Mixed-Signal Functional Verification Methodology for Embedded Non-Volatile Memory Using ESP Simulation

S. Do, J. Park, D. Kim, J. Jang
Samsung Electronics, Foundry Division, Design Enablement Team
1-1, Samsungjeonja-ro, Hwaseong-si,
Gyeonggi-do, Korea, 18448

***Abstract*- IP verification is one of the key features of Foundry business. This is because only sufficiently validated IPs can guarantee the robustness of system level operation. However, mixed-signal IPs are difficult to verify and most of them are not supported by EDA tools. In this paper, we introduce our mixed-signal IP verification method using netlist-extraction, module conversion techniques, and ESP symbolic simulation. We used three types of memory to verify our methodology. One is eMRAM which is the state-of-art emerging NVM and the others are eFLASH and OTP memory which are most commonly used in Foundry business. Our proposed method enables various types of mixed-signal IPs for existing EDA tool and provides effective verification environment.**

I. INTRODUCTION

In general, IP verification is a procedure to ensure that the design intent of designer is completely reflected in IP design. However, from the Foundry perspective, IP verification also indicates mutual verification between different types of design kits, such as simulation model, schematic, and Graphic Design System (GDS), which is key design components. These three design kits should have the same functional consistency. To ensure this consistency, various verification methods[2] and supporting Electronic Design Automation (EDA) tools are developed. Considering schematic and GDS, Layout Versus Schematic (LVS) tool validates the consistency between these design kits. Besides, verification between simulation model and schematic, Simulation Program with Integrated Circuit Emphasis (SPICE) simulation is conventional and ideal method for verification.

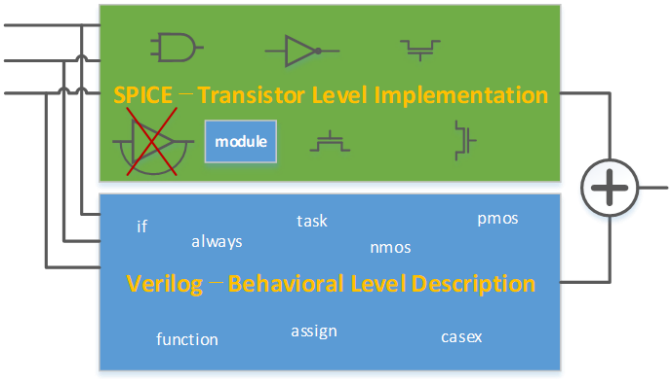


Figure 1. Concept of our proposed methodology

However, SPICE simulation has limitations in terms of simulation time. As the design size is increased, runtime of the SPICE simulation increased exponentially. Eventually, it is no longer available on time. To overcome this limitations, for Very Large-Scale Integration (VLSI) or System-on-Chip (SoC) verification, divide and conquer strategy is used. Large-scale integrated chip is composed of an enormous number of sub modules from library, such as memory, standard cell, I/O cell and custom IPs. For efficient verification, SPICE simulation is only processed for each minimum unit cells, library cell. Using this library verification process, simulation model of each IP is verified with their schematic. For system-level verification, logic simulation using simulation model which is written by Verilog or System Verilog language is used. Logic simulation encapsulate the signal values from real value to 4-state data type, which includes one, zero, unknown(x) and High-Z. This encapsulation decreases simulation runtime and increases simulation performance dramatically while maintaining the behavior of simulation model as same as schematic.

Furthermore, synthesizable digital design or Soft-IPs has many supporting EDA verification tools. Logic equivalence checking, which is usually known as EC, is supported by Cadence Conformal[12] and Synopsys Formality[13]. However, in case of mixed-signal IPs, which has analog blocks inside, conventional EC tools are not applicable. Therefore, various methods have been attempted for mixed-signal IP verification[3-7]. For certain custom design case, Synopsys ESP[11] which is built for custom design formal equivalence checking tool provides verification solution based on symbolic simulation. But its scope of supporting IPs is limited to SRAM, CAM or ROM.

In this paper, we introduce our mixed-signal functional verification methodology that replicates the ESP flow and its verification results. We used three types of memory to verify our methodology. One is eMRAM (embedded magnetic random access memory) which is the state-of-art emerging Non-Volatile Memory (NVM) and the others are eFLASH and OTP memory which are most commonly used in real Foundry business. The main contributions of this work are:

- Create custom testbench for ESP to enable symbolic simulation that is not natively supported by tool.
- Find modules that need to be modeled and create the implementation model from schematic.

The rest of this paper is organized as follows. Section 2 describes Synopsys ESP verification solution. We introduce native ESP flow with Samsung Foundry's SRAM ESP flow based on symbolic simulation and testbench creation. Section 3 describes netlist-extraction and module conversion technique and also includes several conversion examples for eFLASH and eMRAM case. Section 4 provides verification simulation results and comparison with ESP work of ARM[1] which was introduced at SNUG 2021. At last, section 5 summarizes the verification result.

II. ESP

ESP is a verification tool to view the convergence of two designs by using symbolic simulation. For conventional function simulation, a specific vector should have been created to run the simulation and verify the functional consistency. However, ESP reduce vector dependency and enable equivalent check with a digital simulator by using their own data type, "symbol". The definition of symbol is 'ESP unique value'. For static simulation, symbol is represented as unknown 'x' from equation. For dynamic simulation, such as vector base logic simulation, symbol can be either 1 or 0 at the same time. Using this type of special characteristic, ESP can determine the consistency by comparing the output value of simulation results between reference design and implementation design. In addition, ESP automatically generates testbench and performs simulations.

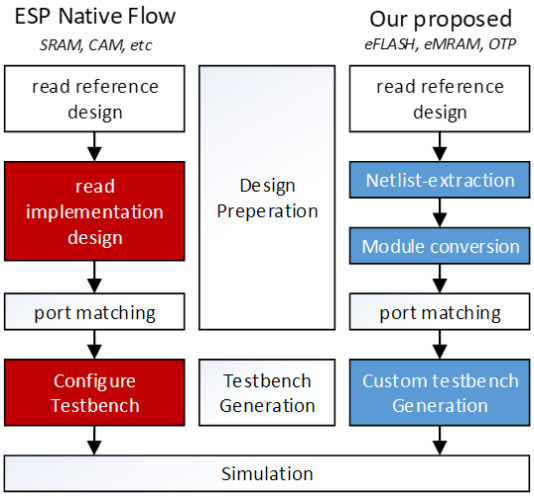


Figure 2. ESP Native & Our proposed verification flow.

A. *SAMSUNG Foundry SRAM ESP Flow*

As shown in Figure 2, ESP tool natively support SRAM. Therefore, ESP implementation for SRAM is quite simple. By setting the behavior model and SRAM circuit as reference design and implementation design, design preparation is performed. Next, setup the configure file for testbench generation according to operation characteristic and SRAM specification. Finally, tool generates testbench and runs the symbolic simulation. This ESP verification flow provides verification solution between behavior model and circuit for very large scale SRAM without SPICE simulation. Samsung Foundry has been using ESP verification flow for solid and reliable verification since 2017.

B. Symbolic Simulation

Verification flow of ESP comprises of three major steps. The first step is design preparation. During the design preparation step, read reference design and implementation design, and match the ports. Reference design is Verilog behavior model and implementation design is transistor-level design, which is transferred from the schematic to the ESP custom format. Port matching step match the ports between reference design and implementation design. From the following testbench generation step, testbench for the symbolic simulation is automatically generated considering configurations.

Testbench generation includes three steps. First step is setting the constraint pins. Second step is defining the clock pin and setting up the period, setup-delay, and hold-delay. Third step is setting up the testbench style. Supporting testbench styles includes symbolic, SRAM, and ROM. Each style has different test scenarios and test cycles. The testbench consists of different type of cycles and each cycle is divided by test clock. There are four types of test clock period: INIT (initialize) cycle, BIN (binary) cycle, SYM (symbolic) cycle and FLUSH cycle. INIT cycle is initialization clock period, which sets the initial data of input or intern nodes. BIN cycle is binary data cycle that all input data and internal node data are represent as binary (0 or 1). SYM cycle represents the symbol assigned simulation cycle. FLUSH cycle supports multi-SYM cycle operation by following the SYM cycle as per the requirement.

In the end, simulation step runs the ESP symbolic simulation and provides the simulation result with coverage reports. When the simulation reaches to the check point at the end of each cycle, simulator stores the result of each cycle. When all simulations are completed, simulator compares the stored simulation result between reference and implementation design. If there exist any mismatch point, simulator initiates simulation dump recording for debugging. ESP verification flow using symbolic simulation is appropriate verification methodology for mixed-signal IPs, but unfortunately it is not applicable for all general IPs. Because, it is digital simulator and supported SPICE model conversion is limited to few supporting devices, such as diode, capacitor, MOSFET, and resistor. Furthermore, analog circuit which has feed-back loop also cannot convert directly for digital circuit. Therefore, mixed-signal IPs which is composed of non-supported instances and analog blocks, such as eMRAM, eFLASH and OTP, are also unavailable to run ESP verification flow natively.

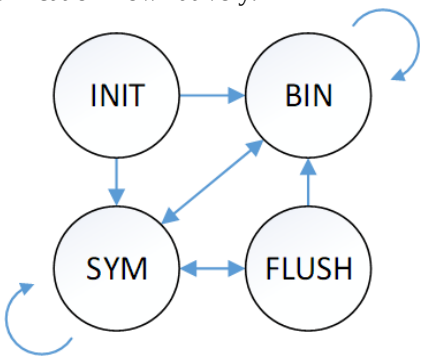


Figure 3. Cycle diagram for ESP symbolic simulation.

C. Custom Testbench Generation

ESP verification flow supports automatic testbench generation using the configuration data from simulation setting. Configuration data contains pin type information, clock period, setup/hold delay, testbench style, timescale, constraint pin information, power supply information, and so on. Using this configuration setting, testbench generator assigns symbol or static value for each pins and creates INIT, BIN, SYM, FLUSH cycle accordingly. However, automatically generated testbench using their own clock signal only supports synchronous IPs, such as SRAM even though many IPs are designed for asynchronous system, such as eFLASH. Due to this limitation, we have created custom testbench for ESP symbolic simulation, which can cover both synchronous and asynchronous designs. Custom testbench generation should be developed by the designer themselves. However, it is an inevitable solution because many custom IPs cannot be standardized.

Natively, ESP tool automatically converts circuit design into ESP custom format. However, as mentioned earlier, ESP only supports few IPs, such as SRAM, CAM or ROM. Therefore, we use Verilog versus Verilog comparison feature[9] from ESP to verify our mixed-signal IPs. To enable this ESP feature, we have to prepare circuit extracted netlist from schematic. Some schematics can be automatically converted into verilog language by using verilog native gate logic. However, most schematics cannot be simulated normally if converted automatically. In this paper, we use extracted netlist with module conversion technique to prepare implementation design for ESP simulation. Section 3 describes model perpetration step, which includes netlist extraction and module conversion.

III. MODEL PREPERATION

Due to the analog circuit embedded inside each of the mixed-signal IPs, mixed-signal IPs are designed by analog circuit designer using analog design tool, such as Cadence Virtuoso or Synopsys Custom Compiler. For symbolic simulation, ESP tool reads circuit schematic and converts it to ESP format. However, ESP does not support converting analog circuit or feedback loop for ESP format. Therefore, analog circuit must be replaced by ESP-readable format, Verilog model. Virtuoso natively supports Verilog netlist extraction function from schematic.

However, extracted Verilog netlist does not fully reflect the analog circuit design and loses analog characteristic during the netlist extraction, such as analog device characteristic or feedback loop. Therefore, we cannot use it directly for ESP simulation. To overcome this limitation, we proposed our own circuit conversion technique and use it for implementation model generation. Section 3.1 describes netlist extraction method and Section 3.2 and following sub-sections describe circuit conversion technique including various conversion examples from eFLASH and eMRAM design.

A. Netlist Extraction

Verilog netlist extraction function extract Verilog netlist from schematic from top block to device level. In case of Cadence Virtuoso, function name "Verilog Environment for NC-Verilog Integration" converts each block from schematic to Verilog netlist. After conversion, CDS directories are created for each block and each netlist inside the CDS directory represents Verilog netlist of each block. To construct the top-level Verilog netlist, we use our own script to rename each netlist file to their module name and allocate them into single top level directory. However, as mentioned earlier, some of this modules are not correctly functioning because of analog conversion limitation. Therefore, we apply module conversion technique for analog modules.

B. Module Conversion

Contrary to digital block, which is operated by 0 or 1 digitized value, analog block operation is based on continuous voltage level or current level with devices. However, converted Verilog language does not support continuous level so it loses analog characteristic. As a result, converted analog block cannot operate properly. However, if we tune or modify this abnormal Verilog block to operate correctly, entire converted Verilog netlist can be used for ESP simulation. Following four subsections (level shifter, charge pump, MRAM bit-cell, and latch) introduce each example of module conversion of our proposed methodology.

1. Level Shifter

In case of level shifter, Figure 4-a) illustrates schematic of level shifter. During digital operation, in level shifter, when voltage level of IN rise, N2 is closed, N1 is opened, and a1 is short with GROUND. However, a1 is conflicted with LOGIC H which comes through P1 when OUT is low. As a result, in digital perspective, a1 goes unknown(x) value. According to unknown value at a1, OUT goes unknown(x) value. This type of conflict also occurs even IN signal falls from High to Low. According to this conflict, converted level shifter model cannot be used as it is. To solve this conflict issue, we use R-PMOS (resistive PMOS). In case of conventional NMOS and PMOS, signal strength is preserved after the signal passed from drain to source. Thus, when same strength but different valued signal meets, value goes to unknown x. However, if one of the two signals has different signal strength, signal value follows a stronger signal. In case of R-NMOS and R-PMOS, because of resistivity, signal strength goes lower when the signal passed from drain to source. Due to this property of R-PMOS, modified level shifter functions correctly as described below.

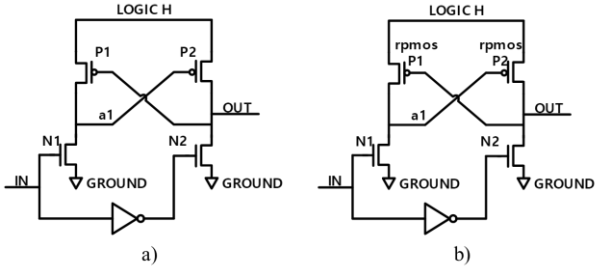


Figure 4. Schematic of level shifter. a) original, b) modified.

When IN is set to high, N1 is opened and N2 is closed. Depending on the N1 and N2 setting, a1 connect with GROUND. Meanwhile, a1 is determined as ground. Because priority of logic high from R-PMOS P1 is lower than ground, which is coming through N1. According to fixed a1 ground, R-PMOS P2 is opened and OUT goes high and R-PMOS P1 is closed. Besides, when IN is set to low, N1 is closed and N2 is opened. Depending on the N1 and N2 setting, OUT, which has high value transfer to the ground value through N2. Because the priority of ground through N2 is higher than logic high from R-PMOS P2. According to OUT value transition, R-PMOS P1 is opened and a1 goes logic high and R-PMOS P2 is closed.

Similar to this operation, when we use R-PMOS instead of PMOS, we can solve the signal conflict issue by distinguishing the signal priority. As a result, we can successfully transform analog level shifter circuit to enable Verilog simulation.

2. Charge Pump

In case of charge pump, Figure 5 illustrates 5-stage charge pump schematic. Charge pump stores input voltage from each clock signal to capacitor and creates high-voltage output. However, Verilog does not support voltage level for functional operation. Therefore, we cannot use charge pump schematic for Verilog simulation. Thus, we create behavior model, which represents charge pump operation. Following Verilog module is extracted netlist from charge pump schematic in a similar manner as illustrated in Figure 6. From an analog operation perspective, charge pump operation is considered to be complex. However, from a digital operation perspective, charge pump only passes the input voltage to output voltage, because there is no voltage level in Verilog. Thus, simply connect Vout with Vdd, and comment out other NMOS whereas CAP creates charge pump behavior model.

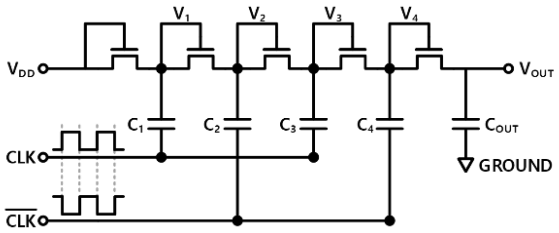


Figure 5. Schematic of 5-stage charge pump. Store input voltage for each clock and create high voltage output.

```
module Charge_Pump (Vdd, CLK, CLKB, Vout);
  inout Vdd, CLK, CLKB, Vout;
  NMOS MD1 (Vdd, V1, Vdd);
  NMOS MD2 (V1, V2, V1);
  NMOS MD3 (V2, V3, V2);
  NMOS MD4 (V3, V4, V3);
  NMOS MD5 (V4, Vout, V4);
  CAP C1 (V1, CLK);
  CAP (V2, CLKB);
  CAP C3 (V3, CLK);
  CAP C4 (V4, CLKB);
  CAP Cout (Vout, GND);
endmodule
```

a)

```
module Charge_Pump (Vdd, CLK, CLKB, Vout);
  inout Vdd, CLK, CLKB, Vout;
  assign Vout = Vdd;
Endmodule
```

b)

Figure 6. Verilog model charge pump. a) extracted from charge pump. b) behavior for charge pump.

3. Bit-cell and Cell Array Model

Figure 7 illustrates schematic of read path including MRAM bit-cell. MRAM determines bit-cell data '0' or '1' by using magnetic resistance effect from Magnetic Tunnel Junction (MTJ). At this time, voltage level of Source Line (SL) and Bit Line (BL) is analog signal. Therefore, this schematic does not support digital simulation. Also, MTJ device does not support digital simulation. Therefore, read path including bit-cell must be modeled for digital verification. Following two sub-section describe write and read operation of MRAM and also provide detailed information on cell array modeling.

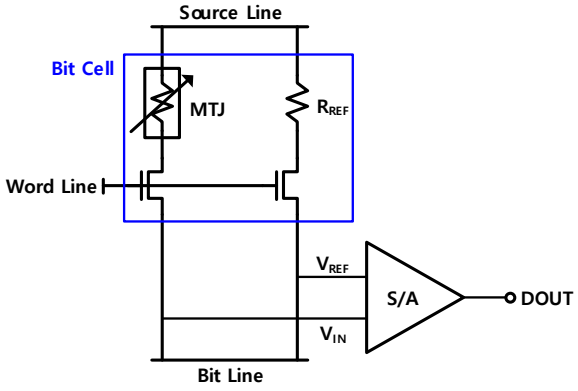


Figure 7. Schematic of MTJ Bit-Cell.

Input data for MRAM write operation is controlled by source line (SL) and bit line (BL). During 'write-1' operation, each SL and BL is set to high and low and then bit-cell is set to 1. In contrast, during 'write-0' operation, each SL and BL is set to low and high and then bit-cell is set to 0. Figure 8 illustrates MRAM cell-array modeling by Verilog language. For modeling efficiency, we pre-defined the specification of eMRAM at the top module, such as word width, column and row address width, number of mux, and number of word lines. Modeling a write operation is quite simple. Figure 8 illustrates write task operation including write operation modeling. First, select column data array from 2-D mem register array using row address. Then, write source line (SL) data into column data array considering mux and column address. Therefore, BL value should be opposite with SL value.

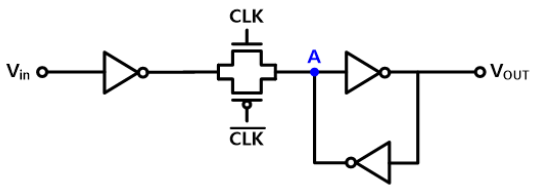
```
module Cell_Array (SL, BL, CA, RA, VREF, VIN);
  input SL[0:word_width-1];
  input BL[0:word_width-1];
  input [0:col_addr_width-1] CA; // Column Address
  input [0:row_addr_width-1] RA; // Row Address
  output VREF, VIN;
  reg [0:word_width*mux-1] mem[0:number of WL];
  task Write;
    col = mem[RA]
    for(i=0;i<word_width;i=i+1) begin
      col[i*mux+CA] = SL[i];
    end
    mem[RA] = col;
  endtask
  task Read;
  col = mem[RA]
  for(i=0; i < word_width; i=i+1) begin
    read_data = col[i*mux + CA];
    VREF[i*mux + CA]=read_data;
    VIN[i*mux + CA]=~read_data;
  end
  endtask
endmodule
```

Figure 8. Cell array model for MRAM.

Read data from MRAM read operation is delivered by voltage difference between reference resister and MTJ. Due to the different resistance between reference resister and MTJ, voltage between source line (SL) and bit line (BL) is different. As illustrated in Figure 7, read data is generated by sensing this voltage difference using sense amplifier. Therefore, cell array model should generate digital value for reference voltage (VREF) and input voltage (VIN) for read operation. If the data stored in MTJ is '0', reference resistance is higher than MTJ resistance. Since current flows from SL to BL, reference voltage (VREF) is lower than input voltage (VIN). Thus, assign 0 and 1 for each reference voltage and input voltage. Sense amplifier receives both voltage signals and prints out read data 0. Read operation modeling is similar with write operation on Figure 8. First, select column data array from 2-D mem register array using row address. Then, read data from bit-cell and assign appropriate value for VREF and VIN. When read data is 0, assign 0 for VREF and assign 1 for VIN. When read data is 1, assign opposite value.

4. Latch

Feedback loop is common scheme for circuit design. In analog simulation, feedback loop stabilizes the value of the node to maintain a certain voltage level. However, in case of digital simulation, feedback loop can crash the net value due to the signal confliction or fluctuate the net value due to the periodic signal assignment. Therefore, feedback loop should be removed from circuit during module conversion. Figure 9 illustrates simple latch schematic and its behavior model. When CLK is high, VOUT is set to VIN. When CLK is low, VOUT is maintained. However, when we run the digital simulation using schematic illustrated in Figure 9.a), a node goes unknown state and VOUT goes unknown. To solve this situation, we can set the signal priority as we performed for level shifter at section 3.2.1. However, in case of latch, it can result in distorting the situation of the actual circuit. Thus, we create behavior model for latch as illustrated in Figure 9.b) Latch behavior model assigns VIN to VOUT when CLK is high and CLKB is low. If CLKB is not low when CLK is high, assign unknown value to VOUT.



a)

```
module Latch (VIN, VOUT, CLK, CLKB);
  output VOUT;
  input VIN, CLK, CLKB;
  always @(CLK==1'b1) begin
   if ( CLKB ==1'b0 ) begin
     VOUT = VIN;
   end
   else begin
     VOUT = 1'bx;
   end
  end
endmodule
```

b)

Figure 9. a) Schematic of Latch. b) Behavior latch modeling

IV. EXPERIMENTAL RESULT

The main purpose of the ESP symbolic simulation is to check a functional equivalence between circuit and Verilog simulation model. In addition, the special feature of ESP, which is known as symbolic simulation, supports vector-less verification ideally. However, due to the functional characteristics of mixed-signal IPs, not all pins can be assigned to symbol. It depends on the IPs, but generally only functional pins are allowed to assign symbol. On the other hand, power pins, power-gating pins or other test pins are invalid to assign symbol.

During the symbolic simulation, symbol is propagated from input to output through the netlist. Thus, we can calculate the percentage of net which is covered by symbolic value during the simulation. Due to the symbolic value's characteristic, it is impossible to obtain 100% symbolic net coverage. However, the greater the percentage, the greater the reliability of verification. Not only symbolic net coverage, line coverage is also important factor for verification reliability. Line coverage represents how much the simulation checks the parts described in the verification model. Our simulation scenarios cover at least 99% of verification model for all tested IPs.

A. Simulation Setup

To evaluate our proposed verification methodology, we used three different types of embedded Non-Volatile Memories (eNVMs), such as eFLASH, eMRAM, and OTP. eFLASH and OTP are most commonly used embedded memory for System-on-Chip (SoC), and eMRAM is an emerging state-of-art embedded non-volatile memory. The symbols are assigned on many pins to improve the symbolic net coverage for ESP simulation. Also, functional operation for each IPs are implemented on the simulation scenario, such as read and write for all memories, program and erase for eFLASH, and power-gating for eMRAM. We run the simulation using Intel(R) Xeon(R) Gold 6142 CPU @ 2.60 GHz, 256 GB RAM with single core for each IP. Memory usage and simulation runtime are measured by ESP tool and runtime is based on wall time.

Table 1. Symbolic simulation coverage.

Type		Non Symbolic Nets	Total Nets	Symbolic Net Coverage	Line Coverage
eFLASH	imp	90885	259865	65.03%	99.71%
	ref	3178	3869	17.86%	
eMRAM	imp	450248	948651	52.54%	99.81%
	ref	52855	86199	38.68%	
OTP	imp	740523	1206951	38.65%	99.27%
	ref	2228	2586	13.84%	

B. Simulation Result

Table 1 describes the ESP simulation coverage report for each IPs. This coverage report is generated by ESP tool when the symbolic simulation is completed. Type imp. represents generated simulation model from circuit using netlist extraction and module conversion technique. Type ref. represents Verilog behavior model for each IPs. Symbolic-Net indicates the net which is covered with the symbolic value at least one time during the ESP simulation.

Line coverage is simulator touched line from ref model or behavior model during ESP simulation. As shown in Table 1, symbolic net coverage of implementation model is higher than behavior(reference) model for all IPs. This is because implementation model from circuit uses bidirectional port and it makes easier to propagate symbol value through the netlist. Besides, behavior model mostly uses directional port and it restricts the symbol value to propagate from input to output only. Symbolic net coverage cannot be 100% due to the functional characteristic of IP. However, from the result of line coverage, we can check that ESP symbolic simulation is tested more than 99% of entire model for all IPs and it guarantees the reliability of verification simulation.

Table 2. ESP simulation computing power.

Memory Type	Instance Size	Memory Requirement	Runtime
eFLASH	4Mb	1.06GB	73min
eMRAM	44Mb	4.37GB	1hr 13min
OTP	32Kb	2.79GB	30min

Table 2 and Table 3 describe simulation computing power of our ESP symbolic simulation and ESP simulation of ARM which is introduced from SNUG world 2021[1]. We tested three different types of IP and the size of each IP is

different. In case of ARM, two different sizes of eMRAM are tested. It is difficult to compare the result between our simulation and ARM directly. Because we do not have detailed simulation result related to different cases of ARM, such as coverage report and symbolic simulation setting. However, when we compare the average computing power from known simulation data considering memory requirement, runtime, and memory size, our simulation is much efficient than the simulation result of ARM.

Table 3. ARM's ESP simulation computing power[1]

Memory Type	Instance Size	Memory Requirement	Runtime
eMRAM	8Mb	4GB	60min
	16Mb	8GB	~6hrs

V. CONCLUSION

In Foundry business, verification of IPs is essential to achieve the reliability of Foundry library. In addition, regarding fundamental IPs such as embedded memories, importance of this verification is even higher. However, it is difficult for embedded non-volatile memories, such as eFLASH, eMRAM, and OTP, to verify their functional operation on IP-level due to the verification tool limits. To improve the reliability of fundamental IPs of Samsung Foundry, we have proposed a mixed-signal functional verification methodology for non-volatile memory using Netlist-extraction and ESP symbolic simulation. Our proposed methodology successfully enabled mixed-signal system-level functional verification in short time with reasonable computing power. Since 2020, Samsung Foundry applied this verification methodology for eFLASH, eMRAM, and OTP including advanced technology node, such as 5 nm or 4 nm. In addition, our proposed method can expand to various types of mixed-signal IPs. Furthermore, our generated simulation model using netlist-extraction technique supports various Verilog simulators, such as Cadence Xcelium, Synopsys VCS, or Mentor QuestaSim[10].

ACKNOWLEDGMENT

We would like to express our deepest gratitude to our team leader Dr. Jongwook Kye and our group leader Dr. Taejoong Song for their support on this project. Furthermore, we would like to acknowledge our principal engineers Jaeseung Choi, Dongho Shin, Myeonghee Oh, and Hyunteak Jung for their review and comments. In addition, we are grateful to Jisoo Lee for technical support of OTP simulation. Lastly, we thank Gayana H. B. for her excellent language correction.

REFERENCES

[1] Buddhi, J. “Functional Equivalence Check of Industry’s First eMRAM.” SNUG World 2021, 22 Apr. 2021, www.snugworld2021.com

[2] Serna-M. Edgar, Morales-V. David, “State of the Art in the Research of Formal Verification”, Ingeniería, Investigación y Tecnología, Volume 15, Issue 4, 2014, Pages 615-623, ISSN 1405-7743, https://doi.org/10.1016/S1405-7743(14)70659-6.

[3] H. Du, M. Tan, X. Tian and H. Xu, "ESP verification for custom SRAM array design," 2009 4th International Conference on Computer Science & Education, 2009, pp. 1889-1892, doi: 10.1109/ICCSE.2009.5228237.

[4] Gajjar, D., eInfochips “Analog Mixed Signal Verification Methodology (AMSVM).” *Design And Reuse*, 30 Jan. 2012, www.design-reuse.com/articles/28333/analog-mixed-signal-verification-methodology.html.

[5] Ito, T. “EETimes - Analog Behavioral Models Reduce Mixed-Signal LSI Verification Time.” *EETimes*, 22 June 2007, www.eetimes.com/analog-behavioral-models-reduce-mixed-signal-lsi-verification-time/#

[6] Thomas, T., Kumar, N. and Sharath, N. 2015. “Modeling and verification of mixed signal ip using systemverilog in virtuoso and ncsim.” SILABTECH Pvt. Ltd., Bangalore, India.

[7] C. Liang, Zhou Fang and C. -. Chen, "Method for analog-mixed signal design verification and model calibration," *2015 China Semiconductor Technology International Conference*, 2015, pp. 1-4, doi: 10.1109/CSTIC.2015.7153486.

[8] “Simplifying Mixed-Signal Verification.” *Verification Academy*, Simens, Nov. 2018, verificationacademy.com/verification-horizons/november-2018-volume-14-issue-3/simplifying-mixed-signal-verification.

[9] “Comparing Verilog Versus Verilog Using ESP-CV.” *Solvnet*, Synopsys, Inc., 12 Dec. 2019, solvnetplus.synopsys.com/s/article/Comparing-Verilog-Versus-Verilog-Using-ESP-CV-1576091129565.

[10] “Running an ESP-CV Testbench With Verilog Simulator.” *Solvnet*, Synopsys, Inc., 11 Dec. 2019, solvnetplus.synopsys.com/s/article/Running-an-ESP-CV-Testbench-With-Verilog-Simulator-1576070054890.

[11] “ESP User Guide.” Solvnet, Synopsys, Inc., spdocs.synopsys.com/dow_retrieve/S-2021.06/dg/esp_olh/Content/espug/pdf/espug.pdf. Accessed 21 July 2021.

[12] “Conformal Equivalence Checker.”, Cadence Design Systems, Inc., www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/digital-design-signoff/conformal-equivalence-checker-ds.pdf.

[13] “Formality Equivalence Checking.” Synopsys, Inc., www.synopsys.com/implementation-and-signoff/signoff/formality-equivalence-checking.html.