저작자표시-비영리-변경금지 2.0 대한민국

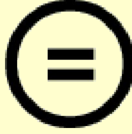
이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

Master's Thesis of Engineering Practice

Modeling and Simulation of NAND Flash Memory Sensing Systems with Cell-to-Cell V_{th} Variations

낸드플래시 메모리 셀 간의 문턱전압 변화를 반영한 센싱 시스템 모델링 및 검증 방법

February 2021

Graduate School of Engineering Practice
Seoul National University
Department of Engineering Practice

Nayoung Choi

Modeling and Simulation of NAND Flash Memory Sensing Systems with Cell-to-Cell V_{th} Variations

Prof. Jaeha Kim

Submitting a Master's Project Report

February 2021

Graduate School of Engineering Practice
Seoul National University
Department of Engineering Practice

Nayoung Choi

Confirming the master's thesis written by
Nayoung Choi
February 2021

Chair	Hoon Han	(Seal)
Examiner	Jaeha Kim	(Seal)
Examiner	Cheol Seong Hwang	(Seal)

Abstract

The sensing system in NAND flash memories is a complex mixed-signal circuit consisting of a large-scale cell array, wordline decoders, page buffers, analog/digital bit-counters, and digital sequence controllers. This paper proposes a model and simulation framework that can assess the effectiveness of various incremental/adaptive algorithms used by digital controllers for the read, program, and erase operations, while simulating the progression of individual cell threshold voltages (V_{th}) and modeling the detailed analog characteristics of the page buffers. The proposed model is written entirely in SystemVerilog, and its analog parts are described using the XMODEL primitives, which enable efficient and event-driven simulation of analog circuits. The proposed model can simulate a 40μs-long incremental step pulse programming (ISPP) sequence with the maximum loop iteration count of 4 on a 12K-bit block of single-level cells (SLC) in less than 2 minutes, and can assess the trade-offs between the programming speed and reliability as a function of the pulse step size and the impacts of the page buffer's sensing time on the final cell V_{th} distribution.

Keyword : NAND flash sensing system, mixed-signal circuit, cell threshold voltage distribution, XMODEL, SystemVerilog.

Student Number : 2018-29065

Table of Contents

Chapter 1. Introduction 1

1.1. Study Background 1

1.2. Thesis Organization 3

Chapter 2. Background 4

2.1. NAND Flash Memory Architecture and Its Operations 4

2.2. Previous Works 8

Chapter 3. Proposed SystemVerilog Model of NAND Flash Memory Sensing System 11

3.1. Cell Array Model 12

3.2. Page Buffer Model 15

3.3. Analog Bit-Counter Model 19

3.4. Digital System Model 22

Chapter 4. Experimental Results 23

4.1. SLC Program with Different ISPP Steps 25

4.2. SLC Program with Different Sensing Times 28

Chapter 5. Conclusions 30

Bibliography 31

Appendix 33

Abstract in Korean 40

Chapter 1. Introduction

1.1. Study Background

To address ever-increasing demands for capacity while keeping the costs and bit errors low, NAND flash memories use various techniques including multi-level cells and incremental programming [1], [2]. Consequently, the sensing system in today's NAND flash memories has become a complex mixed-signal circuit, consisting of memory cell arrays, wordline decoders, page buffers, analog/digital bit-counters, and digital control blocks [3]-[5]. In particular, the close interaction between the analog and digital circuits within the sensing system makes its system-level validation and performance evaluation challenging because the simulation of the analog parts requires the high precision of an analog-type circuit simulator, e.g. SPICE, whereas the simulation of the digital parts requires the high efficiency of a digital-type simulator, e.g. Verilog. While solutions exist for the co-simulation between SPICE and Verilog, the difference between the two simulators and conflicts at their boundaries often result in even slower simulation speeds [6]. This paper presents a system-level verification framework for the sensing system of a NAND flash memory, which can simulate the read, program, and erase operations of its memory cell array entirely in SystemVerilog, and estimate the resulting statistical distributions of the cell characteristics.

NAND flash memory is a nonvolatile data storage that stores information by varying the V_{th} of a floating-gate transistor device [7], and for each read, program, or erase operation, a sensing operation that checks its current V_{th} level is required. Considering an example of a charge-trap flash (CTF) cell [8], reading a cell involves applying a certain read voltage (V_R) to its gate and sensing its current, which tells whether V_R is higher than V_{th} (1) or not (0). Programming a cell involves applying a high program voltage (V_{PGM})

and raising its V_{th} by moving electrons into the charge trapping layer via a channel-hot electron (CHE) injection mechanism. However, erasing a block of cells involves applying a high erase voltage (V_{ERS}) and lowering the V_{th} by removing the trapped electrons via a hot-carrier injection mechanism. The V_{PGM} and V_{ERS} voltages are applied for both the program and erase operations as a finite-duration pulse, which is repeated with increasing amplitude until the V_{th} of the cell shifts to the desired level. To prevent some slowly-responding cells from degrading the overall programming/erasing performance, most commercial NAND flash memories set a maximum loop count for repeating these pulses and stop repeating when the number of unprogrammed/unerased cells drops below a certain reference number above zero.

The above description on the read, program, and erase operations of the NAND flash memories illustrates the complexity of its sensing systems and the difficulty of verifying them. The V_{th} of each cell is incrementally updated with a different progression and loop count, depending on its initial value and the amount of shift caused by each program/erase pulse. The total time required to program or erase a block of cells may vary depending on the V_{th} distribution of the cells. Also, the number and distribution of the cells that remain unprogrammed or unerased can vary depending on the conditions of the other cells within the same block. The complexity further increases as the digital controllers adopt adaptive schemes to determine the sensing level, pulse amplitudes, loop stopping criteria, timing conditions, etc. [1], [9]. The V_{th} distribution of the resulting cell is a complex function of the circuit/device-level characteristics of the analog circuits as well as the algorithms employed by the digital logic. Verifying whether the read, program, and erase operations can work correctly in all possible conditions is a challenging task.

As mentioned earlier, considering the cell-to-cell V_{th} variations, neither SPICE- nor Verilog-only simulation can achieve the satisfactory speed and accuracy required to verify the operations of the NAND flash memory sensing system. For example, the work in

[10] had to assume fixed V_{th} conditions and reduce the size of the cell array to mitigate the slow speed of SPICE. In contrast, the Verilog-only models in [11] could not reflect the realistic shifts in V_{th} due to the pulse duration and level, hence the resulting changes in the cell current, page buffer timing, and sequence controlled by the digital logic.

This paper proposes a model for a NAND flash memory sensing system, which can overcome all of these challenges by modeling both of its analog and digital parts in SystemVerilog. Particularly, the analog parts are modeled using the XMODEL primitives by Scientific Analog [12], which can perform the efficient, event-driven simulation of functional and circuit-level models of analog circuits within SystemVerilog without invoking SPICE. The proposed model can simulate the progression of the V_{th} values of the individual cell in a given block and analyze its statistical distribution while performing read, program, and erase operations controlled by the digital logic, which employs various adaptive and/or incremental algorithms. For instance, the proposed model can assess the trade-offs between the programming speed and reliability as a function of the programming pulse step size and analyze the impacts of the page buffer's sensing time on the final cell V_{th} distribution. The simulation of a 40μs-long incremental step pulse programming (ISPP) sequence with the maximum loop iteration count of 4 on a 12K-bit block of single-level cells (SLC) takes less than 2 minutes.

1.2. Thesis Organization

The rest of the paper is organized as follows. Chapter 2 provides the background on the NAND flash memory sensing systems and discusses the previous efforts of verifying them. Chapter 3 describes the proposed SystemVerilog model for the NAND flash memory sensing system and Chapter 4 presents the simulation results with the proposed model. Finally, Chapter 5 concludes this thesis.

Chapter 2. Background

2.1. NAND Flash Memory Architecture and Its Operations

Figure 1 shows the sensing system blocks that perform the read, program, and erase operations of a vertical-NAND (V-NAND) flash memory [13]: a memory cell array storing the data, wordline decoders driving the wordlines WL[0:n-1] and select lines GSL/ SSL, and page buffers sensing currents on the bitlines BL[0:k-1]. The analog/digital bit counters, pass/fail (PF) checker, and sequence control logic order the incremental steps of the program/erase operations. A string is the basic unit composing the cell arrays of NAND flash memories, which is basically a

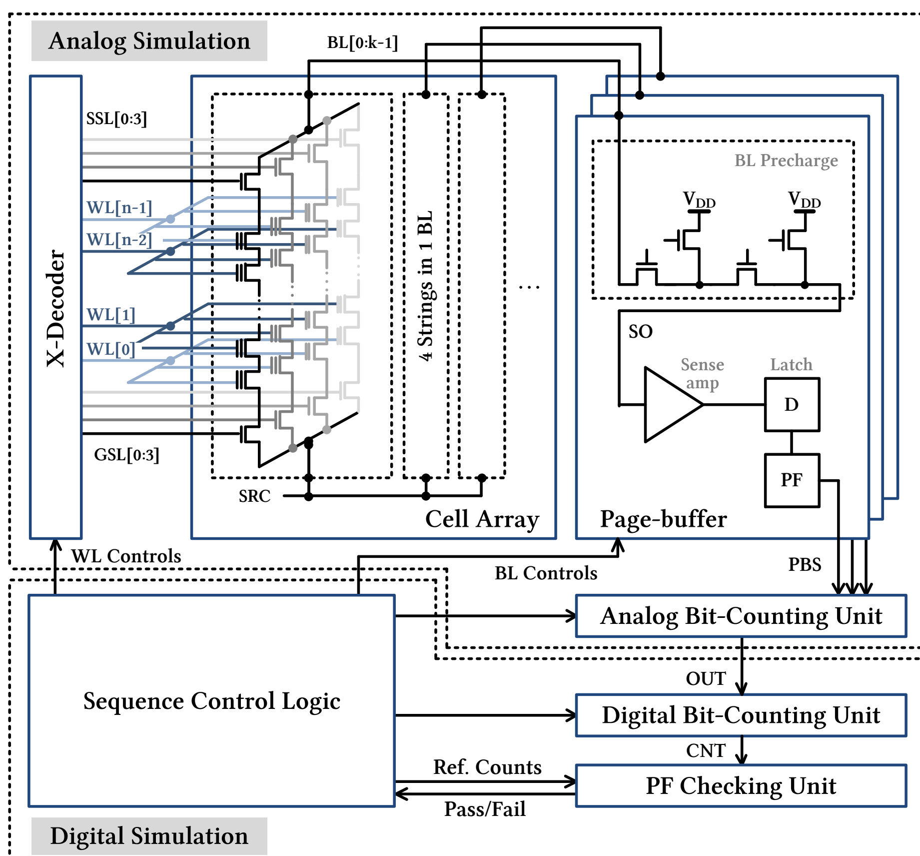
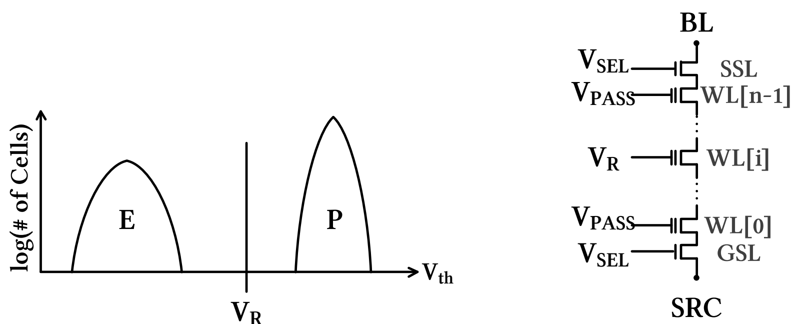


Figure 1: Overall architecture of NAND flash memory sensing system.

series stack of CTF cells and select transistors. Each flash cell on the string can store one or more bits of information by varying its V_{th}. Multiple strings may share the same wordlines (WL), string select lines (SSL), ground select lines (GSL), and bitlines (BL), as depicted in Figure 1. Depending on how they share these lines, a set of strings may be grouped into a page, block, and plane [14].

When reading a data stored in a particular cell of a particular string, the X−decoder asserts the SSL and GSL to select the desired string and applies the WL voltage of V_R to the gate of the desired cell and the WL voltage of V_{PASS} to the rest of the cells in the string (Figure2). While V_{PASS} is high enough to always turn the cell transistor on, V_R is set so that the selected cell would conduct current only when its V_{th} is lower than V_R. The page buffer circuit connected to its BL then senses the current and produces a digital output in three phases. First, the BL and SO nodes within the page buffer are pre−charged to a positive supply level (e.g. V_{DD}). Second, the BL current discharges the SO node with a multiplication effect owing to the charge sharing between the BL and SO nodes. Third, the sense amplifier detects the polarity of the SO voltage and drives the final digital output (D_{out}). For the NAND flash memories employing all bitline (ABL) current sensing [15], the same set of WL’ s and SSL/GSL’ s may drive multiple strings simultaneously, and a set of page buffers, each dedicated to a string, can produce a multi−bit digital output collectively.



(a) Read operation with V_{th} **(b) Read operation in NAND string**

Figure 2 : Read operation (a) with V_{th} distribution and (b) in NAND string.

When programming data into the cells, a high program voltage of V_{PGM} is applied to the gate of the selected cells of the selected strings. In the widely−adopted scheme of incremental step pulse programming (ISPP) [16], a series of finite−duration pulses with a gradually increasing voltage level is applied as V_{PGM}, for fast programming speed and tightly−controlled V_{th} distribution of the programmed cells. Figure 3 illustrates an example sequence of ISPP. In the first iteration loop, the initial V_{PGM} is applied to the WL of the selected cells. Then, a verification step follows, which checks the resulting V_{th} of the cells by performing a read/sensing operation. The cells that are programmed satisfactorily are marked with the program−inhibit state to avoid being over−programmed in the next loop iterations. Simultaneously, the analog and digital counter blocks count the number of un−programmed cells, and the PF checking unit checks if this number is below a predefined reference number. If not, a new ISPP loop iteration starts with the higher level of V_{PGM}. As illustrated in Figure 3(b), the ISPP scheme can achieve a tight cell

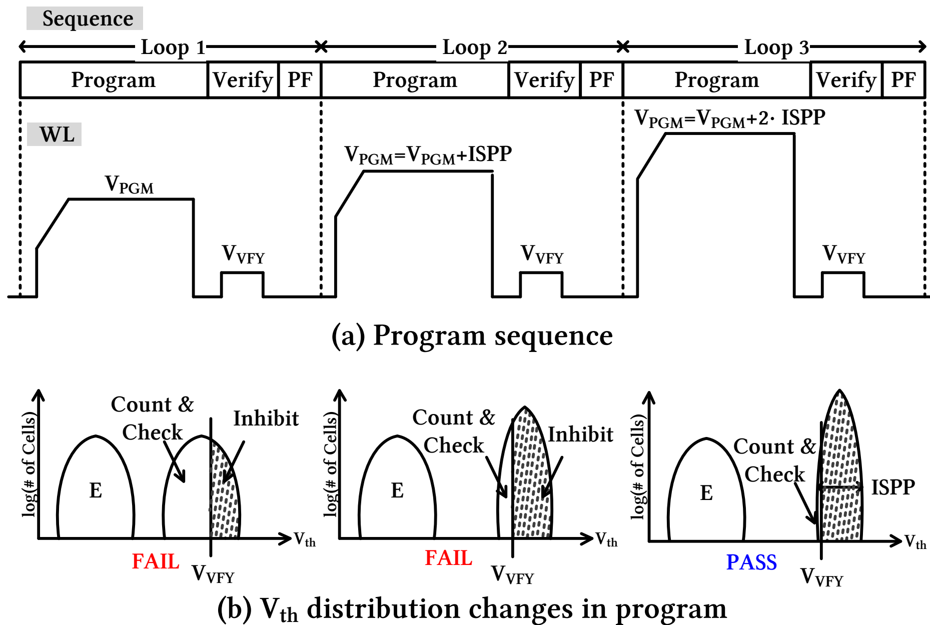


Figure 3 : (a) Program sequence and (b) V_{th} distribution of resulting cell after each iteration with the incremental step pulse programming (ISPP).

V_{th} distribution after programming by applying incrementally-increasing pulses of V_{PGM} to the un-programmed cells alone. However, its iterative nature makes it difficult to verify the correct ISPP operation against all possible conditions of the cells within the array.

When erasing data stored in the cells, NAND flash memories reset the V_{th} of all the cells within a selected block at once by setting all of its WLs to a low voltage and driving the channels of the cells with a high voltage V_{ERS}. To prevent deep erase cases which can degrade the endurance and retention characteristics of the cells, the erase operation also adopts a similar sequential algorithm, called incremental step pulse erase (ISPE). Hence, the same challenge exists when verifying the correctness of the erase operation.

2.2. Previous Works

The existing ways of verifying the operations of a NAND flash memory sensing system rely on "divide-and-conquer" approaches that simulate each part of the system with a different simulator and combine the results under certain limiting assumptions [6].

For instance, first, a so-called worst on-cell current (WOC) of a cell string is estimated using three-dimensional (3D) technology computer-aided design (TCAD) simulation [10]. This WOC is defined as the current of a string in which cells are all programmed except the last one. Second, considering this WOC as the fixed string current and modeling the BL as a resistor-capacitor circuit, the pre-charge and sensing delays of the page buffer are estimated with SPICE simulation. Third, these delays are then considered as fixed, where the correctness of the read, program, and erase operations is verified with digital system simulations (Figure 4). It is evident that assuming fixed string current and fixed pre-charge/sensing delays cannot produce diverse scenarios to fully exercise different cases of the ISPP and ISPE algorithms.

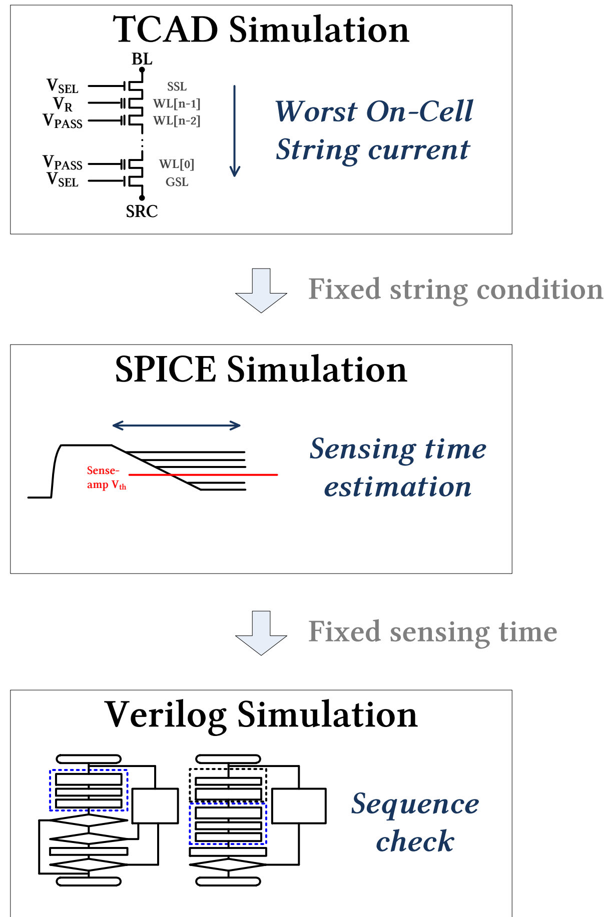


Figure 4 : Previous verification flow for sensing systems.

Although combining these separate simulations into a single, unified simulation with SPICE or HDL−SPICE co−simulators has been attempted, the slow speeds in performing high−precision simulations of large, complex circuits carrying out a long sequence of operations make it difficult to verify the operations of the sensing system across all possible V_{th} distributions and program/erase speeds of the cells.

Therefore, to fully verify the operations of a NAND flash memory sensing system, its model must satisfy the following requirements as shown in Figure 5. First, the model must include at least a block−sized cell array of which cell V_{th}s can have individually different initial values and changing rates responding to the program/erase pulses. Second, its page buffers should

produce individually different delays depending on the cell conditions of the selected strings. Finally, the simulation including the models for the analog/digital bit counters, PF checker unit, and sequence control logic must run efficiently enough to carry out a long sequence of operations, and observe the resulting V_{th} distributions of the cell.

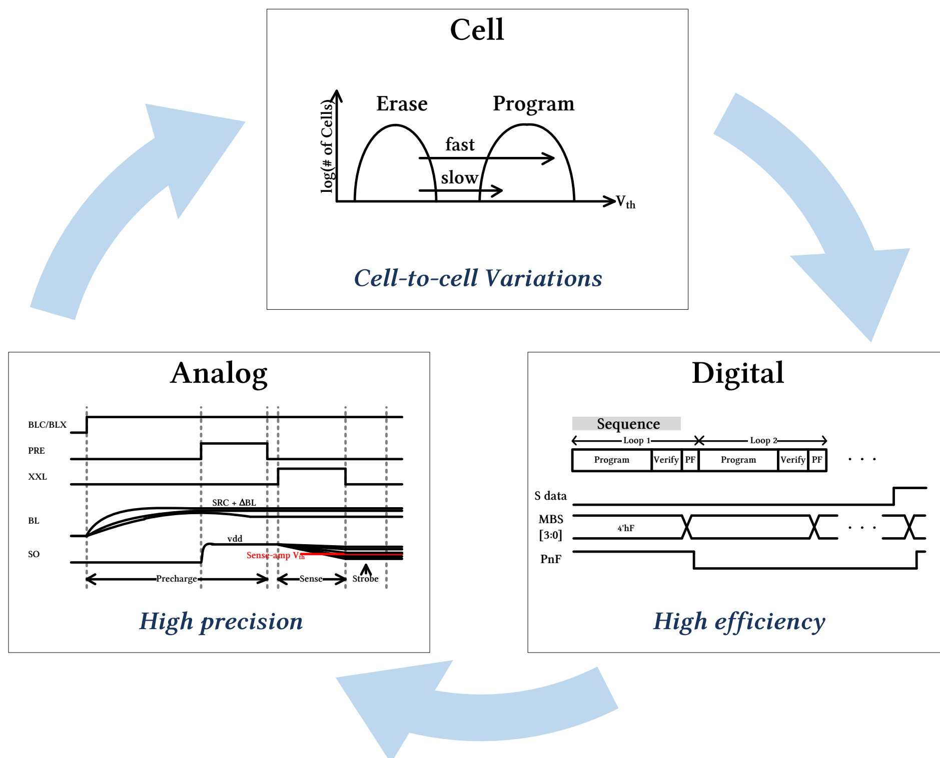


Figure 5 : Requirements for modeling NAND flash memory sensing systems.

Chapter 3. Proposed SystemVerilog Model of NAND Flash Memory Sensing System

This section describes the proposed SystemVerilog model of the NAND flash memory sensing system that meets the aforementioned requirements. First, its cell array models an entire block with at least 16 WLs, 192 BLs, and 4 SSLs, in which cells can have individually different V_{th}s. However, its simulation time increases only with the number of BLs and not with the numbers of WLs or SSLs, by modeling only the selected strings of the cell array and dynamically loading/storing the V_{th} values of the cell when the selection changes. Second, the page buffer and analog bit counter models are described with the XMODEL's circuit and function primitives, which can model the detailed analog behaviors of the SO node varying with the BL current and the bit counter output varying with the page buffer results while running efficiently within SystemVerilog. Third, the Verilog RTL models describing the digital bit counters, PF checker unit, and sequence control logic can carry out a long sequence of read, program, and erase operations, while observing the progression of the V_{th} distribution of the cell array (Figure 6).

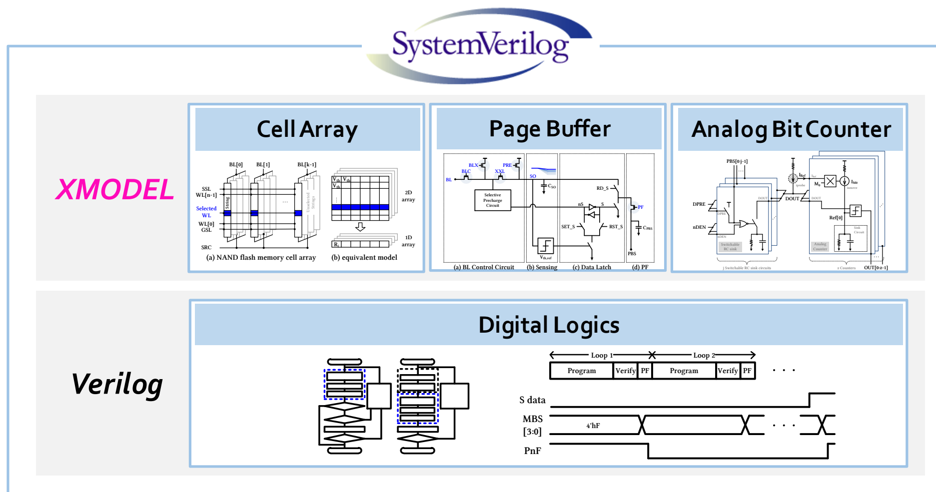


Figure 6 : Key ideas for modeling NAND flash memory sensing systems using XMODEL for the analog circuits and Verilog for the digital logics.

3.1. Cell Array Model

The cell array model has plane−block−string hierarchy. Plane is consisted with multiple blocks and each block includes multiple strings. Since the basic read, program and ereas operations in NAND flash memories are performed within the single selected block, the proposed cell array model has a separated 1−block sized register for simulation efficiency (Figure 7). Based on the block selection, the model loads the V_{th} values of the individual cell stored in a data register into the active block and if the value changes, their updated values get stored back to the V_{th} data array before the next operation.

While it is straightforward to model a cell array as shown in Figure 8(a), keeping its simulation efficient is difficult owing to the large number of instances within the array and the dense connectivity between them. Both the elaboration and simulation phases of its SystemVerilog simulation can become considerably slow as the array size increases.

Recognizing that the SSLs and GSLs select only one string at a time that drives a given BL, one can significantly reduce the complexity of the cell array model without compromising the accuracy. In other words, the proposed cell array model models only the active string driving each BL, and dynamically updates its cell V_{th} values based on the SSLs and GSLs values, as shown in

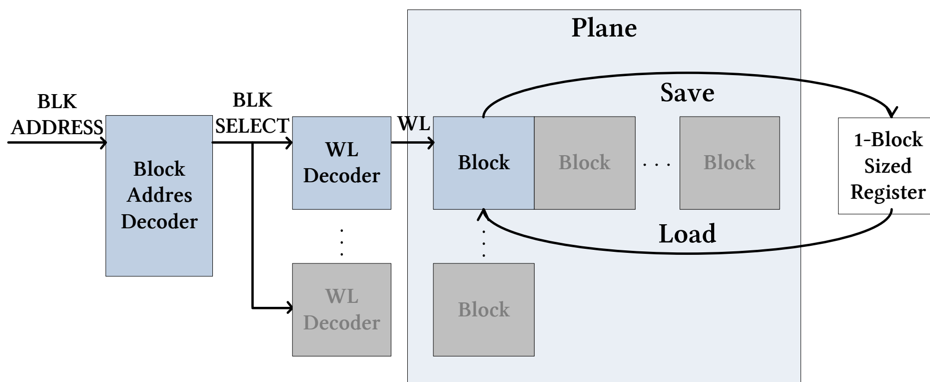


Figure 7 : The architecutre of cell array model and its save and load operations with sepereated register.

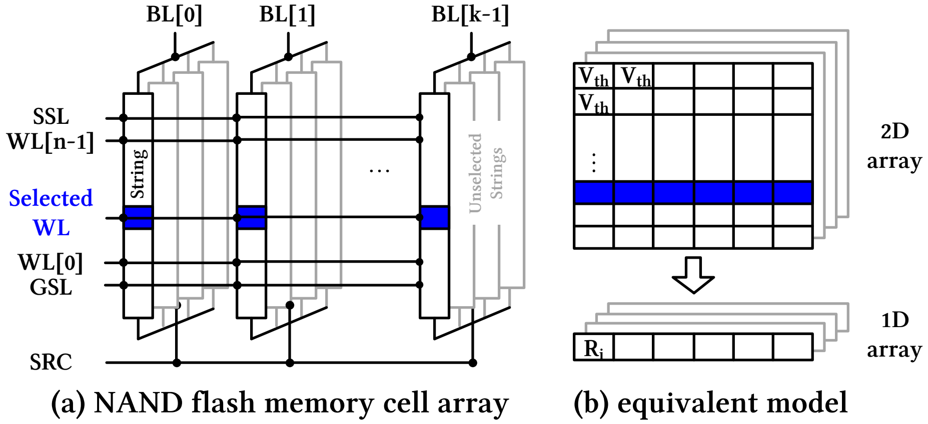


Figure 8 : (a) Cell array of NAND flash memory and (b) its equivalent model.

Figure 8(b). The V_{th} values of the individual cell are stored in a data array and the model loads the V_{th} values into the active strings when the SSLs and GSLs are switched to new values. The active strings then produce the BL currents according to their cell V_{th} values. In case of the program operations, the V_{th} values on the active strings may change, and their updated values get stored back to the V_{th} data array before the next operation.

Each active string in the cell array is modeled as a variable resistor, which has a resistance R_{string} equal to the sum of the individual cell resistances (Figure 9). Its cell resistance $R_{cell,i}$ is in turn proportional to $(V_{GS,i}-V_{th,i})^{-1}$, assuming that the cell device operates in the linear region:

$$R_{string} = \sum_{i=0}^{n-1} R_{cell,i} = \sum_{i=0}^{n-1} \begin{cases} \frac{1}{K_P \cdot (V_{GS,i}-V_{th,i})} & \text{if } V_{GS,i} > V_{th,i}, \\ \infty & \text{otherwise.} \end{cases} \quad \textbf{(1)}$$

where K_P is a technology-dependent scale factor.

Figure 10 lists the SystemVerilog pseudo-code of the string model. Each string is described as a series of two nmosfet and one res_sw primitives, which are the XMODEL circuit primitives for nMOS and switchable resistors, respectively. The resistance value is computed according to Eq. (1). The cell array model consisting of these string models runs efficiently in an event-driven fashion, generating only one output event when receiving a

new input event. The detailed SystemVerilog model for the cell array plane and block are attached in the appendix.

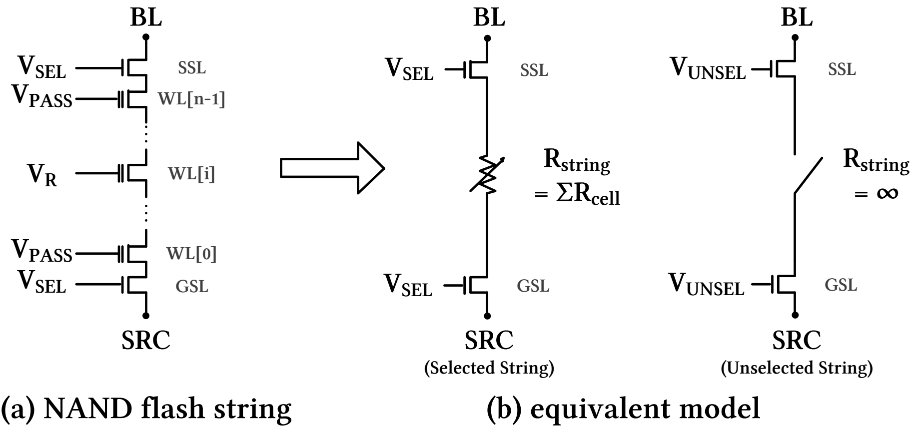


Figure 9 : (a) NAND flash memory string circuits and (b) its equivalent model.

```
module nand_str #(
     parameter num_wl = 16,
     parameter VTH_SEL = 0.5,
     parameter KP_SEL = 1e-4,
     parameter KP_CELL = 1e-4
)(
     input xreal BL,
     input xreal SSL,
     input xreal GSL,
     input xreal SRC,
     input xreal [num_wl-1:0] WL,
     input longint mem_block
);
     real r_str, r_t, v_ov;
     int i;

     nmosfet #(.Vth(VTH_SEL), .Kp(KP_SEL)) MSSL (.g(SSL), .d(BL), .b(`ground), .s(sp));
     res_sw R0 (.neg(sn), .pos(sp), .R(r_str));
     nmosfet #(.Vth(VTH_SEL), .Kp(KP_SEL)) MGSL (.g(GSL), .d(sn), .b(`ground), .s(SRC));

     always @(posedge read_flag) begin
        if(MSSL.level>0 && MGSL.level>0) begin
            r_t = 0.0;
            for (i=0;i<num_wl;i++) begin
               if (sample(WL[i])<V_PASS) begin
                  v_ov = (sample(WL[i])-getelem(mem_block,i));
                  r_t += (v_ov>0) ? 1/(KP_CELL*v_ov) : `INFINITY;
               end
            end
            r_str = r_t;
        end
        else r_str = `INFINITY;
     end
endmodule
```

Figure 10 : SystemVerilog model for NAND flash string using XMODEL primitives: *nmosfet* and *res_sw*.

3.2. Page Buffer Model

A page buffer senses the BL currents drawn by the selected strings and produces full-digital read-outs. Therefore, its operation is mixed-signal in nature and it is particularly important to model the SO node timing, which varies with the BL current level. Again, the XMODEL primitives allow us to model such detailed analog behaviors in a digital simulator, SystemVerilog.

Figure 11 shows the proposed page buffer model modeling the state-of-the-art page buffer circuits in [17]-[19] while Figure 12 lists the corresponding pseudo-codes in SystemVerilog. The set of nMOS transistors modeled with the nmosfet primitives convert the BL input current to the SO node voltage via pre-charging and discharging operation. A slice primitive then compares the SO node voltage with a reference and produces a digital output 1 or 0. The page buffer model also contains a latch for the strobe operation, which is described with and_xbit and nor_xbit (i.e., the AND and NOR gate with xbit input/output) primitives. When the strobe pulse SET_S or RST_S is asserted, the latch stores the sensed data. In addition, the page buffer includes the PF circuits for bit-counting operations, which are described later.

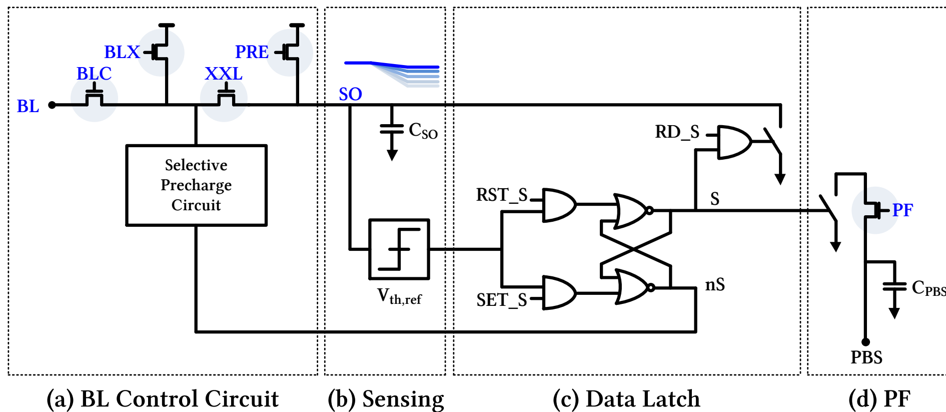


Figure 11 : Page buffer model with XMODEL primitives.

```
module page_buffer #(
  parameter real C_SO = 1e-14)(
  input xreal BL,
  input xreal BLC,
  input xreal BLX,
  input xreal XXL,
  input xreal PRE,
  input xreal PF,
  input xreal PBS,
  input xbit RD_S,
  input xbit SET_S,
  input xbit RST_S);

  xreal SO;
  xbit ilat_S, iRD_S, iPF;
                    :
  // BL control circuit
  nmosfet #(.Vth(0.5), .Kp(0.001)) MBLC (.g(BLC), .d(iBLC), .b(`ground), .s(BL));
  nmosfet #(.Vth(0.5), .Kp(0.001)) MBLX (.g(BLX), .d(vdd), .b(`ground), .s(iBLC));
  nmosfet #(.Vth(0.5), .Kp(0.001)) MXXL (.g(XXL), .d(SO), .b(`ground), .s(iBLC));
  nmosfet #(.Vth(0.5), .Kp(0.001)) MPRE (.g(PRE), .d(vdd), .b(`ground), .s(SO));

  // Sensing
  capacitor #(.C(C_SO)) C0 (.neg(vss), .pos(SO));
  slice #(.threshold(0.0)) MP0 (.in_ref(Vref), .in(SO), .out(ilat_S));

  // Data latch
  nor_xbit #(.num_in(2), .delay(0.0)) XP10(.out(S), .in({nS,iRST_S}));
  nor_xbit #(.num_in(2), .delay(0.0)) XP11(.out(nS), .in({S,iSET_S}));
  and_xbit #(.num_in(2), .delay(0.0)) XP12(.out(iRST_S), .in({ilat_S, RST_S}));
  and_xbit #(.num_in(2), .delay(0.0)) XP13(.out(iSET_S), .in({ilat_S, SET_S}));

  and_xbit #(.num_in(2), .delay(0.0)) XP14(.out(iRD_S), .in({S, RD_S}));
  switch #(.R1(0.01), R0(`INFINITY),  .ic(0))
  SW0(.neg(SO), .pos(vss), .ctrl(iRD_S));

  // PF
  switch #(.R1(0.01), R0(`INFINITY),  .ic(0)) SW0(.neg(iPF), .pos(vss), .ctrl(S));
  nmosfet #(.Vth(0.5), .Kp(0.001)) MPRE (.g(PF), .d(PBS), .b(`ground), .s(iPF));
  capacitor #(.C(C_PBS)) C0 (.neg(vss), .pos(PBS));

endmodule
```

Figure 12 : SystemVerilog model for page buffer using XMODEL.

Figure 13 shows the simulated waveforms of the page buffer' s SO node in SystemVerilog, considering the best on−cell current (BOC), worst on−cell current (WOC), and off−cell current conditions. First, the BLC/BLX transistors pre−charge BL and the PRE transistor pre−charges SO. Next, the string current drawn from the BL input starts discharging BL. When BL is discharged sufficiently, the XXL transistor may turn on, which then starts discharging the SO node. The amount of change in the SO node voltage is a function of the BL current and the ratio between the BL capacitance and SO capacitance. Figure 13 shows the SO' s discharging slope varying with the cell current conditions. The sense time must be optimized to maximize the SO difference between the off−cell current and WOC conditions.

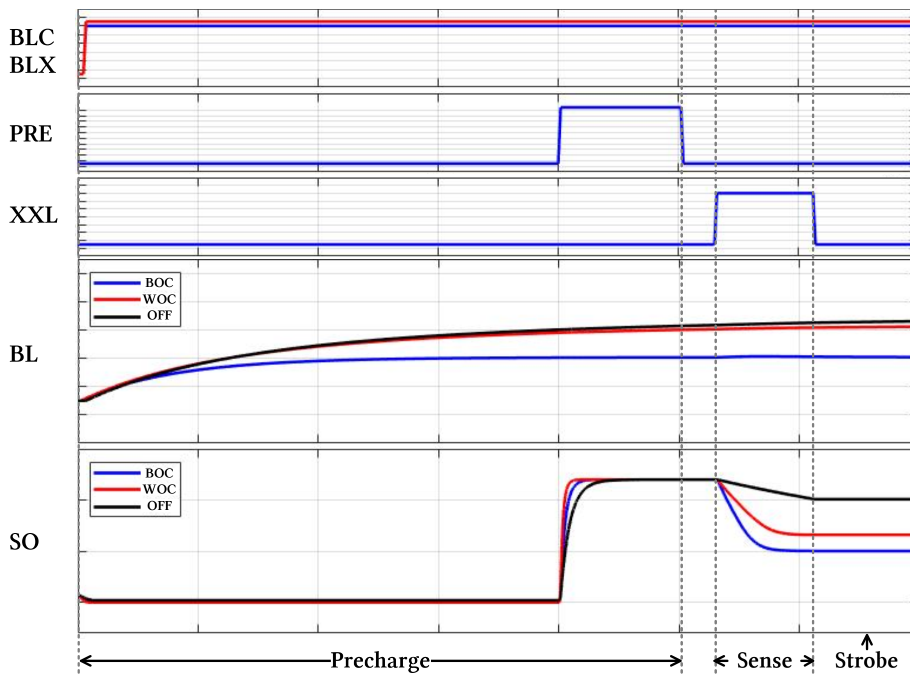


Figure 13 : Simulated waveforms of proposed page buffer model while performing sensing operations.

Figure 14 compares the simulated waveforms of a unit page buffer model in XMODEL and the simulated waveforms of a unit page buffer circuit in HSPICE. For both cases, the current conditions of the cell are modeled using an equivalent resistor and capacitor. Figure 14 shows that the results match well between the XMODEL and HSPICE. However, XMODEL was 5.8 times faster than HSPICE, thus demonstrating that the proposed page buffer model can efficiently represent the sensing operation without compromising accuracy in SystemVerilog.

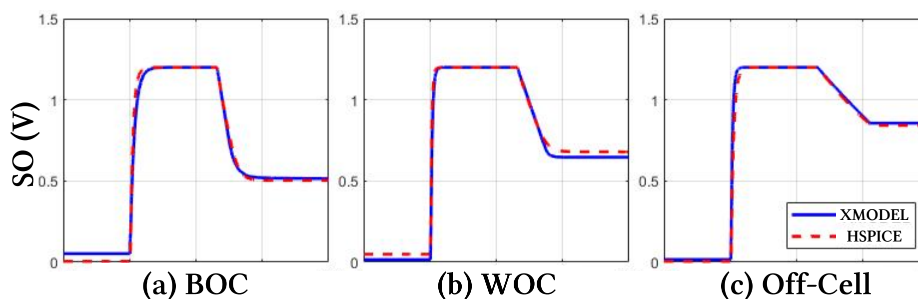
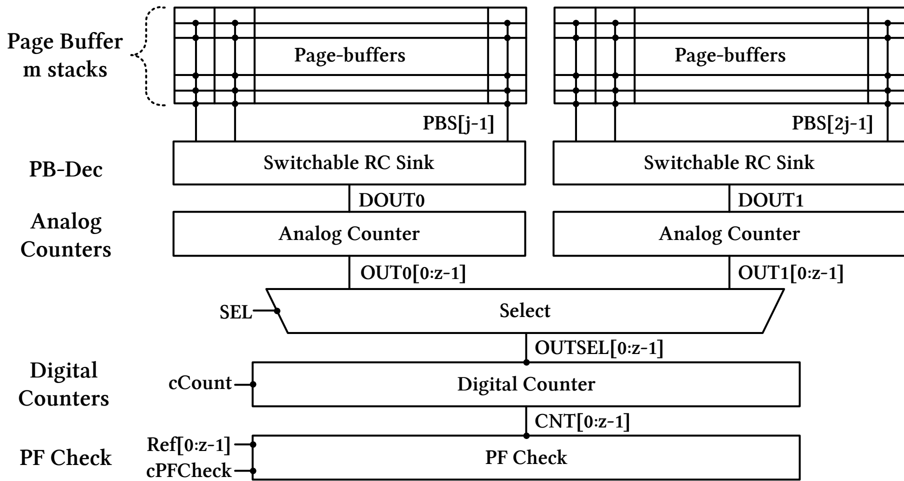


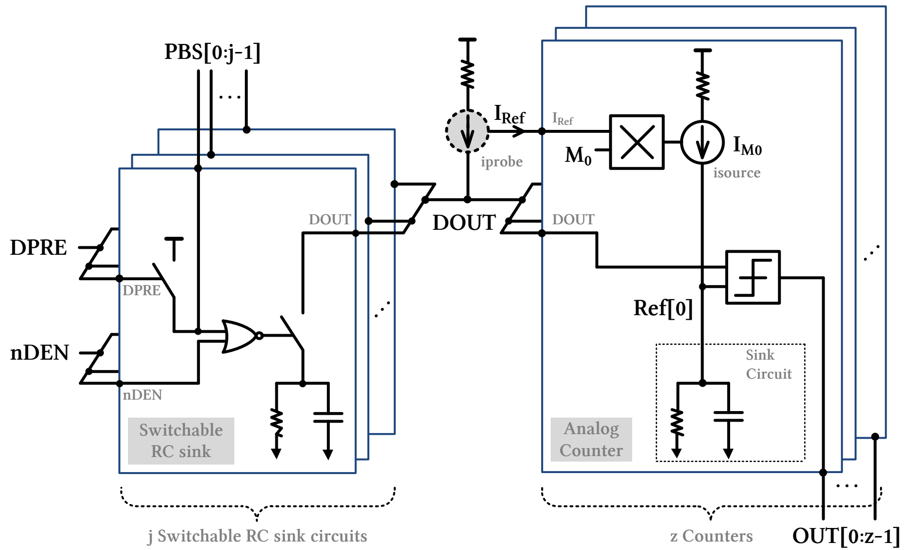
Figure 14 : Comparison between the XMODEL and HSPICE simulation results with (a) best−on−cell current (BOC), (b) worst−on−cell current (WOC), and (c) off−cell current conditions.

3.3. Analog Bit−Counter Model

Figure 15(a) illustrates the organization of the bit−counter units that count the number of failing bits during the incremental program/erase operations. The analog bit−counter counts the number of failing bits in an analog fashion by aggregating the voltages on the shared PBS lines. This so−called slow bit−counter architecture is an effective way to estimate the approximate number of failing bits with low−complexity and low−power analog circuits [13]. After each sensing operation, a set of m page buffers drives a shared PBS line with the charge indicating the pass/fail result, which develops a voltage proportional to the number of failed bits. The analog bit−counter then takes a set of j PBS lines and compresses their information into z−bit digital output using a set of current mirrors and resistor−capacitor (RC) sink circuits. The digital bit−counter then further aggregates the results from multiple analog bit−counters, producing the final count CNT[0:z−1]. Finally, the PF checker unit compares CNT[0:z−1] with a pre−determined reference and determines whether the program/erase has succeeded or not. Figure 15(b) illustrates the model of the analog bit−counter unit described with the XMODEL circuit and function primitives. For instance, the switchable RC sink is modeled with the switch, resistor, capacitor, and nor_xbit (i.e., the NOR gate with xbit input/output) primitives. The current mirror with the multiplication factor of M_0 is modeled with the iprobe, scale, and isource primitives. The output of the current mirror is fed to another RC sink to produce a reference voltage, which is then compared to DOUT by the slice primitive to determine the final digital output OUT[0:z−1].



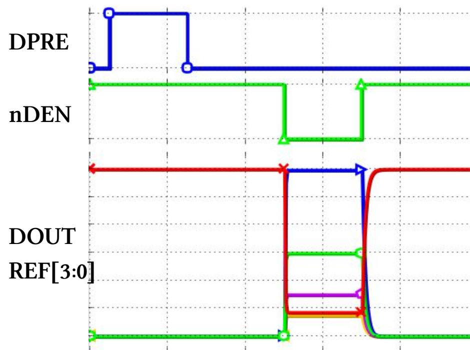
(a) Counter structure with shared PBS lines

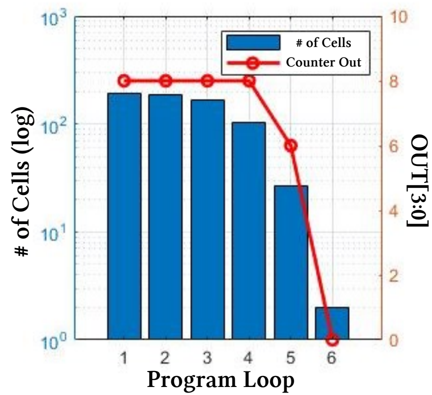


(b) Analog bit-counting unit model

Figure 15 : (a) Organization of the bit−counter units in the slow bit−counter architecture [13] and (b) the proposed model for the analog bit−counter unit.

Figure 16(a) shows the simulated waveforms of DOUT and REF[0:z−1] using this analog bit-counter model. The model essentially outputs a log function of the number of failed bits, which provides the finer resolution counting small numbers with the limited number of output bits. Figure 16(b) plots the output of the analog bit-counter changing as the program iteration progresses.





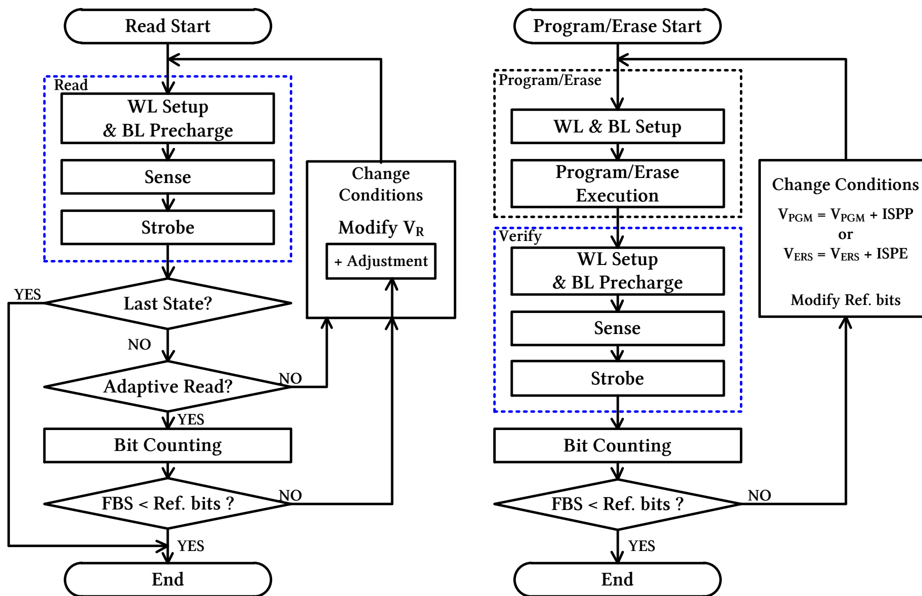
(a) Simulation Results **(b) Analog Counter Out**

Figure 16 : (a) Simulated waveforms of the analog bit-counter model and (b) its output OUT[3:0] varying as the program loop progresses.

3.4. Digital System Model

The digital circuits of the NAND flash memory sensing system include the digital bit-counter, pass/fail (PF) checker, and sequence control logic. These models are described in Verilog, which run efficiently while carrying out long sequences of the incremental/adaptive read, program, and erase operations.

Figure 17 illustrates the sequencing algorithms for the read and program/erase operations for the SLC arrays modeled in this work. For the read operation, sensing is performed in three phases as explained earlier. For the program/erase operation, sensing is performed after each program/erase execution step to verify the results. The bit-counters and PF checker unit output then serves as an indicator whether to continue or terminate the iteration loop.

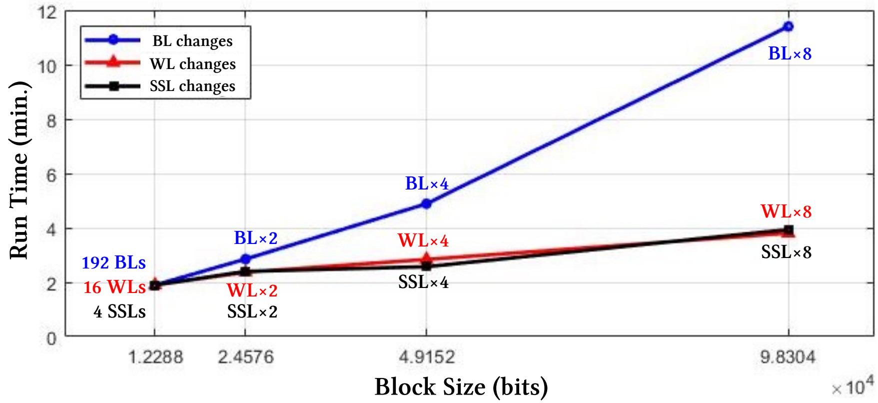


(a) Read Sequence **(b) Program/Erase Sequence**

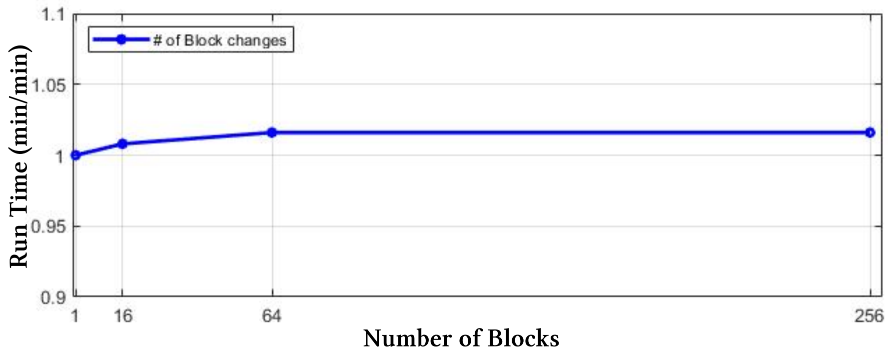
Figure 17 : Flowcharts illustrating the sequences of (a) read and (b) program/erase operations.

Chapter 4. Experimental Results

The SystemVerilog model of the NAND flash memory sensing system includes a block-size array of single-level cells (SLCs) with 16 WLs, 192 BLs, and 4 SSLs. The threshold voltages of the cell array are individually randomized to a Gaussian distribution [20] with a mean of −2.5V and standard deviation of 0.5V. The read and program operations are conducted per page containing 192 bits. The ISPP algorithm uses the initial V_{PGM} of 15.4V, incremental V_{PGM} step of 1V, verify voltage (V_{VFY}) of 3V, and maximum number of loop iterations of 4. The SystemVerilog simulations with the XMODEL primitives are carried out with the Scientific Analog's XMODEL release 2019.09 and Cadence Xcelium version 19.03.009 on a 4-core Intel i5-4460 computer with 16-GB memory. Figure 18(a) shows the run time of the simulations varying with the numbers of BLs, WLs, and SSLs within a block. And figure 18(b) shows how much the simulation run time changes if the number of blocks in a plane increases. Thanks to the models discussed in Section 3.1, the run time increases only with the number of BLs and not with the numbers of WLs or SSLs. And the number of blocks in a plane does not affect the overall simulation time. The proposed model can simulate a 40μs-long single-level cells (SLC) programming with incremental step pulse programming (ISPP) sequence achieving the maximum loop iteration count of 4 on a 12K-bit block in less than 2 minutes.



(a) Simulation run time with a single block



(b) Simulation run time varying the number of blocks

Figure 18 : Run time of the presented SystemVerilog models performing the 40 μs of SLC program operations varying (a) the numbers of BLs, WLs, and SSLs within a sinlge block and (b) the number of blocks in a plane.

4.1. SLC Program with Different ISPP Steps

Figure 19 shows the simulated waveforms of the key signals going through the iterations of ISPP during the program operation. The program voltage (V_{PGM}) with incrementally increasing level is applied to the WL. Depending on the resulting changes in the cell V_{th}, the page buffers' SO outputs change. As the number of the programmed cells increases, the PBS voltages reflect the change and the bit-counters and PF checker determine whether the program is successful or not.

Figure 20 (a) and (b) plot the simulated progressions of the cell V_{th} distribution during 16 SLC program operations. Figure 20(a) is the result with the V_{PGM} step of 1.0 V and maximum loop iteration count of 4. After the fourth iteration, the V_{th} distribution of the cell has a mean (μ) of 3.245V and standard distribution (σ) of 0.2692V. In contrast, Figure 20(b) shows the results obtained with the V_{PGM} step of 0.5V and maximum loop iteration count of 7. When compared to Figure 20(a), this finer V_{PGM} step yields the narrower cell V_{th} distribution of σ = 0.1664V despite

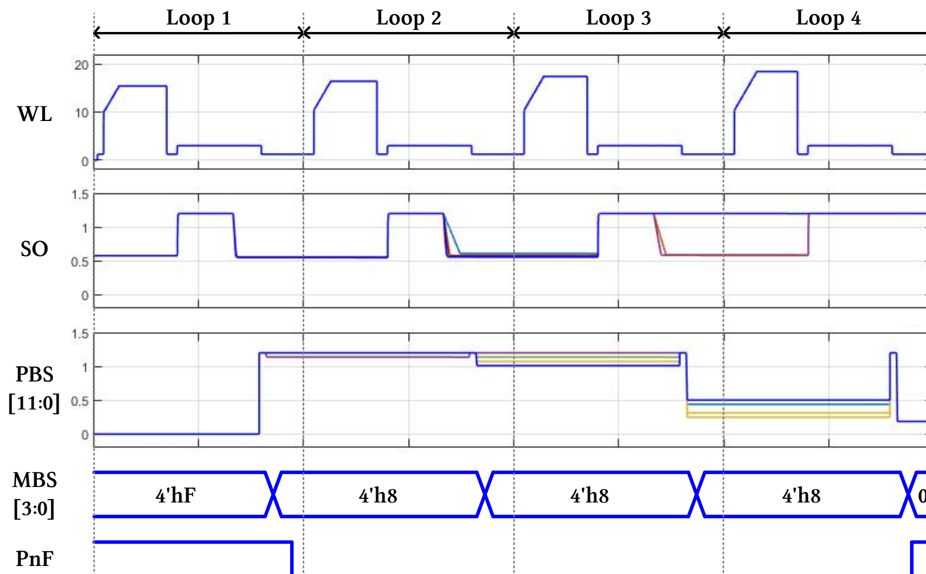
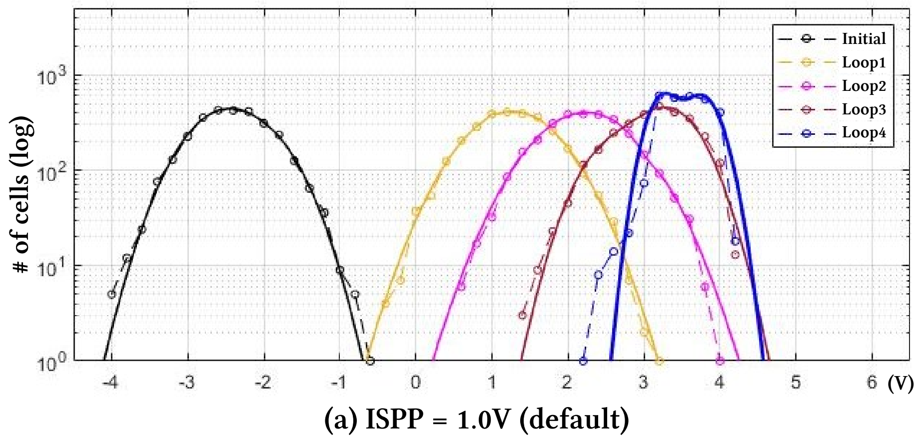
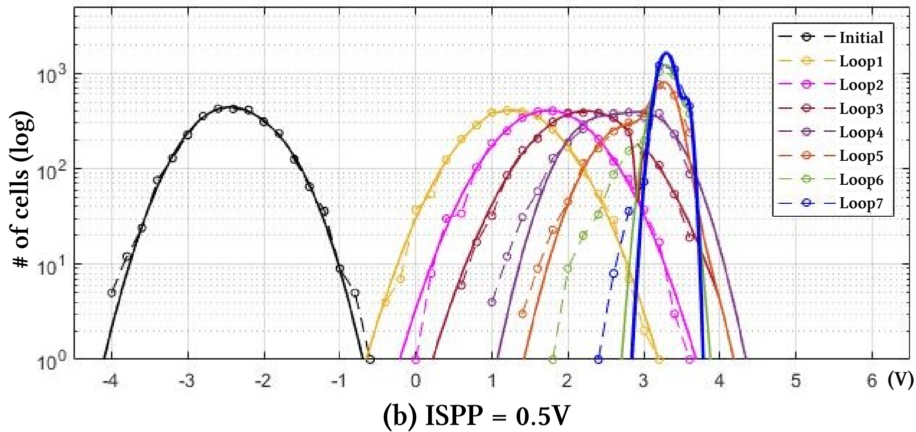


Figure 19 : Simulated waveforms of the key signals during the SLC program operation with 4 ISPP iteration steps.

the slower programming speed (i.e. the more loop iterations required). Figure 21(b) plots the trade-offs between the standard deviation of the final cell V_{th} distribution and the required loop iterations as the V_{PGM} step varies. As expected, the finer V_{PGM} step yields a more tightly-controlled cell V_{th} distribution and sacrifices the program performance. Based on this result, one can choose the optimal V_{PGM} step that satisfies both the performance and reliability requirements.

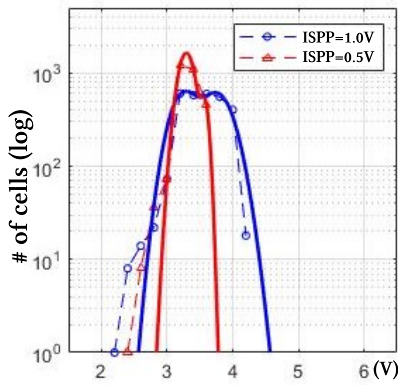


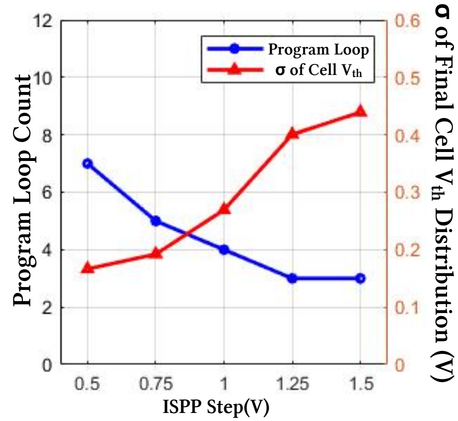
(a) ISPP = 1.0V (default)



(b) ISPP = 0.5V

Figure 20 : Simulated progressions of the V_{th} distributions of the cell with various V_{PGM} steps: (a) V_{PGM} step of 1.0V, (b) V_{PGM} step of 0.5V.





(a) Final Cell V_{th} Distribution (ISPP=1.0V vs ISPP=0.5V)

(b) Performance vs. Cell V_{th} Distribution with ISPP Changes

Figure 21 : (a) Comparison of their final cell V_{th} distributions, and (b) trade−off between the program performance and reliability.

ISPP Steps (V)	0.5	0.75	1	1.25	1.5
Program Loop Count	7	5	4	3	3
σ of Final Cell V_{th} (V)	0.1664	0.1921	0.2692	0.4006	0.4392

Table 1 : Simulation results for the program loop count and cell V_{th} distribution changes with various ISPP steps.

4.2. SLC Program with Different Sensing Times

The presented model can also verify whether the sensing timing margins are adequate for the page buffers to distinguish between the on- and off-cells. If the sensing time is too short (i.e., the period of the XXL pulse staying high in Figure 22), the on-cells may not sufficiently discharge the SO node in time and will be misinterpreted as off-cells. During programming, it implies that the digital controller may incorrectly treat them as programmed cells, and mark them in the program-inhibit state to prevent further programming. Unlike previous studies in [6], [10], the presented model can properly replicate these failures due to the page buffer timings varying with the individual cell conditions.

Figure 23(a) shows the simulated results with various sensing time conditions. When the sensing time is 25% shorter than the optimal value, the resulting V_{th} distribution of cell after 4 loop iterations has the lower mean, implying that the programming is insufficient. The results deteriorate further as the sensing time is reduced by 37.5% and 50%, and the loop count is reduced to 3. Figure 23(b) plots the mean value of the final cell V_{th} distribution

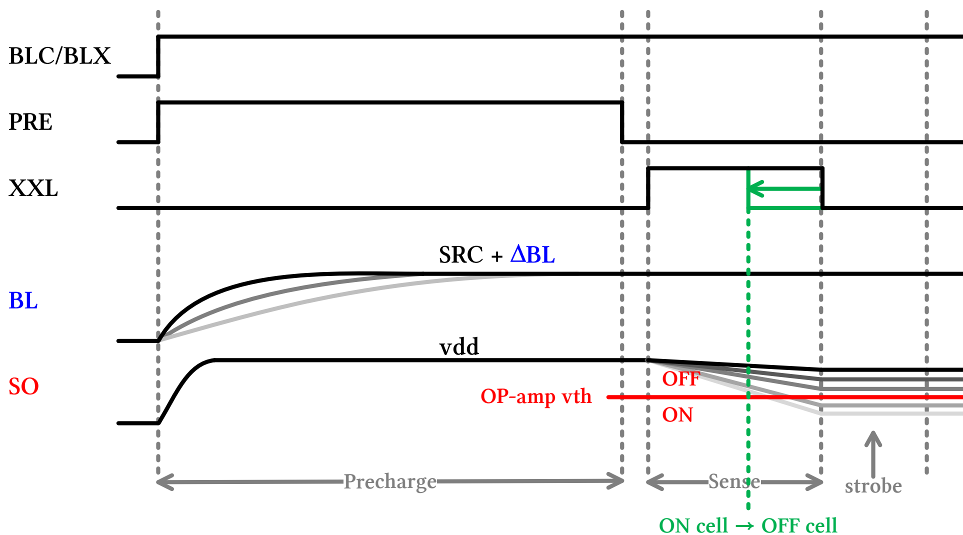
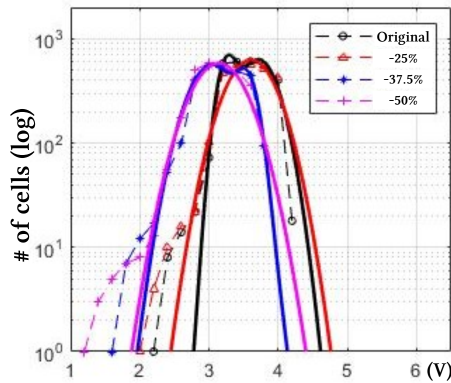
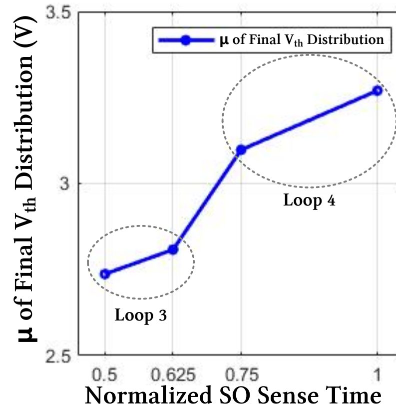


Figure 22 : Timing diagram of the sensing operation with different sensing times (XXL = ‘High’).

varying with the sensing time. To the best of our knowledge, the proposed model is the first one that can predict this correlation between the sensing time and cell V_{th} distribution. This model can help determine the adequate sensing times required for future generations of NAND flash memories by employing new ISPP/ISPE algorithms.





(a) Final Cell V_{th} Distribution with Different SO Sense Time

(b) V_{th} Distribution Shift with Different SO Sense Time

Figure 23 : (a) Simulated cell V_{th} distributions with different sensing times and (b) their mean values (μ) varying with sensing time.

Chapter 5. Conclusions

This thesis presents an efficient SystemVerilog model for NAND flash memory sensing systems, which describes its analog and digital parts in XMODEL and Verilog, respectively. The model enables efficient simulation in SystemVerilog carrying out long sequences of the incremental/adaptive read, program, and erase operations in sophisticated NAND flash memories while tracking the progressions of the individual cell V_{th}s and simulating their impacts on the page buffer timings and sequencing algorithms. We believe that the proposed model can serve as an effective framework to evaluate performance and verify the correctness of new ISPP/ISPE algorithms developed for NAND flash memories, and can be extended to other types of non−volatile memories employing similar incremental/adaptive algorithms.

Bibilography

[1] D. Kim, *et al.*, "A 1Tb 4b/cell NAND flash memory with tPROG=2ms, tR = 110μs, and 1.2 Gb/s high-speed IO rate," in Proc. of *IEEE Int'l Solid-State Circuits Conf.* (*ISSCC*), pp. 218-220, Feb. 2020.

[2] C. Kim, *et al.*, "A 512-Gb 3-b/Cell 64-stacked WL 3-D-NAND flash memory," *IEEE J. Solid-State Circuits*, vol. 53, no. 1, pp. 124-133, Jan. 2018.

[3] Y. Kang, *et al.*, "High-voltage analog system for a mobile NAND flash," *IEEE J. Solid-State Circuits*, vol. 43, no. 2, pp. 507-517, Feb. 2008.

[4] J. Kim, *et al.*, "A 120-mm^2 64-Mb NAND flash memory achieving 180 ns/Byte effective program speed," *IEEE J. Solid-State Circuits*, vol. 32, no. 5, pp. 670-680, May 1997.

[5] K. Park, *et al.*, "Three-dimensional 128Gb MLC vertical NAND flash memory with 24-WL stacked layers and 50 MB/s high-speed programming," *IEEE J. Solid-State Circuits*, vol. 50, no. 1, pp. 204-213, Jan. 2015.

[6] P. Daglio, "A complete and fully qualified design flow for verification of mixed-signal SoC with embedded flash memories," in *Proc. of the Design Automation & Test in Europe Conf.* (*DATE*), pp. 1-6, Mar. 2006.

[7] K. Parat and A. Goda, "Scaling trends in NAND flash," in *Proc. of IEEE Int' l Electron Devices Meeting* (*IEDM*), pp. 2.1.1-2.1.4, Dec. 2018.

[8] C. Chen, *et al.*, "Study of fast initial charge loss and its impact on the programmed states Vt distribution of charge-trapping NAND flash," in *Proc. of IEEE Int' l Electron Devices Meeting* (*IEDM*), pp. 5.6.1-5.6.4, Dec. 2010.

[9] D. Kang, *et al.*, "A 512Gb 3-bit/Cell 3D 6th-Generation V-NAND flash memory with 82MB/s write throughput and 1.2Gb/s interface," in *Proc. of IEEE Int' l Solid-State Circuits Conf.* (*ISSCC*), pp. 216-218, Feb. 2019.

[10] H. Oh, *et al.*, "3-dimensional analysis on the cell string current of NAND flash memory," in *Proc. of Non-Volatile Memory Tech. Symp.* (*NVMTS*), pp. 137-139, Nov. 2005.

[11] S. Ray and J. Bhadra, "Abstracting and verifying flash memories," in *Proc. of Non-Volatile Memory Tech. Symp. (NVMTS)*, pp. 1-5, Nov. 2008.
[12] Scientific Analog, Inc., *XMODEL Reference Manual*, Sep. 2019.
[13] T. Kim, *et al.*, "Non-volatile memory device and memory system including the same," U.S. Patent 20180075918 A1, Mar. 2018.
[14] S. Lee, *et al.*, "A 1Tb 4b/cell 64-stacked-WL 3D NAND flash memory with 12MB/s program throughput," in *Proc. of IEEE Int'l Solid-State Circuits Conf. (ISSCC)*, pp. 340-342, Feb. 2018.
[15] R. Cernea, *et al.*, "A 34MB/s-program-throughput 16Gb MLC NAND with all-bitline architecture in 56nm," in *Proc. of IEEE Int'l Solid-State Circuits Conf. (ISSCC)*, pp. 420-624, Feb. 2008.
[16] K. Suh, *et al.*, "A 3.3V 32Mb NAND flash memory with incremental step pulse programming scheme," *IEEE J. Solid-State Circuits*, vol. 30, no. 11, pp. 1149-1156, Nov. 1995.
[17] B. Lim, *et al.*, "Nonvolatile memory device including page buffer and method for verifying program operation thereof," U.S. Patent 20170278580 A1, Sep. 2017.
[18] S. Joo, *et al.*, "Page buffer, memory device comprising page buffer, and related method of operation," U.S. Patent 9007850 B2, Apr. 2015.
[19] N. Shibata, *et al.*, "13.1 A 1.33Tb 4-bit/Cell 3D-flash memory on a 96-word-line-layer technology," in *Proc. of IEEE Int'l Solid-State Circuits Conf. (ISSCC)*, pp. 210-212, Feb. 2019.
[20] Y. Cai, *et al.*, "Threshold voltage distribution in MLC NAND flash memory: characterization, analysis, and modeling," in *Proc. of Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pp. 1285-1290, Mar. 2013.

Appendix

1. Cell array *plane* model

```
/*------------------------------------------------------------------------
MODULE nand_plane.sv

= Purpose =
An array plane model of a NAND flash memory.
------------------------------------------------------------------------*/

`include "xmodel.h"

module nand_plane #(
    parameter num_block = 1,              // # blocks
    parameter num_bl = 192,               // # strings/block
    parameter num_ssl = 4,                // # strings/wl share
    parameter num_wl = 16,                // # cells/string
    parameter num_pbs = 12,               // # pgbufs/pbs
    parameter num_pf = 16,                // # pgbufs/pf
    parameter real KP_CELL = 1e-4,        // Kp of cell transistors
    parameter real KP_SEL = 1e-3,         // Kp of GSL/SSL select transistors
    parameter real VTH_SEL = 0.5,         // Vth of GSL/SSL select transistors
    parameter real VTH_MIN = -3.0,        // minimum Vth of cell devices
    parameter real VTH_MAX = 5.0,         // maximum Vth of cell devices
    parameter real V_PGM = 15.0,          // minimum program voltage
    parameter real V_PASS = 8.0,          // vpass voltage
    parameter real V_ERASE = -15.0,       // minimum erase voltage
    parameter real ISPP_slope = 1.0,      // program rate (ISPP Slope)
    parameter real OneShot_slope = 0.25,  // Oneshot program efficiency
    parameter real K_ERASE = 1e4,         // erase rate (V/s)
    parameter real vth_std = 0.5,         // standard deviation for the initial vth
distribution
    parameter speed_mean = 0.95,          // mean value for the cell program speed
    parameter speed_std = 0.05,           // standard deviation for the cell program
speed
    parameter real C_GSL = 1e-12,         // load capacitance on each GSL
    parameter real C_SSL = 1e-12,         // load capacitance on each SSL
    parameter real C_WL = 1e-12,          // load capacitance on each WL
    parameter real C_BL = 1e-10,          // load capacitance on each BL
    parameter real C_SO = 5e-12,          // load capacitance on each SO
    parameter string filename = ""        // initial memory state
)(
    input xreal    [num_wl-1:0] WLPre,
    input xreal    [num_ssl-1:0] GSL,
    input xreal    [num_ssl-1:0] SSL,
    input xreal    SRC,
    input xreal    VBB,
    input xbit     BLKWL,
    input xbit     pgm_mode,
    input xbit     ers_mode,
    input xbit     read_mode,
    input xbit     [7:0] BLK_ADDR,
    input xbit     ADDR_CMD,
    input xbit     CTL_1,
```

```
    input xreal     VPP,
    input xbit      XDEC_CTL,
    input xbit      XDEC_Init,
    input xreal     vdd,
    input xreal     BLC,
    input xreal     BLH,
    input xreal     BLS,
    input xreal     BLX,
    input xreal     CLK1,
    input xbit      Monitor_M,
    input xbit      Monitor_S,
    input xreal     [num_pf-1:0] PF,
    input xreal     [num_pbs-1:0] PBS,
    input xreal     PRE,
    input xbit      RST_M,
    input xbit      RST_S,
    input xbit      SET_M,
    input xbit      SET_S,
    input xreal     VH,
    input xreal     Vref,
    input xreal     XXL,
    output xbit     [num_bl-1:0] lat_S,
    output xbit     [num_bl-1:0] lat_nS
);

    xreal       [num_wl-1:0] WL;
    xbit        [num_block-1:0] BLK_SEL;
    reg         [num_block-1:0] BLK_SEL_BIT;

    //-----------------------------------------------------------
    // block decoder
    //-----------------------------------------------------------

    blockdec #(.num_block(num_block)) blockdec0
(.BLK_ADDR(BLK_ADDR[7:0]), .BLK_SEL(BLK_SEL[num_block-1:0]), .ADDR_CMD(ADDR_CMD));
    xbit_to_bit #(.width(num_block)) xmodel_conn0 (.in(BLK_SEL[num_block-
1:0]), .out(BLK_SEL_BIT[num_block-1:0]));

    //-----------------------------------------------------------
    // nand block selection
    //-----------------------------------------------------------

    int i, num_blocksel, blocksel;

    initial begin
        blocksel = 0;
        #100;
        for (i=0; i<num_block; i++) begin
            blocksel = blocksel + BLK_SEL_BIT[i];
            if(BLK_SEL_BIT[i]>0) begin
                num_blocksel = i;
            end
        end
        if (blocksel < 1 || blocksel > 1) $xmodel_error("Block selection fail");
    end
```

```
    xdec XDEC0 (.BLK(BLK_SEL[num_blocksel]), .WLPre(WLPre[num_wl-1:0]), .XDEC_CTL
(XDEC_CTL), .CTL_1(CTL_1), .WL(WL[num_wl-1:0]), .VPP(VPP), .XDEC_Init(XDEC_Init));

    nand_block
#(.filename(filename), .num_bl(num_bl), .num_ssl(num_ssl), .num_wl(num_wl), .num_pbs
(num_pbs), .num_pf(num_pf), .KP_CELL(KP_CELL), .KP_SEL(KP_SEL), .VTH_SEL(VTH_SEL), .
VTH_MIN(VTH_MIN), .VTH_MAX(VTH_MAX), .V_PGM(V_PGM), .V_PASS(V_PASS), .V_ERASE(V_ERAS
E), .ISPP_slope(ISPP_slope), .OneShot_slope(OneShot_slope), .K_ERASE(K_ERASE), .vth_
std(vth_std), .speed_mean(speed_mean), .speed_std(speed_std), .C_GSL(C_GSL), .C_SSL(
C_SSL), .C_WL(C_WL), .C_BL(C_BL), .C_SO(C_SO))  block0 (.ers_mode(ers_mode),
.vdd(vdd), .VBB(VBB), .XXL(XXL), .read_mode(read_mode), .BLX(BLX), .VH(VH), .SET_S(S
ET_S), .Monitor_S(Monitor_S), .PRE(PRE), .BLC(BLC), .RST_M(RST_M), .pgm_mode(pgm_mod
e), .BLKWL(BLKWL), .BLH(BLH), .CLK1(CLK1), .RST_S(RST_S), .SRC(SRC), .BLS(BLS), .SET
_M(SET_M), .Vref(Vref), .Monitor_M(Monitor_M), .WL(WL[15:0]), .GSL(GSL[num_ssl-
1:0]), .SSL(SSL[num_ssl-1:0]), .lat_S(lat_S[num_bl-1:0]), .lat_nS(lat_nS[num_bl-
1:0]), .PBS(PBS[num_pbs-1:0]), .PF(PF[num_pf-1:0]));

endmodule
```

2. Cell array *block* model

```
/*------------------------------------------------------------
MODULE nand_block.sv

= Purpose =
An array block model of a NAND flash memory.
------------------------------------------------------------*/

`include "xmodel.h"

module nand_block #(
    parameter num_bl = 192,              // # strings/block
    parameter num_ssl = 4,               // # strings/wl share
    parameter num_wl = 16,               // # cells/string
    parameter num_pbs = 12,              // # pgbufs/pbs
    parameter num_pf = 16,               // # pgbufs/pf
    parameter real KP_CELL = 1e-4,       // Kp of cell transistors
    parameter real KP_SEL = 1e-3,        // Kp of GSL/SSL select transistors
    parameter real VTH_SEL = 0.5,        // Vth of GSL/SSL select transistors
    parameter real VTH_MIN = -3.0,       // minimum Vth of cell devices
    parameter real VTH_MAX = 5.0,        // maximum Vth of cell devices
    parameter real V_PGM = 15.0,         // minimum program voltage
    parameter real V_PASS = 8.0,         // vpass voltage
    parameter real V_ERASE = -15.0,      // minimum erase voltage
    parameter real ISPP_slope = 1.0,     // program rate (ISPP Slope)
    parameter real OneShot_slope = 0.25, // Oneshot program efficiency
    parameter real K_ERASE = 1e4,        // erase rate (V/s)
    parameter real vth_std = 0.5,        // standard deviation for the initial vth
distribution
    parameter speed_mean = 0.95,         // mean value for the cell program speed
    parameter speed_std = 0.05,          // standard deviation for the cell program
speed
    parameter real C_GSL = 1e-12,        // load capacitance on each GSL
    parameter real C_SSL = 1e-12,        // load capacitance on each SSL
    parameter real C_WL = 1e-12,         // load capacitance on each WL
    parameter real C_BL = 1e-10,         // load capacitance on each BL
    parameter real C_SO = 5e-12,         // load capacitance on each SO
    parameter string filename = ""       // initial memory state
)(
    input xreal     [num_wl-1:0] WL,
    input xreal     [num_ssl-1:0] GSL,
    input xreal     [num_ssl-1:0] SSL,
    input xreal     SRC,
    input xreal     VBB,
    input xbit      BLKWL,
    input xbit      pgm_mode,
    input xbit      ers_mode,
    input xbit      read_mode,
    input xreal     vdd,
    input xreal     BLC,
    input xreal     BLH,
    input xreal     BLS,
    input xreal     BLX,
    input xreal     CLK1,
    input xbit      Monitor_M,
    input xbit      Monitor_S,
```

```
    input xreal     PRE,
    input xbit      RST_M,
    input xbit      RST_S,
    input xbit      SET_M,
    input xbit      SET_S,
    input xreal     VH,
    input xreal     Vref,
    input xreal     XXL,
    input xreal     [num_pf-1:0] PF,
    input xreal     [num_pbs-1:0] PBS,
    output xbit     [num_bl-1:0] lat_S,
    output xbit     [num_bl-1:0] lat_nS,
);

    // local parameters
    localparam size_block = num_ssl*num_bl*num_wl; // size of a block
    xreal [0:num_bl-1] BL;

    // gaussian random variables
    real mean, stdddev, gaussian;
    import "DPI-C" context function real xmodel_rand_gaussian(real mean, real
stddev);

    //------------------------------------------------------------
    // initialization of memory array
    //------------------------------------------------------------

    reg [0:0] XADDR;
    longint MEM_STATE, MEM_STATE_SPEED;
    reg [63:0] data_bin [0:size_block-1];
    integer file;
    integer output_file,output_file_speed;
    real init_vth, cell_speed;
    int i, j, k;

    initial begin
    $display("block size = ", size_block);
    $display("mem block = ", idx_block*size_block);
        $display("Initializing memory state...");
        if (filename != "") begin
            file = $fopen(filename, "rb");
            if (file == 0) $xmodel_error("cannot read memory state file: ",
filename);
            MEM_STATE = NPRIMS_initarray_real(size_block);
            MEM_STATE_SPEED = NPRIMS_initarray_real(size_block);
            k = $fread(data_bin, file);
            if (k < size_block) $xmodel_error("insufficient data stored in file: ",
filename);
            for (j=0; j<size_block; j++) begin
                NPRIMS_putelem_real(MEM_STATE, i*size_block+j, $bitstoreal
(data_bin[j]));
            end
        end
        else begin
            output_file = $fopen("initial_vth_out.txt","w");
            MEM_STATE = NPRIMS_initarray_real(size_block);
```

```
            for (i=0; i<size_block; i++) begin
                init_vth = xmodel_rand_gaussian(VTH_MIN,vth_std);
                $fwrite(output_file,"%f\n",init_vth);
                NPRIMS_putelem_real(MEM_STATE, i, init_vth);
            End
            $fclose(output_file);

            output_file_speed = $fopen("cell_speed.txt","w");
            MEM_STATE_SPEED = NPRIMS_initarray_real(size_block);
            for (i=0; i<size_block; i++) begin
                cell_speed= xmodel_rand_gaussian(speed_mean,speed_std);
                $fwrite(output_file_speed,"%f\n",cell_speed);
                NPRIMS_putelem_real(MEM_STATE_SPEED, i, cell_speed);
            end
            $fclose(output_file_speed);
        end
        $display("Done.");
    end

    // wordline loads
    capacitor #(.C(C_WL)) cload_WL [num_wl-1:0] (.pos(WL[num_wl-1:0]), .neg(VBB));
    capacitor #(.C(C_GSL)) cload_GSL [num_ssl-1:0] (.pos(GSL[num_ssl-1:0]),
 .neg(`ground));
    capacitor #(.C(C_SSL)) cload_SSL [num_ssl-1:0] (.pos(SSL[num_ssl-1:0]),
 .neg(`ground));

    //------------------------------------------------------------
    // array of cell strings
    //------------------------------------------------------------

    int idx_block;
    longint mem_block;
    longint mem_block_speed;
    longint mem_string;

    assign idx_block = int'(12'b0000_0000_0000);
    assign mem_block = NPRIMS_getarray_real(MEM_STATE, idx_block*size_block);
    assign mem_block_speed = NPRIMS_getarray_real(MEM_STATE_SPEED,
idx_block*size_block);

    // selected memory block
    genvar gen_i;
    generate
        for (gen_i=0; gen_i<num_ssl*num_bl; gen_i++) begin:g1
            nand_str #(.num_wl(num_wl), .KP_CELL(KP_CELL), .KP_SEL(KP_SEL),
 .VTH_SEL(VTH_SEL), .V_PGM(V_PGM), .V_PASS(V_PASS), .ISPP_slope(ISPP_slope),
 .OneShot_slope(OneShot_slope), .str_idx(gen_i*num_wl))
str .BL(BL[$rtoi(gen_i/num_ssl)]), .WL(WL[num_wl-1:0]), .GSL(GSL[(gen_i%num_ssl)]),
.SSL(SSL[(gen_i%num_ssl)]), .SRC(SRC), .mem_block(mem_block), .mem_block_speed(mem_b
lock_speed), .pgm_mode(pgm_mode), .ers_mode(ers_mode), .read_mode(read_mode));
        end
    endgenerate
```

```
    //-----------------------------------------------------------
    // page buffers
    //-----------------------------------------------------------
    generate
        for (gen_i=0; gen_i<num_bl; gen_i++) begin:g3
            nand_pgbuf #(.C_BL(C_BL), .C_SO(C_SO)) pgbuf0 .BL(BL[gen_i]),
.lat_S(lat_S[gen_i]), .lat_nS(lat_nS[gen_i]), .PBS(PBS[$rtoi(gen_i/num_pf)]), .PF(PF
[gen_i%num_pf]), .BLX(BLX), .VH(VH), .XXL(XXL), .Monitor_S(Monitor_S), .SET_S(SET_S)
, .BLC(BLC), .PRE(PRE), .RST_M(RST_M), .BLH(BLH), .CLK1(CLK1), .RST_S(RST_S), .BLS(B
LS), .Monitor_M(Monitor_M), .SET_M(SET_M), .Vref(Vref), .vss(`ground), .vdd(vdd));
        end
    endgenerate

endmodule
```

초록

NAND 플래시 메모리의 센싱 시스템은 대용량의 데이터를 저장할 수 있는 셀 어레이와 이를 구동시키기 위한 워드 라인 디코더, 페이지 버퍼, 아날로그 / 디지털 비트 카운터 및 디지털 시퀀스 컨트롤러로 구성된 복잡한 혼성신호 회로이다. 본 연구에서는 개별 셀의 초기 조건과 특성에 따라 서로 다른 양상을 보이는 문턱 전압 (V_{th})의 변화를 반영할 수 있으며, 페이지 버퍼의 특성을 포함한 상세한 아날로그 동작들의 모델링하여 디지털 컨트롤러가 읽기, 프로그램 및 삭제 작업에 사용하는 다양한 알고리즘의 효율성을 평가할 수있는 모델 및 시뮬레이션 프레임워크를 제안한다. 제안하는 모델은 디지털과 아날로그로 나뉘어진 검증 환경이 아닌 하나의 통합된 SystemVerilog기반으로 작성되었으며, 특히 XMODEL 프리미티브를 사용하여 아날로그 회로의 이벤트 기반 시뮬레이션을 통해 효율적인 검증이 가능하게 되었다. 해당 시스템 모델을 기반으로 12K 비트의 단일 레벨 셀 (SLC) 블록에서 최대 루프 반복 횟수가 4 회인 40μs 길이의 ISPP (Incremental Step Pulse Programming) 동작을 2 분 이내에 시뮬레이션 할 수 있었다. 또한, 검증과정을 통해 얻게 되는 개별 셀 V_{th} 분포 분석을 통해서 프로그래밍 속도와 신뢰성 사이의 관계를 펄스 스텝 크기의 함수로서 표현할 수 있었으며, 페이지 버퍼의 센싱 시간 조절을 통한 최종 셀 V_{th} 분포의 중심치에 대한 영향에 대해서도 검증가능하다.

주요어 : 낸드플래시 메모리, 혼성신호회로, 셀 문턱전압 분포, XMODEL, SystemVerilog.

학번 : 2018-29065