data of the module to be connected to the UPF in order to determine whether there is a malfunction in case of bit inversion. Utilizing the power consumption data, it is important tofind the location where the ECC block is required by determining the vulnerable position of each module for the bit inversion error in the RTL simulation step or to send the result to the gate level, netlist design and layout to request an additional design.

It is also very time-consuming to simulate many bit inversions in a large RTL design model. The communication part and the bit inversion simulation for each module consume simulation time, which increases exponentially according to the RTL size. In this paper, we saved and recalled snapshots of simulations to accelerate the RTL design process. Through this, meaningless repetition was reduced, and the simulation time was reduced by 40%.

This paper guarantees robustness against the bit inversion phenomenon, which was difficult to verify in the RTL design stage, in order to solve problems that may occur after chip design. In addition, because numerous repetitive experiments are required to ensure robustness, the bit reversal risk check was simulated in the RTL simulation runtime through the process of branching the simulation by saving and recalling snapshots of the simulation. This has significance in that chip problems that do not occur due to RTL problems in RTL simulation can be checked in the simulation runtime. For the future work, we plan to verify additional issues that may occur in a chip using the proposed model.

References

[1] K. Rahul and S. Yachareni, “Area and power efficient ECC for multiple adjacent bit errors in SRAMs,” 2020 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 2020, pp. 1-4.

[2] Kang M, Park D. “Lightweight Microcontroller with Parallelized ECC-Based Code Memory Protection Unit for Robust Instruction Execution in Smart Sensors” in Sensors, vol.21, 2021.

[3] K. C. Chun et al., “A 16-GB 640-GB/s HBM2E DRAM With a Data-Bus Window Extension Technique and a Synergetic On-Die ECC Scheme,” in IEEE Journal of Solid-State Circuits, vol. 56, no. 1, pp. 199-211, Jan. 2021.

[4] “IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems,” in IEEE Std 1801-2015 (Revision of IEEE Std 1801-2013) , vol., no., pp.1-1068, 25 March 2016.

[5] T. English, K. L. Man, E. Popovici and M. P. Schellekens, “HotSpot : Visualizing dynamic power consumption in RTL designs,” Proceedings of IEEE East-West Design, Test Symposium (EWDTS’08), Lviv, Ukraine, 2008, pp. 45-48.

[6] A. Kalsing, L. Fesquet and C. Aktouf, “Towards consistency checking between HDL and UPF descriptions,” 2017 Forum on Specification and Design Languages (FDL), Verona, Italy, 2017, pp. 1-6.

[7] V. Gherman, S. Evain and B. Giraud, “Binary Linear ECCs Optimized for Bit Inversion in Memories with Asymmetric Error Probabilities,” 2020 Design, Automation, Test in Europe Conference, Exhibition (DATE), Grenoble, France, 2020, pp. 298-301.

[8] M. Kang and D. Park, “Remote Monitoring Systems of Unsafe Software Execution using QR Code-based Power Consumption Profile for IoT Edge Devices,” 2021 International Conference on Electronics, Information, and Communication (ICEIC), Jeju, Korea (South), 2021, pp. 1-4.

[9] C. C. . -H. Hsu and C. H. . -P. Wen, “Speeding up power verification by merging equivalent power domains in RTL design with UPF” 2017 International Test Conference in Asia (ITC-Asia), Taipei, Taiwan, 2017, pp. 168-173.

[10] Y. Lee and D. Park, “Fast Verilog Simulation using Tel-based Verification Code Generation for Dynamically Reloading from Pre-Simulation Snapshot,” 2023 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Bali, Indonesia, 2023, pp. 595-597.