Implementation and Evaluation of a Speculative Execution-Based FPGA Accelerator for Electronic Circuit Simulation Using Gauss–Jordan and BiCGSTAB Methods

Yuma Omoto Atsushi Kubota Tetsuo Hironaka

Department of Computer and Network Engineering
Graduate School of Information Sciences, Hiroshima City University
3-4-1 Ozuka-Higashi, Asaminami-ku, Hiroshima City, Hiroshima 731-3194, Japan
mk66007@e.hiroshima-cu.ac.jp kubota@hiroshima-cu.ac.jp hironaka@hiroshima-cu.ac.jp

This paper proposes a speculative execution system on FPGA to accelerate circuit simulation by combining Gauss–Jordan Elimination (GJE) and the BiCGSTAB method. Both solvers run in parallel, and the first to converge is adopted to reduce latency. Experiments on 20×20, 40×40, 80×80, and 160×160 matrices measured latency and resource use to evaluate scalability. A prototype on a Xilinx ZCU104 FPGA was further tested with a 20×20 circuit. Results show that GJE is effective for small problems, while BiCGSTAB achieves higher efficiency for larger dimensions and certain conditions, confirming the benefit of the proposed speculative execution approach.

I. Introduction

Hardware-in-the-Loop (HIL) simulation is increasingly used in automotive and aerospace industries for early-stage verification by integrating real hardware with simulation. Efficient HIL requires fast, continuous circuit simulation, but conventional CPU-based and GPU-based methods are constrained by linear solver costs and I/O latency. FPGA acceleration, with high parallelism and rich I/O, offers a promising alternative[1][2]. This study implements a SPICE[3]-based circuit simulator on FPGA to provide a high-performance HIL platform for machine control electronics.

However, implementing direct solvers such as Gauss–Jordan Elimination (GJE) on FPGAs is constrained by resource usage and latency. As matrix size grows, the resource demand of GJE escalates, limiting scalability. In contrast, the BiCGSTAB iterative solver, though not always ensuring convergence, offers better efficiency and scalability. This study proposes speculative parallel execution of BiCGSTAB with GJE to reduce latency.

The remainder of this paper is organized as follows. Section II introduces the simulator and solvers, Section III describes the implementation and optimization, Section IV presents the evaluation, Section V discusses the results, and Section VI concludes the paper with future directions.

II. System Architecture

A. *Overview of the Electronic Circuit Simulator*

To accelerate the electronic circuit simulator using FPGA, this study adopts a speculative execution system that runs multiple solvers concurrently in order to reduce simulation time. Toward this goal, we have designed the system architecture illustrated in Figure1.

Processing System (PS) Section: The PS, equipped with an onboard ARM CPU, receives a SPICE netlist, parses it to extract component parameters and connectivity, and stores this data in DRAM for use by the Programmable Logic (PL) section.

Programmable Logic (PL) Section: At startup, the control unit transfers circuit data from DRAM to on-chip shared memory (BRAM). Simulation proceeds in three steps:

1. **Linear Equation Construction:** Generates linear equations from the input data. Multiple 32-bit input channels (#1, #2, ..., #N) are prepared for external inputs.
2. **Solver Execution:** Dedicated solver units solve the equations in parallel.
3. **Output:** The resulting branch voltages are output via multiple channels (#1, #2, ..., #N) to external systems.

Data exchange between processing units is performed over a 64-bit data bus.

B. *Proposed Speculative Execution System*

Solving linear equations dominates computational time in electronic circuit simulation, making its acceleration essential. The direct Gauss–Jordan Elimination (GJE) method has $O(n^3)$ complexity, which is feasible for small matrices but becomes costly as size increases. In contrast, the iterative BiCGSTAB method operates at $O(kn^2)$,

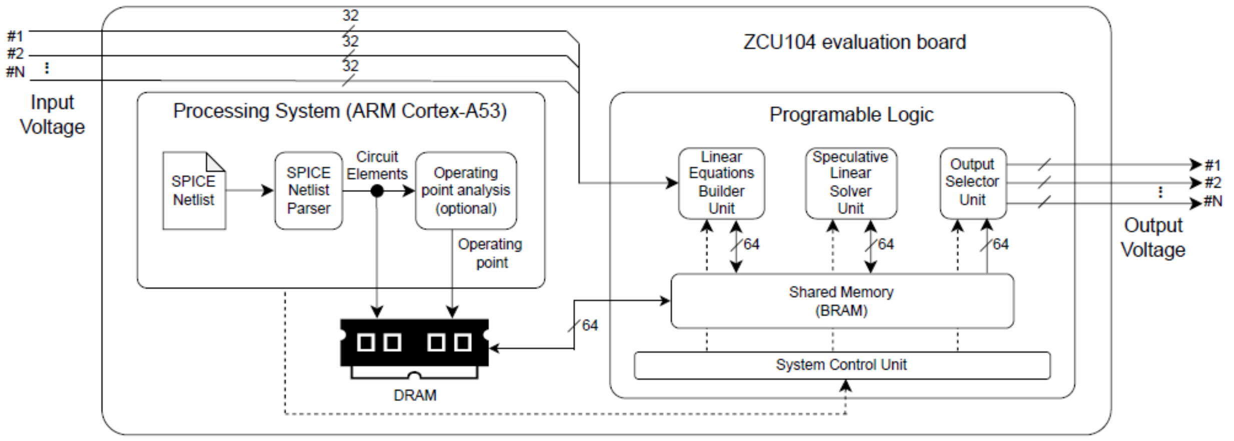

Fig. 1. System architecture of the speculative execution system

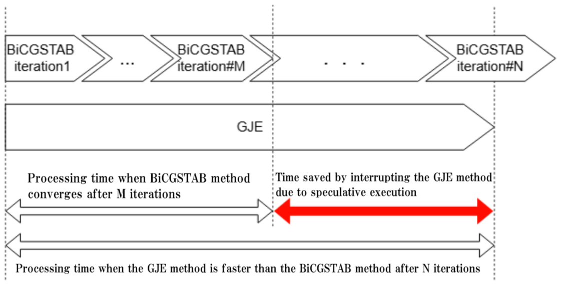
BiCGSTAB iteration1
...
BiCGSTAB iteration#M
. . .
BiCGSTAB iteration#N
GJE
Processing time when BiCGSTAB method converges after M iterations
Time saved by interrupting the GJE method due to speculative execution
Processing time when the GJE method is faster than the BiCGSTAB method after N iterations

Fig. 2. Overview of speculative execution using BiCGSTAB and GJE methods

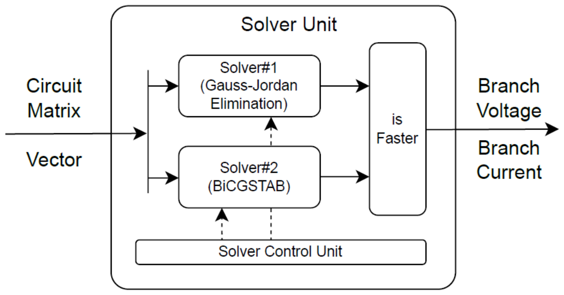

Fig. 3. Solver unit used in the speculative execution system

where k is the iteration count, offering efficiency for large matrices but with convergence depending on problem characteristics. Since iterative methods can diverge, combining them with direct solvers improves simulation stability.

Considering these characteristics, we propose a speculative execution system that runs the BiCGSTAB method in parallel with the GJE method. The speculative execution technique, illustrated in Figure 2, follows these principles:

Parallel Execution: Both the GJE solver and BiCGSTAB solver are executed simultaneously from the start of the computation.

Dynamic Selection: The result of the faster solver is adopted; the other is aborted. If the BiCGSTAB solver converges within a small number of iterations (denoted as M in Figure 2), and is faster than GJE, GJE is terminated early, reducing execution time.

Speculative execution allows the solver to dynamically choose the fastest solution method, regardless of matrix size or properties, consistently accelerating electronic circuit simulation.

C. *Architecture of the Speculative Execution System*

This section details the architecture of the proposed speculative execution system (see Figure 3), which comprises the key following components:

GJE Unit: Triggered by the Solver Control Unit, this unit runs the GJE algorithm (Algorithm 1) and sends its solution to the Dynamic Selection Unit. If the BiCGSTAB Solver Unit finishes first, the Solver Control Unit halts the GJE Unit.

BiCGSTAB Unit: Activated by the Solver Control Unit, executes Algorithm 2 and forwards its solution to the Dynamic Selection Unit. If the GJE Solver Unit finishes first, BiCGSTAB is halted by the Solver Control Unit.

Dynamic Selection Unit (isFaster): This unit continuously monitors the completion status of both the GJE and BiCGSTAB solvers in real time. It adopts the solution from the solver that finishes first and notifies the Solver Control Unit about the completion of the computation.

Solver Control Unit: Manages all solver units, speculative execution, and termination. Upon completion signal from the Dynamic Selection Unit, halts all solver operations.

Algorithm 1: Gauss–Jordan Elimination[4]

```
1: $C_0 = A|b$
2: for $k \leftarrow 1$ to $N$ do
3:     partialPivoting($C_k, k$)
4:     $M = C_{k-1}(k,k)$
5:     if $M == 0$ then
6:         continue
7:     end if
8:     for $i \leftarrow 1$ to $N$ do
9:         $L = C_{k-1}(i,k)$
10:        if $k == i$ then
11:            $C_k(k,\ldots) = C_{k-1}(k,\ldots)$
12:        else
13:            $C_k(i,\ldots) = MC_{k-1}(i,\ldots) - LC_{k-1}(k,\ldots)$
14:            $C_k(i,\ldots) = \frac{1}{2^x\ estimate\ of\ M} C_k(i,\ldots)$
15:        end if
16:    end for
17: end for
```

Algorithm 2: BiCGSTAB

```
1: $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$
2: $\mathbf{r}' = \mathbf{r}$
3: $\mathbf{p} = \mathbf{r}$
4: stop_tol = $(\mathbf{b} \cdot \mathbf{b}) \times$ tol_error$^2$
5: if $(\mathbf{r} \cdot \mathbf{r}) \leq$ stop_tol then
6:     $\mathbf{x}_{\text{next}} = \mathbf{x}$
7: else
8:     for $k \leftarrow 0$ to $N-1$ do
9:         $dotR = (\mathbf{r}' \cdot \mathbf{r})$
10:        $\alpha = dotR/(\mathbf{r}' \cdot \mathbf{Ap})$
11:        $\mathbf{s} = \mathbf{r} - \alpha\mathbf{Ap}$
12:        $\omega = (\mathbf{As} \cdot \mathbf{s})/(\mathbf{As} \cdot \mathbf{As})$
13:        $\mathbf{x} = \mathbf{x} + \alpha\mathbf{p} + \omega\mathbf{s}$
14:        $\mathbf{r} = \mathbf{s} - \omega\mathbf{As}$
15:        if $(\mathbf{r} \cdot \mathbf{r}) \leq$ stop_tol then
16:            break
17:        end if
18:        $\beta = \alpha(\mathbf{r}' \cdot \mathbf{r})/(\omega \times dotR)$
19:        $\mathbf{p} = \mathbf{r} + \beta(\mathbf{p} - \omega\mathbf{Ap})$
20:    end for
21:    $\mathbf{x}_{\text{next}} = \mathbf{x}$
22: end if
```

III. Implementation Methodology

This section describes the FPGA-based implementation of the proposed speculative execution system. Section 3.1 introduces the implementation environment, while Sections 3.2 and 3.3 detail the optimization techniques applied to the solvers.

A. Implementation Environment

The proposed system was implemented on AMD's ZCU104 evaluation board, and its implementation environment is summarized in Table I.

Double precision was used to ensure high numerical accuracy, which was required for simulating the evaluation circuits. The development process primarily relied on high-level synthesis (HLS) using Vitis HLS, generating hardware from C/C++ source code.

TABLE I
Implementation Environment

Component	Specification
FPGA	Zynq UltraScale+ XCZU7EV-2FFVC1156 MPSoC
Development Tools	Vitis IDE 2023.1, Vivado 2023.1, Vitis HLS 2023.1
Clock Frequency	100 MHz
Data Type	Double-precision floating-point (double)

B. Implementation and Optimization of the GJE Solver Unit

To optimize the GJE solver for FPGA, we used a modified Gauss–Jordan Elimination (GJE) method that reduces division operations[4]. The solver employs a triply nested loop (see Algorithm 1), with pipeline directives applied to the middle loop (lines 8–16), which provided better performance than pipelining the innermost or outermost loops. Additional optimizations applied to the GJE solver include:

Algorithm Acceleration: The solver supports linear systems up to a fixed maximum matrix size, set at synthesis. When solving smaller matrices, repeated size checks in loop iterations degrade performance. To address this, we fixed the loop count to the maximum size and used **continue** statements to skip unnecessary computations, enabling efficient execution for matrices of sub-maximal matrix sizes.

Insertion of Pipeline Directives: Pipeline directives such as `#pragma HLS PIPELINE II=N` were used to pipeline loops, where N (an integer) specifies the Initiation Interval (II). This optimization enables trade-offs between latency and resource usage.

C. Implementation and Optimization of the BiCGSTAB Solver Unit

The BiCGSTAB solver (see Algorithm 2) repeatedly performs matrix-vector multiplications and inner products, with matrix-vector multiplication being the most computationally intensive step. To accelerate this process, pipeline directives were introduced into the matrix-vector multiplication function, enabling parallel execution. The main optimizations are summarized as follows:

Pipelining of Matrix-Vector Multiplication: Pipeline directives were inserted into the matrix-vector multiplication to reduce the processing time of each iteration.

Efficiency of Convergence Check: The residual norm is calculated at the algorithm's start, thereby avoiding unnecessary iterations when the input remains unchanged or the solution already satisfies the convergence threshold.

Improved Stability: To prevent divergence caused by division by zero in BiCGSTAB, a small perturbation is added to the divisor when it becomes zero. In this study, 10^{-8} is used; however, the optimal value may vary depending on the problem characteristics. Although this may slightly increase the number of iterations, convergence is generally maintained.

Initial Guess Strategy: The solution from the most recent speculative execution system is employed as the initial guess. Starting from an estimate closer to the final solution reduces the number of iterations required.

IV. Evaluation

A. *Evaluation Methodology*

To analyze resource usage and latency with the matrix size for the GJE and BiCGSTAB solvers, we conducted the following evaluations:

Performance Evaluation of Individual Solvers: We implemented GJE and BiCGSTAB for matrix sizes 20×20, 40×40, 80×80, and 160×160. For each size, we measured latency and resource usage to assess solver efficiency across different dimensions.

Effectiveness of the Speculative Execution System: We developed an electronic circuit simulator on the target FPGA to evaluate how combining GJE method and BiCGSTAB method via speculative execution reduces computation time in circuit simulations. Note that real-time capability such as input signal sampling rate synchronization with simulator operation were not evaluated in this study.

B. *Evaluation Results: Resource Usage and Latency of Individual Solvers*

First, to independently evaluate each solver, we varied the matrix size from 20×20 to 160×160, measuring latency and resource usage. For FPGA implementation, we adjusted the Initiation Interval (II) for each solver as described in Sections 3.2 and 3.3, aiming to select the fastest solver whose resource usage remained below 50%. Table II summarizes the results for GJE and BiCGSTAB at each matrix size.

TABLE II
Latency and resource usage of each solver

Matrix Size	Solver	Latency[1] (unit: cycle)	Resource usage (unit: piece)				II
			BRAM	DSP	FF	LUT	
20×20	GJE	8,691	84	294	21,130	29,266	16
	BiCGSTAB	1,447	42	605	130,843	76,366	1
40×40	GJE	33,751	164	574	40,298	55,787	16
	BiCGSTAB	2,800	82	605	219,881	108,280	2
80×80	GJE	161,791	324	756	68,996	81,124	20
	BiCGSTAB	6,999	152	465	110,720	78,484	8
160×160	GJE	6,301,633	94	38	21,247	15,638	242
	BiCGSTAB	15,719	266	550	160,404	114,994	14

[1]The latency of the BiCGSTAB solver is the latency per iteration and does not include memory initialization time within the solver.

From the results, the GJE solver's latency increased sharply with matrix size, especially between 80×80 and 160×160. In contrast, BiCGSTAB solver latency per iteration rose more gradually. However, total BiCGSTAB latency depends on the required number of iterations, which varies with matrix properties and initial values. Thus, comprehensive evaluation requires individual verification.

Regarding resource usage, the GJE solver generally required more resources as matrix size increased, but for 160 × 160 matrices, usage decreased due to FPGA resource constraints limiting optimization. Reducing the initiation interval (II) caused resource usage on the ZCU104 to exceed 100%, making implementation infeasible. In contrast, the BiCGSTAB solver's resource usage increased steadily with matrix size, and even at 160 × 160, further optimization was possible.

C. *Effectiveness of Speculative Execution*

To evaluate the effectiveness of speculative execution, we implemented the circuit simulator (Figure 1) on a ZCU104 board and evaluated its computational efficiency using several benchmark circuits. For simplicity, external input signals (Input Voltage in Figure 1) were internally generated by the FPGA at each simulation step. We measured the average latency from forming linear equations to solving them at each simulation time step using an embedded runtime debugger.

TABLE III
Resource usage for the entire electronic circuit simulator with a matrix size of 20x20 (unit: piece)

BRAM	DSP	FF	LUT
191(31%)[2]	850(49%)	174,128(38%)	159,322(69%)

Due to FPGA resource constraints, the simulator was designed for 20×20 matrices. Table III shows that LUT usage reaches about 70% for this size, synthesized with Vivado. As a result, larger matrices—which might further improve performance—were not implemented.

D. *Benchmark Circuits*

Two types of circuits were used for evaluation:

2-stage Cockcroft–Walton circuit (Figure 4)**:** A voltage-doubling rectifier. The modified nodal analysis yields a 6×6 linear system.

6-stage RLCG circuit (Figure 5) **:** A filter with resistors (R), inductors (L), capacitors (C), and conductances (G), resulting in a 20×20 linear system.

[2]The numbers in parentheses are FPGA resource utilization rates for ZCU104.

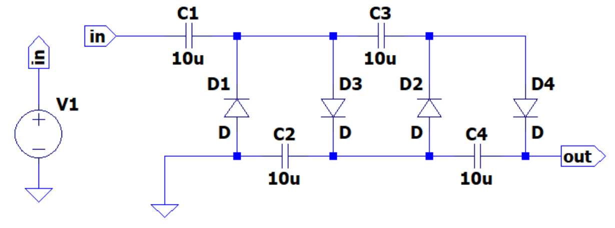

Fig. 4. 2-stage Cockcroft–Walton circuit

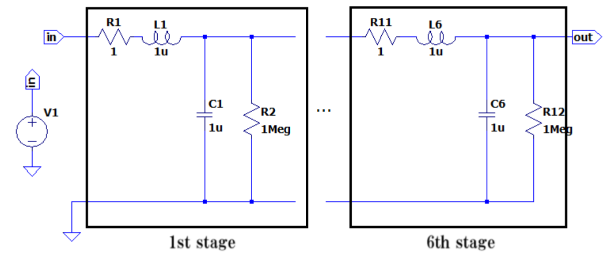

Fig. 5. 6-stage RLCG circuit

E. *Input Signals for Benchmark Circuits*

The benchmark circuits were tested with 1 kHz sine, square (rise/fall: 20 µs), and triangular waves (rise/fall: 20 µs) to evaluate the speculative execution system. The 2-stage Cockcroft circuit used a 10 V amplitude, and the 6-stage RLCG circuit used 5 V. The simulation interval was set to 1 µs.

F. *Evaluation Results*

Table IV presents the average latency of each solver for benchmark circuit simulations with test inputs. In addition, the simulation results of each benchmark circuit using the proposed solver are compared with those obtained from LTspice.

For the two-stage Cockcroft–Walton circuit (Figures 6–8), the output voltage waveforms for sine, square, and triangular input signals are shown. Similarly, for the six-stage RLCG circuit (Figures 9–11), the output voltages for one cycle of each input waveform are presented.

These results indicate that the simulation outcomes using the proposed solver are comparable to those obtained with LTspice. When the matrix size is small, the GJE solver demonstrates higher speed, and speculative execution enables automatic selection of the fastest solver.

TABLE IV
EVALUATION RESULTS OF THE BENCHMARK ELECTRONIC CIRCUIT

Benchmark Circuits	Input Signals	Solve latency (unit: cycle/solution)		
		GJE solver	BiCGSTAB solver	Speculative Execution solver
2-stage Cockcroft–Walton circuit	sine	6,923	10,673	6,924
	square	6,959	10,937	6,960
	triangular	6,932	10,296	6,932
6-stage RLCG circuit	sine	16,887	38,631	16,882
	square	16,645	36,988	15,526
	triangular	17,309	40,034	17,308

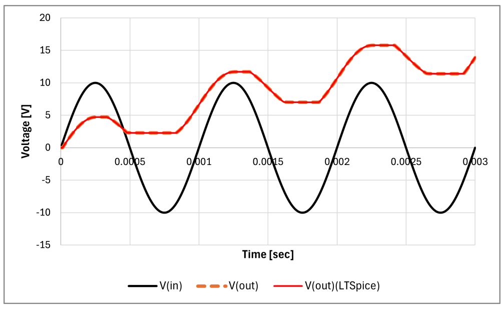

Fig. 6. Simulation results of the 2-stage Cockcroft–Walton circuit (sine wave)

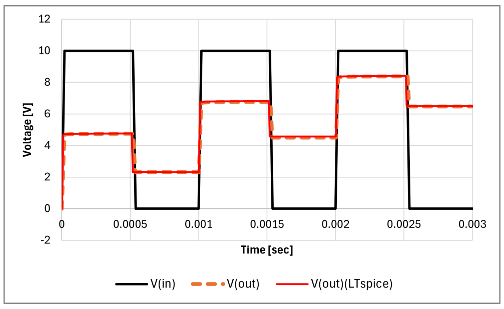

Fig. 7. Simulation results of the 2-stage Cockcroft–Walton circuit (square wave)

V. DISCUSSION

This section discusses the effectiveness of the speculative execution system based on the evaluation results.

A. *Theoretical Considerations from Solver Evaluation*

Table II shows that GJE solver latency increases rapidly with matrix size due to its $O(n^3)$ computational complexity. In contrast, BiCGSTAB grows more gradually, with per-iteration complexity of $O(n^2)$.

Therefore, for small matrix sizes (e.g., 20 $\times$ 20), the GJE solver is faster and more advantageous than the BiCGSTAB solver. However, as the matrix size increases, the latency of the GJE solver increases significantly, and the BiCGSTAB solver may become faster overall.

B. *Practical Evaluation of Speculative Execution*

For a 20 $\times$ 20 matrix, speculative execution reduced computation time by exploiting solver characteristics. Especially with large circuits and square wave inputs, BiCGSTAB required fewer iterations, highlighting the benefits of speculation. However, this approach significantly increased resource usage compared to the GJE solver. Thus, further accelerating GJE may lower resource demands while retaining high computational speed.

In fact, for both solvers corresponding to a matrix size of 20 $\times$ 20, optimizations such as memory access parallelization through data replication and loop unrolling were applied. As a result, a GJE solver with a latency of 2,291

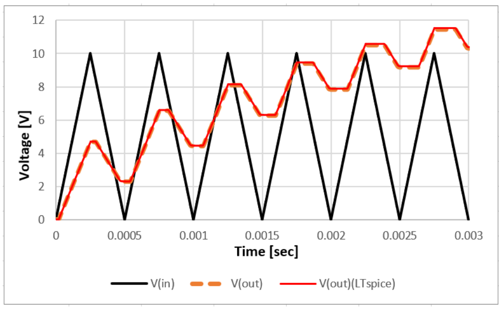

Fig. 8. Simulation results of the 2-stage Cockcroft–Walton circuit (triangular wave)

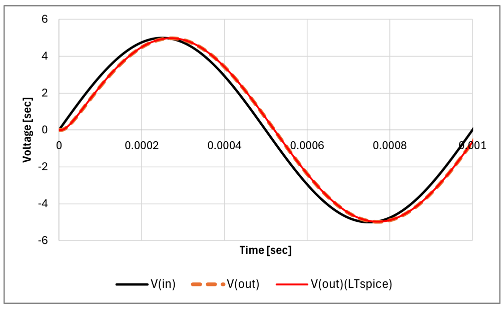

Fig. 9. Simulation results of 6-stage RLCG circuit (sine wave)

cycles was achieved. Applying similar optimizations to the BiCGSTAB method yielded a solver with a latency of 945 cycles per iteration. However, these optimizations also led to an increase in the resource usage of the GJE solver.

Overall, for 20×20 matrices, the optimized GJE solver is fast enough that speculative execution with BiCGSTAB offers limited benefit. Therefore, for speculative execution to be advantageous, BiCGSTAB must significantly outperform GJE.

C. *Differences in Speculative Execution Effectiveness Based on Circuit Scale*

Speculative execution consistently selects the faster solver, regardless of matrix or circuit size. For small circuits (e.g., 2-stage Cockcroft–Walton), GJE is faster and thus chosen. For larger circuits (e.g., 6-stage RLCG), speculative execution lowers latency compared to GJE, reducing computation time. Even with the small-scale 20×20 solver used in evaluation, its effectiveness was demonstrated.

Section 5.2 shows that when both solvers are well-optimized, GJE is faster for small matrices like 20×20, making speculative execution largely ineffective. However, although practical evaluation under these conditions could not be performed due to resource constraints, as discussed in Section 4.2, for circuits involving larger matrix sizes, the GJE method—difficult to accelerate under resource constraints—can serve as the reliable solver, while the BiCGSTAB solver—more easily accelerated—can be

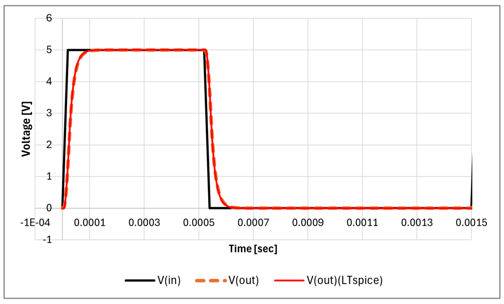

Fig. 10. Simulation results of 6-stage RLCG circuit (square wave)

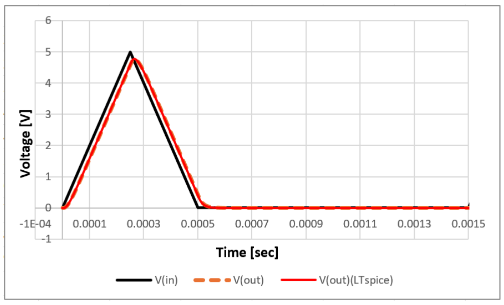

Fig. 11. Simulation results of 6-stage RLCG circuit (triangular wave)

executed speculatively. This approach could significantly reduce average latency and achieve substantial speed-up.

VI. Conclusion

We implemented a speculative execution system combining GJE and BiCGSTAB on an FPGA-based simulator, achieving continuous simulation with delays of about 6.9k cycles (69 µs) for a two-stage Cockcroft–Walton circuit and 17k cycles (170 µs) for a six-stage RLCG circuit. These results confirm high-speed performance for 20 × 20 matrices. However, for larger matrices, the resource usage of the GJE solver increases significantly, making optimization challenging. To address this issue, we integrated the BiCGSTAB solver, which exhibits more efficient resource utilization, aiming to achieve faster computation and stable convergence. Future work will focus on evaluating the solver's effectiveness for larger-scale matrices.

References

[1] M. Baghdadi, E. Elwarraki and I. Ait Ayad, "FPGA-Based Hardware-in-the-Loop (HIL) Emulation of Power Electronics Circuit Using Device-Level Behavioral Modeling," *MDPI Journal, Designs*, vol 7, no. 5:115, 2023.

[2] F. Montano, T. Ould-Bachir and J. P. David, "An Evaluation of a High-Level Synthesis Approach to the FPGA-Based Submicrosecond Real-Time Simulation of Power Converters," *IEEE Trans. Ind. Electron.*, vol.65, no.1, pp.636–644, 2018.

[3] L. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," *EECS Department*, University of California, Berkeley, 1975.

[4] J. Pierre, "Low latency and division free Gauss–Jordan solver in floating point arithmetic," *Journal of Parallel and Distributed Computing*, vol.106, pp.185–193, 2017.