2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Expedited Gate Level Verification: Unleashing the Potential of Netlist Integrated Emulation Platforms

Samhith Kumar Pottem, Vasudeva Reddy Ambati, Rahul S S, Sarang Kalbande, Garima Srivastava
Samsung Semiconductor India Research,
Bengaluru – 560048, India.

Hyundon Kim
Samsung Electronics
Hwaseong-si, Gyeonggi-do 18448, Republic of Korea.

***Abstract*-The typical design verification process in simulation, reliant on RTL, provides detailed insights into SoC chipsets but may not adequately capture the physical implementation aspects and switching activities. To address this, the industry increasingly turns to Gate Level Simulation, tailored for specific IP and test features, despite its tediousness and high time-consuming nature. Thus, innovation is crucial for efficient SoC design verification, ensuring accuracy and reliability without compromising speed. Gate Level Emulation, executed on physical hardware, offers faster validation times due to higher clock frequencies (in the Mega Hertz range), making it ideal for Gate Level Verification while maintaining reliability. In this paper, we introduce a cutting-edge GLE framework featuring custom Application Specific Integrated Circuit (ASICs) alongside an advance software test bench for design verification and debugging. Our platform is notable for its optimization techniques, which substantially enhance efficiency and benchmark its performance against two complex SoCs. Furthermore, we demonstrate this GLE platform's superiority over traditional GLS by highlighting a remarkable execution time improvement of 55X. Additionally, our test scenarios achieve around 60% greater coverage compared to conventional GLS. This robust GLE platform is implemented on the Cadence Palladium Z2 emulator.**

Keywords-Gate Level Emulation (GLE); Netlist; System-on-Chip (SoC); Gate Level Simulation (GLS); Application Specific Integrated Circuit (ASIC); Register Transfer Logic (RTL); Design under Test (DUT); Scan Dump; Design-for-Testability (DFT), Switching Activity Interchange Format (SAIF); Field Programmable Gate Arrays (FPGA); Gate Level Verification

I. INTRODUCTION

The landscape of chip development has evolved, with the ever-expanding complexity of design inherent in myriad SoC architectures leading to prolonged timeframes for bringing the chips to market. Consequently, the validation of SoC functionalities post-tape-out has become a time-intensive endeavor. Moreover, the prospect of SoC re-fabrication introduces significant bottlenecks and carries substantial financial implications for the businesses, regardless of whether vulnerabilities surface during this phase. To navigate these challenges, pre-silicon verification is of paramount importance in ensuring the functionality, performance, reliability, and adherence to standards of semiconductor chip designs before they undergo mass production. Simulators and Emulation Platforms are extensively leveraged in pre-silicon verification to play pivotal roles in rigorously validating SoC functionalities prior to chip fabrication, thereby safeguarding against potential hazards and enhancing the overall product integrity.

The typical pre-silicon verification process in simulation meticulously adheres to an RTL-based approach to analyze and validate the design features of SoC chipsets. This technique is highly regarded for its ability to offer an exceptional level of detail, providing deep insights into the complex design architectures of SoCs. However, despite the merits, this type of verification method faces vulnerabilities. Specifically, it may not adequately capture the physical implementation aspects and assess switching activity, which are crucial factors for precise power estimation within the SoC. To address these critical challenges, the industry has increasingly embraced Gate Level Simulation, an alternate abstraction methodology tailored to evaluate specific Intellectual property (IP) features embedded within SoC architectures. Nonetheless, transitioning to Gate Level Simulation poses its own set of

obstacles. Notably, the process is often fraught with tediousness and time consumption, with the verification of even a single test case stretching across multiple days within the simulation environment. This situation underscores the urgent need for innovation and efficiency enhancement in the realm of SoC design verification. As we strive towards the next frontier of technological advancement, it is crucial to develop and implement streamlined strategies that expedite the verification process without compromising on accuracy or reliability. Only through such efforts, we can fully unleash the potential of Gate Level Verification and drive forward the boundaries of modern technology.

The Gate Level Emulation approach stands as a paragon of efficiency and efficacy, primarily owing to its execution on the physical hardware infrastructure. Unlike simulations, which are confined to operate at lower frequencies in the Hertz range, the Gate Level Emulation operates within the Mega Hertz spectrum range. This significant contrast in the clock frequency translates into faster execution and validation times, making it a compelling choice in the Gate Level Verification arena. Furthermore, the emulation's resemblance to the real silicon ensures that verification outcomes are not only swift but also remarkably reliable.

We have previously proposed a distinctive GLE methodology employing FPGA emulator hardware, which was published at the DVCon India Conference 2023, as referenced in[1]. While this implemented approach has effectively tackled a significant issue in Gate Level Verification i.e. the duration needed to validate each scenario, yet it has also introduced some drawbacks. Notably, employing this GLE platform on FPGA hardware has led to substantial overheads, including prolonged compile times and reduced performance during runtime, particularly evident in complex SoCs housing billions of gates. The authors of[2] presented a vector-mode based GLE intended for improving DFT functionalities. However, their assessment was restricted to DFT features and smaller SoCs with around 50 million gates.

In this paper, we propose a Novel GLE Platform utilizing specialized hardware components, including custom ASICs, in conjunction with comprehensive software test bench for design verification, debugging, and analysis. Furthermore, a series of optimization techniques and performance strategies are introduced to establish a highly efficient Gate Level Emulation Platform. Additionally, we will conduct a comparative analysis of this efficient GLE compiled database across a pair of intricate SoCs (An Automotive chipset and an Exynos Premium Mobile SoC). We will also showcase a range of application scenarios validated on the GLE, illustrating its efficacy along with the greater verification coverage compared to traditional GLS.

II. GATE LEVEL NETLIST

A Netlist is a fundamental concept in electronic design, serving as a foundational sketch of a circuit's connectivity. It outlines the interconnections between various electronic components, such as transistors, resistors, and capacitors, without delving into their physical layout. Essentially, a netlist provides a schematic overview of how components are linked together within a circuit, playing a crucial role in the design and analysis of electronic systems.

Similarly, a Gate Level Netlist encapsulates a circuit's blueprint at its lower abstraction level, detailing its structure using logic gates. It represents the interconnections between components using logic gates and is typically generated through logic synthesis. Therefore, RTL emulation precedes synthesis, whereas Gate Level Emulation follows synthesis, focusing on the finalized gate-level design.

III. GLE IMPLEMENTATION

A. Implementation Prerequisites

To compile design files using this Incisive compiler, several prerequisites typically need to be met. These prerequisites ensure that the compilation process runs smoothly and efficiently. Here are some common prerequisites:

1) Synthesizable Design Files and Compatibility Assessment
   Before initiating compilation, ensure that the design files typically written in Hardware Description Languages (HDLs) such as Verilog or Very High Speed Integrated Circuit Hardware Description Language (VHDL) are complete, error-free, and adhere to the design specifications. Conduct a thorough compatibility check to ensure seamless integration of the design files with the Incisive compiler and the emulation platform. Verify that the versions of the compiler and associated tools align with the project requirements.
2) Primitives and Library Management
   Ensure that all requisite cell libraries, Intellectual Property (IP) blocks, and associated dependencies are readily available and accessible to the compiler.
3) Memory Models and Phase Locked Loop (PLL) Management
   Memory models are usually optimized to a particular technology node cell. Ensure that all the appropriate and required memory models, PLLs are synthesizable before passing them to the Incisive compiler.
4) License Acquisition
   Verify the availability and validity of licenses required for accessing the Incisive compiler and associated emulation platform resources.
5) Resource Allocation Planning
   Strategically allocate hardware resources on the emulation platform to accommodate the anticipated design complexity and simulation workload. Evaluate memory like LSF queues, disk space, processing cores, and emulation capacity to prevent resource bottlenecks and ensure timely completion of compilation processes.
6) Robust Verification Environment
   Set up the verification environment, including test benches, analysis tools, to validate the compiled design against functional and performance requirements.

B. Implementation Flow

The compilation process comprises several stages aimed at transforming the design source files generated by design tools from the physical implementation into a compilation database that can later on be downloaded on the emulator. The structure of this compilation database varies based on the parameters set during each compilation step. Additionally, a part of this compilation process is automatically triggered in the simulator to compile the non-synthesizable aspects of the design. Later, a swapping mechanism allows for seamless switching between the simulator and emulator while the process is running.

As shown in the Figure 1, the generic GLE DB compilation flow goes as follows:

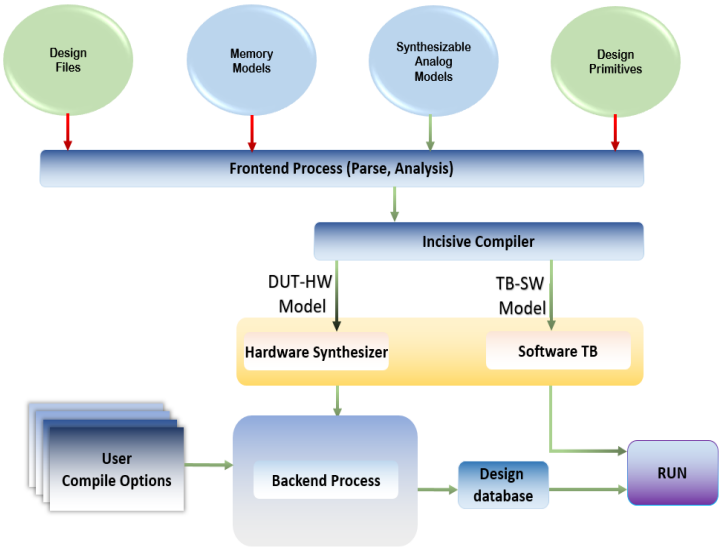

Figure 1. Typical Netlist Compilation Flow

All the required design files, module primitives, memory models, and synthesizable analog models mentioned earlier are imported incrementally by default to the Front-end process of the emulation compiler. This allows for an incremental import approach rather than importing the entire file each time, as only necessary updates trigger re-importation.

After the parsing stage is done, the Analysis, Abstraction and Normalization stage takes place wherein, the Analysis stage involves performing various checks (syntax, semantics, and perhaps custom rule checks) to ensure the source files correctness and compatibility with the verification environment. Additionally, this stage might involve optimizations or transformations to prepare the code for subsequent stages in the compilation flow.

- In the Abstraction and Normalization stage, the source design is transformed into a standardized or normalized form that is suitable for emulation. This might involve mapping complex hardware constructs into simpler, more abstract representations, resolving dependencies, and ensuring consistency across different parts of the design.
- An incisive compiler wrapper is the smartest and the intelligent tool where the source design is mapped onto the specific target resources and also involves dividing the design into smaller units for parallel processing or distribution across multiple hardware resources for the synthesis. It also setups configuring the non-synthesizable logic, test bench initialization and other parameters to be synthesized on the SW environment.
- The synthesis phase translates the Verilog or VHDL design, imported initially, into a functionally equivalent gate-level netlist or database. The Incisive compiler offers an efficient feature called distributed synthesis, particularly beneficial for synthesizing VHDL or Verilog designs comprising tens of millions or billions of gates. This feature allows for the distribution of synthesis tasks at the module level, dynamically allocating tasks during runtime to balance the workload across multiple workstations or processes. Consequently, this capability enhances the speed of runtime execution during the design synthesis process.

Subsequent to the synthesis phase, a library mapping file is generated. This file serves to establish a mapping correspondence between physical library paths and the logical directory paths.

The following step is the back-end stage, where the synthesized netlists produced by the HW synthesizer is imported to the back-end process pre-compile stage. This stage gets rid of any unnecessary tasks, thereby

accelerating the pre-compilation process and enhancing its efficiency. Additionally, various user compile commands such as assigning the net to a certain value, enforcing the signal values, implementing optimization techniques, partitioning trials, can all be passed during this stage. Subsequently, the compiler distributes the workload across multiple processing units or cores, facilitating the parallel execution of the pre-compilation process and thus resulting in faster completion. Upon completion, a unified flattened netlist is generated, which is then fed to the compile stage to generate a design database. This final design database is loaded on to the emulator hardware during run time along with the SW testbench for initialization. Furthermore, users have the flexibility to switch between the SW testbench running on the simulator and the DUT during runtime.

1) FPGA vs ASIC Emulators Comparison

The Table I. depicts this generic GLE compilation flow discussed above and implemented on both ASIC and FPGA emulation platforms for a huge SoC design of reference around 1.5B gates.

TABLE I. FPGA vs ASIC EMULATORS COMPARISON

Emulator Type	Compilation Time (Hrs)	Operating Clock Frequency (KHz)
FPGA	24	2000
ASIC	18	600

It is evident from the data of Table I. that while the emulator clock frequency is slightly higher in FPGA emulation[1], the compilation time is significantly shorter with ASIC emulation. Furthermore, ASIC-based emulation compilers provide superior scalability, integration capabilities, and debugging support compared to FPGA-based emulation compilers.

IV. GLE Optimization Techniques

A. GLE Compile Time Optimization

In a nut shell, the netlist design files are passed from the initial import stage that goes through the front-end stage. This compilation process resembles basic RTL compilation, where the netlist file is treated similarly to an RTL file. The netlist undergoes elaboration and synthesis again during the synthesis stage. However, this generic flow results in a longer compilation time compared to RTL compilations due to the inclusion of scan and other circuitry, thus creating an overhead.

Although there is a reduction in compile time compared to FPGA emulation compilers, it still remains substantial. We conducted a thorough analysis and devised strategies to enhance it, leading to the development of smarter and more effective solutions.

1) Intermittent Flow

The Incisive compiler offers a user-friendly approach for analysing netlist files (usually obtained from physical design or physical implementation team), organizing them into a logical library, and referencing previously analysed netlist libraries during the elaboration phase. This analysed data is utilized by the compiler to extract information regarding netlist object access and design unit binding. Subsequently, once all necessary information is gathered, the netlist files are forwarded to the back-end tools for final design elaboration. This mechanism aims to simplify the compilation process for external netlists with minimal user intervention. However, it's worth noting that implementing this enhancement may result in additional compile-time overhead due to the analysis of specific netlist files and the elaboration of the required netlist hierarchy.

The Incisive compiler leverages compiled netlist information to import netlist modules in the following manner as shown in the Figure 2.

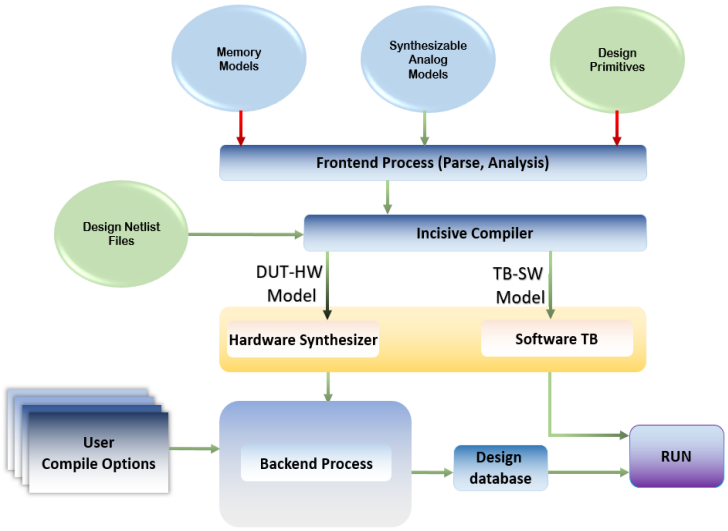

Figure 2. Intermittent Compilation Flow

During the elaboration of the RTL hierarchy, the compiler explores the netlist definitions to identify any unbound module instances within the RTL. Upon discovering a module within the netlist domain, it is labelled as an imported module, accompanied by an informational message. For all netlist modules earmarked for import, the compiler extracts the I/O interface details associated with each module. The netlist files are passed directly to the Incisive compiler stage without undergoing analysis again, thus bypassing the front-end process.

2) Explicit Netlist Flow

As the netlist files that are obtained from physical design or physical implementation team have undergone prior synthesis, there is no necessity for further analysis, elaboration, or synthesis. Consequently, they can be directly forwarded to the Synthesis stage, bypassing both the Front-end process and the Incisive compiler stage as depicted in the Figure 3.

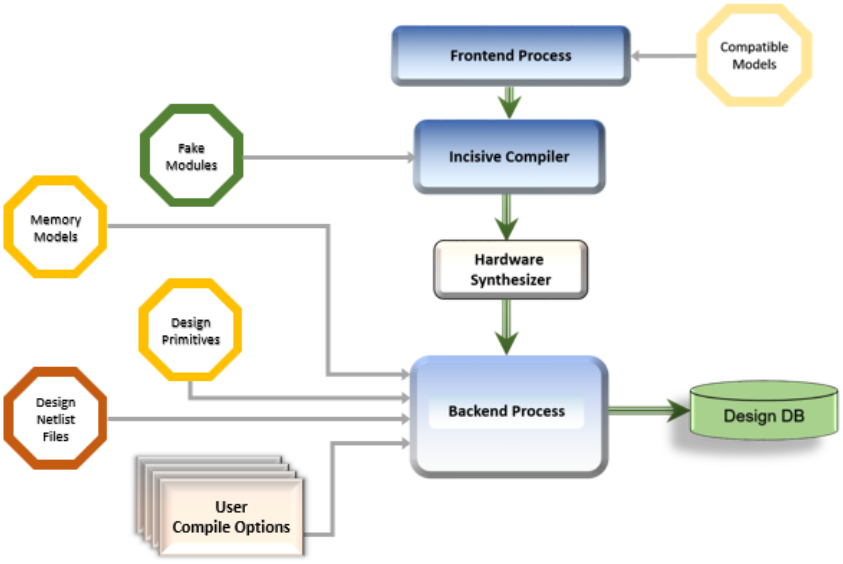

Figure 3. Explicit Netlist Compilation Flow

Therefore, netlist modules containing explicit information can be directly imported eliminating the need for compiling the netlist files. This approach, provided for backward compatibility, also serves to mitigate the overhead associated with netlist compilation. However, since the Incisive compiler does not undertake the compilation of netlist files in this context, it becomes imperative to explicitly furnish the necessary information for importing netlist modules. Furthermore, under this method, it is essential to explicitly define references to the netlist objects.

- During the elaboration stage, the incisive compiler employs a fake module to handle instance declarations before substituting them with the netlist option.
- A complete and full netlist is necessary for proper post-compilation.
- The ASIC cell libraries cited by the external netlist must be compatible with the technology being used.
- If the cell libraries are in RTL format, they need to be converted into gate-level cells through synthesis.

The Table II. illustrates the time taken by the respective compilation stages across each mentioned GLE flow:

TABLE II. COMPILATION TIME EXPEDITION ACROSS VARIOUS COMPILATION STAGES

Compilation Stage		Compilation Type		
		RTL	Intermittent	Explicit Flow
Dependency Netlist Conversion		NA	NA	00:30:00
Front End	Parse	00:32:00	NA	00:08:00
	Analysis	00:50:30	NA	00:08:00
Incisive Compiler	Elaboration	09:50:30	14:25:00	00:15:00
	Instrumentation	00:59:32	00:42:00	00:06:00
Synthesis	HW Synthesize	20:05:44	00:42:00	00:19:00
Back End	Import	03:32:25	01:34:00	00:29:00
	Compile	06:32:00	02:38:00	03:03:00
Total Time (Hrs)		42:36:00	20:01:00	04:57:00

B. GLE Performance Improvement Strategies

Similar to the compile time overheads caused by the inclusion of scan and other circuitry, there is also a significant reduction in the emulator clock frequency, creating additional runtime overheads. This issue has been tackled through the following strategies.

1) Oversampling Technology

In the fundamental mode of operation described earlier, each net in the design is refreshed at least once per two emulator clock cycles. This means that inputs are sampled at least once per two emulator clock cycles, consequently resulting in outputs changing once per two emulator cycles as well. The sampling happens once per two cycles of fastest clock to ensure the safe operation. Oversampling technology presents an effective means of adjusting the frequency of sampling these inputs and outputs relative to the emulator clock cycle.

The oversampling ratio denotes the number of emulator cycles per cycle of the fastest design clock. This ratio can be configured to values such as 0.5, 1, or any even number. Opting for a smaller value accelerates the design's performance. However, there are potential drawbacks to selecting a small oversampling ratio:

- A reduced oversampling ratio may lead to a higher capacity cost for the designs.

- Some designs may incorporate asynchronous loops, where a small oversampling ratio could adversely affect the functionality of the design.

Consequently, it is imperative to carefully consider and select the oversampling ratio to ensure the correct functioning of the design. The Figure 4. Illustrates alignment of the clocks for oversampling values of 1, 3 and 2 respectively.

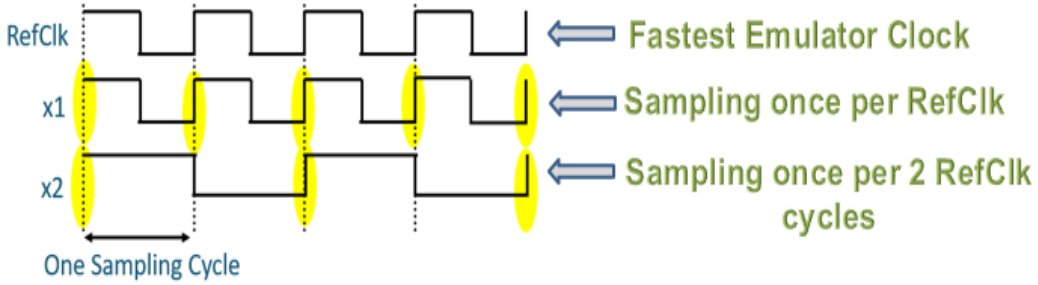

Figure 4. Alignment of Design Clocks for different Oversampling Ratios

2) Adopting Hybrid and Stub Methodology

a) Hybrid Approach

Let us consider some of our verification scenarios within our purview necessitate the validation of only select IPs within the design at the Gate Level. Consequently, we have the liberty to exclusively incorporate the design netlist files pertaining to that IPs, while retaining the remaining blocks in their RTL state before passing them to the emulation compiler. By adopting this strategy, we effectively circumvent the integration of extraneous circuitry and associated netlist overheads, thereby augmenting performance.

b) Stub Approach

Another technique involves the stubbed approach, where IPs or blocks slated for validation operate independently without relying on other blocks. Hence, by excluding these IPs or blocks before passing the design to the emulation compiler, we can achieve a remarkable enhancement in the operational speed.

The Table III. depicts the achieved operating speed across different smarter solution methods.

TABLE III. PERFORMANCE STRATEGIES

Sampling Mode	Smarter Solution Methods		
	Full Net (KHz)	Hybrid (KHz)	Stubbed (KHz)
With Over Sampling (x1)	800	1000	1400
Without Over Sampling (x2)	400	600	800

V. APPLICATIONS

The GLE platform accommodates validation of all zero-delay and unit-delay GLS verification scenarios. Notably, time-intensive scenarios such as Dynamic Power Analysis and a DFT application Scan data validation have been considered and executed on this GLE platform.

A. Dynamic Power Analysis

Dynamic Power Analysis is used to analyze dynamic power consumption due to switching activities in the design, helping to optimize power efficiency. This is performed using Switching Activity Interchange Format (SAIF) files.

SAIF files capture the switching activity of signals in a design over time, providing a detailed representation of toggle count required for power analysis.

B. Scan Data Validation

In any panic situation of SoC, the user captures output responses, and identifies any issues by comparing the observed and expected responses. It is vital in fault detection and debugging in case of a malfunction of the SoC.

VI. RESULTS

The devised GLE platform has been implemented on two distinct sets of SoCs and the collective outcomes regarding the compile time and operating speed post the adoption of the expedition strategies are outlined in the Table IV. and depicted graphically in the Figure 5.

TABLE IV. COMPARISON OF COMPILE TIME AND OPERATING SPEED ACROSS TWO DISTINCT SET OF SoCs

Verification Stage	Design Type	SoC1 (1Billion Gates)		SoC2 (2Billion Gates)	
		GLS	ASIC	GLS	ASIC
Compile Duration (Hrs)	FULL NET	18	12	34	18
	Adopting Expedition Strategies	15	8	25	13
Operating Speed (KHz)	FULL NET	0.6	1100	0.45	780
	Adopting Expedition Strategies	0.9	1400	0.7	1050

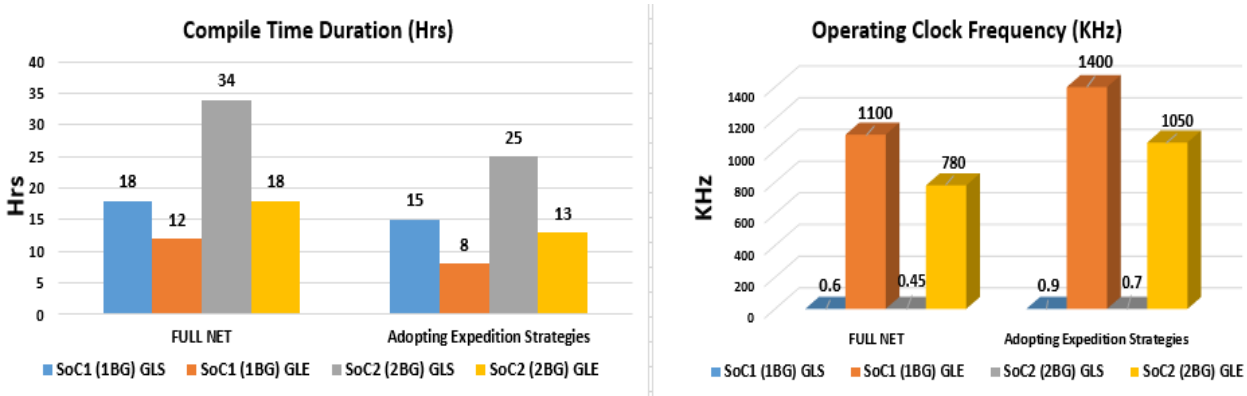

Figure 5. Compile Time and Emulator Clock Frequency post the Adoption of the Expedition Strategies

The aforementioned scenarios have been executed on the Exynos SoC2 on both GLS and this effective GLE platform, with the findings depicted in the Figure 6.

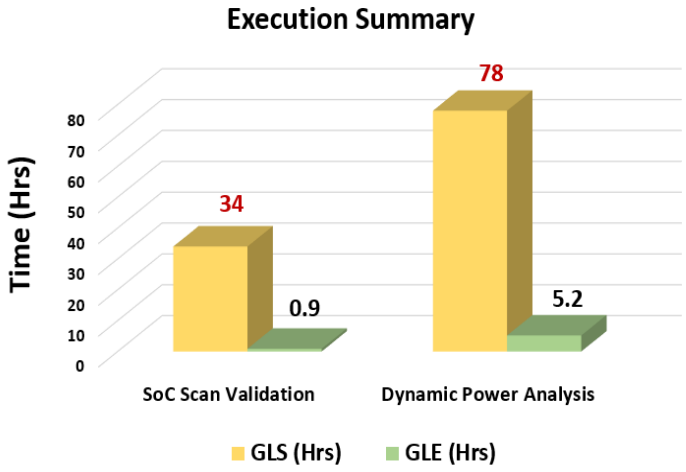

Figure 6. Scenarios Execution Summary on GLS and GLE Platforms

Furthermore, as it is commonly acknowledged, GLS verification usually commences after RTL sign-off and extends until Metal Tape-Out (MTO), with a relatively short timeframe, resulting in achieving only a minimal verification coverage of approximately 10%. If a bug arises post-tape out, it necessitates a costly SoC re-spin. In contrast, leveraging this robust GLE platform enhances the verification coverage to around 60% while also reducing overall Turn Around Time (TAT) significantly.

VII. CONCLUSION

We proposed a cutting-edge Netlist Integrated Gate Level Emulation platform, leveraging specialized hardware components such as custom ASICs alongside a comprehensive software test bench tailored for the usage across different set of SoCs. Additionally, we introduce a range of expedition techniques and performance strategies to establish a highly efficient Gate Level Emulation platform, significantly reducing compilation time and boosting runtime performance. By implementing these strategies, compilation time is slashed by approximately 7 times, and the operating speed sees a remarkable improvement of about 6 times.

Furthermore, we showcased a variety of application scenarios validated on both the GLE and GLS platforms, revealing a striking execution time difference of 50-60 times and an improved test scenarios coverage of 60% . This underscores the efficacy of our approach compared to the traditional Gate Level Simulation methods. Additionally, this implemented GLE platform has been successfully utilized on two different set of SoCs, with its performance thoroughly evaluated.

VIII. LIMITATIONS AND FUTURE SCOPE

- The verification of test scenarios is typically performed using zero-delay and unit-delay netlist files, as emulators cannot handle timing or Standard Delay Format (SDF) annotations. However, we are currently exploring the feasibility of implementing a hot-swap mechanism, wherein the simulator would handle all the timing aspects and the emulator would execute functional parts.
- Currently, the GLE platform and the Unified Power Format (UPF) integrated 2-state emulation platform have been independently devised and successfully implemented. Therefore, we are now developing a

framework to integrate this GLE with the 4-state enabled UPF power aware emulation. This integration will allow for the effective and precise validation of the design's power intent, enhancing low-power verification significantly.

- Built-in-Self-Test (BIST) is a technique utilized in electronic devices for self-assessment, eliminating the requirement for external testing apparatus. Thus, the device can self-diagnose and pinpoint faults or flaws, contributing to enhanced reliability and performance. All these various BIST scenarios can be efficiently validated using this GLE platform.

ACKNOWLEDGMENT

The authors express gratitude to the Cadence support team, India for their consistent support and guidance throughout the implementation of this efficient GLE platform.

REFERENCES

[1] Samhith Kumar Pottem, Vasudeva Reddy Ambati, Rahul S S, Sarang Kalbande, Garima Srivastava and Hyundon Kim "Netlist enabled emulation platform for accelerated gate level verification," Design and Verification Conference and Exhibition (DVCon), India, 2023.

[2] K. S. Das, P. Prakash and A. Zala, "Accelerating GLS simulation closure in DFT with smulator," 2021 IEEE International Test Conference India (ITC India), Bangalore, India, 2021, pp. 1-6, doi: 10.1109/ITCIndia52672.2021.9532898.

[3] J. Aggarwal, T. Nguyen and P. K. Gupta, "DFT+Emulation – A faster way to close verification," 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Lisbon, Portugal, 2018, pp. 11-12, doi: 10.1109/QRS-C.2018.00015.

[4] FPGA based Emulators - "Synopsys ZeBu5 user guide," unpublished.

[5] ASIC based Emulators – "Cadence Palladium Z1/ Z2 user guide", unpublished.