Tampere University

Maxim Shilov

# IMPLEMENTATION OF CONFIGURABLE AND DELAY-OPTIMAL STATIC RANDOM-ACCESS MEMORY GENERATOR

Bachelor's thesis
Faculty of Information Technology and Communication Sciences
Examiner: University lecturer Erja Sipilä
November 2020

# ABSTRACT

Maxim Shilov: Implementation of configurable and delay-optimal static random-access memory generator
Bachelor's thesis
Tampere University
Degree Programme in Computing and Electrical Engineering, BSc (Tech), Electrical Engineering
November 2020

---

Computer static random-access memory (SRAM) can greatly vary in size. Peripheral circuitry which provides memory readability and writability must be changed accordingly with a size order to keep memory operation fast enough. Development of a memory layout generator with these architectural changes may require huge effort. However, within a given architecture, delay optimization can be made by making logic gates scalable and furthermore they must be scaled appropriately. One way to estimate these scale factors is to use the Method of Logical Effort.

In this work a configurable generator for a small and delay-optimized static RAM-memory for the 180 nm process was implemented. The main emphasis in this work was made toward the least delay for different memory sizes within the same architecture. To achieve this goal, some of the logic gates were implemented scalable and the Method of Logical Effort was used to scale these gates appropriately. To validate the obtained results, delay-optimal logic gate sizes for a different number of bits were found empirically by straightforward repetitive simulations. To see how significant relative error was, absolute delays caused by differently sized gates were simulated. Also, these delays were compared to ones caused by temperature rise and voltage drop.

It was shown that the Method of Logical Effort can be used in a development of semi-custom SRAM generator, although with some constraints. It was also shown that error caused by this method is not significant and that environmental changes can have a larger impact on a delay. This work can serve as a guide during development of a similar macro and its layout generator if the fastest topology by scaling logic gates must be achieved.

Keywords: SRAM, static random-access memory, generator, layout, the Method of Logical Effort

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

# PREFACE

I would like to thank Teppo Karema and Tuukka Vaaraniemi from VLSI Solution Oy for a great opportunity to gain experience and knowledge while doing this project and for all provided help and support.

Tampere, 24 November 2020

Maxim Shilov

# CONTENTS

1. INTRODUCTION ........................................................................................ 1
2. THEORETICAL BACKGROUND........................................................................ 3
2.1 Static random-access memory ........................................................................ 3
2.2 L language and ”Led” layout editor ................................................................. 5
2.3 The Method of Logical Effort ........................................................................... 7
3. METHODOLOGY........................................................................................... 12
3.1 SRAM implementation .................................................................................. 12
3.1.1 SRAM cell ...................................................................................................... 13
3.1.2 Row circuitry ................................................................................................. 14
3.1.3 Column circuitry ............................................................................................ 15
3.2 SRAM optimization....................................................................................... 16
3.2.1 Building empirical model for optimal inverter sizing ..................... 16
3.2.2 Adapting the Method of Logical Effort for optimal inverter sizing . 17
4. RESULTS AND DISCUSSION.......................................................................... 19
5. CONCLUSION ................................................................................................. 26
REFERENCES...................................................................................................... 28

# LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| DSP | Digital signal processor |
| IC | Integrated circuit |
| MOS | Metal-oxide-semiconductor |
| *rwl* | Read word line |
| SRAM | Static random-access memory |
| SoC | System on a chip |
| *wl* | Word line |
| *wwl* | Write word line |

# 1. INTRODUCTION

Modern SoC's (system on a chip) are dependent on integrated SRAM (static random-access memory) and overall memory area can overlay large part of a chip [1]. During development of such a system, particularly DSP (digital signal processor), there may rise a need for a small-size, high-speed SRAM for buffering. In a semi-custom layout design technique this need is usually satisfied with a help of memory generator which allows automatically instantiate repetitive structures like standard cells, contacts, wires or transistors.

Peripheral circuitry which provides memory readability and writability must be changed accordingly with a size order to keep memory operation fast enough. Development of a memory layout generator with these architectural changes may require a huge effort. Also, size-configurable generator with same architecture will be optimal only for a particular size. However, within a given architecture, delay optimization can be made by making logic gates scalable. Depending on the memory size, these logic gates must be scaled appropriately. One way to estimate these scale factors is to use the Method of Logical Effort [2].

In this work a size-configurable generator for small and delay-optimized SRAM memory was developed. In this work, unlike in similar one [3], peripheral circuitry's logic gates were implemented as scalable components where possible. In addition, to scale them appropriately depending on the memory size, the Method of Logical Effort was used. In order to validate the results gotten from the Logical Effort Method, additional model based on experimental simulations was developed and results were compared. The difference in results, in particular its magnitude and significance, was studied. The delay increase caused by this error was compared to a delay increase caused by common environmental changes, such as voltage drop and temperature rise.

Chapter 2 of the present work gives brief overview on SRAM operation principle and how size-configurable SRAM generator with scalable logic gates can be implemented by means of L language and "Led" tool. It also presents needed theoretical background on the Logical Effort Method. Chapter 3 focuses on how SRAM was implemented in this work. Additionally, it shows what quantities must be measured in order to adapt the

Logical Effort Method, and how to build an empirical model to validate results. In Chapter 4 all measurements and calculations are presented and briefly discussed. Finally, Chapter 5 aims to summarize and analyze all results obtained in this work.

# 2. THEORETICAL BACKGROUND

This chapter provides basic information on SRAM memory operation and its crucial parts. After that, it gives a brief overview on how scalable logic gates can be implemented by means of L language and "Led" layout editor. Finally, in order to understand how gates can be scaled to achieve the least possible delay, a required part of the theory on the Method of Logical Effort is presented.

## 2.1 Static random-access memory

Static random-access memory is a type of volatile computer memory that is primarily used in caches and buffers due to its speed [4]. The main building block of any SRAM is the memory cell itself, where "0" or "1" is stored. Different topologies of a memory cell have been developed; each with a different number of transistors and each with its own advantages and disadvantages [5][6][7]. However, most of them are based on the most common topology which consists of six transistors (6T). Its schematic is depicted in Figure 1. The cell consists of two cross-connected inverters and two access transistors. To write a value into the cell ("1" for example) *bit* line is pulled-high and its complement value ("0" in this case) is set in *bit_b*. Once the wordline (*wl*) is raised, it opens access transistors and shortens *bit* and *bit_b* to nets Q and *Q_b* respectfully. Both inverters start to feed each other complement values, positive feedback loop occurs, and inverters keep storing the bit within the cell as long as voltage is supplied.



***Figure 1.*** *Schematic of 6T SRAM cell.*

The written value is stored in net *Q*. To be read, *wl* must be low and both *bit* and *bit_b* must be precharged high. Once *wl* is raised again, *bit* or *bit_b* (depending on stored value) will be tied to ground through nMOS transistor in the corresponding inverter.

When several (or dozens) of cells are aligned horizontally and all share common *wl* then *word* is formed. Words, in turn, inserted one below the another (with common *bit* and *bit_b* lines) form a memory array. However, to be able to store information within a memory, additional circuitry is required. Example of such circuitry is a decoder which is used to choose from $n$ bit sequence one of the $2^n$ words. [8, pp. 498-501][9, pp. 418-423] Block diagram of exemplary simple SRAM can be seen below (Figure 2).

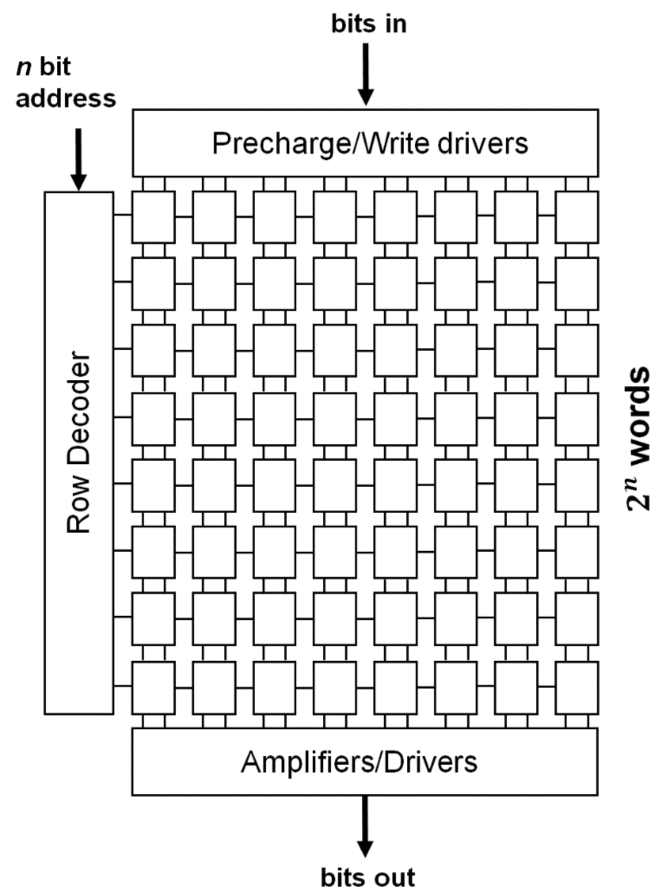


***Figure 2.*** *Simple generic SRAM architecture.*

The choose of one or another architectural solution may depend on many factors such as: memory cell topology, memory size (meaning number of words and bits per word), speed, area and power consumption constrains, requirement of additional features (like sleep mode) and so on [10]. Memory implemented in this work will be presented and discussed in detail in Chapter 3.1.

## 2.2 L language and "Led" layout editor

Mentor Graphics' "Led" layout editor is a tool for implementing IC (integrated circuit) layout within a graphical user interface. Implemented circuits are stored in so-called L-files and described in L language that has a C-like syntax. Each circuit is called a cell. Simple circuits such as logic gates can be saved in L-files and instantiated in a more complex circuits on a higher level of abstraction. A cell instantiation in another cell is not limited to any number and complex designs may consist of many layers of cells. This is how complexity is handled in L. Simple inverter that is described in L language (Program 1) and its corresponding layout representation in "Led" editor (Figure 3) can be found below.

```
 1  L::TECH tech_file
    # Declaring cell component named "inverter"
 3  CELL inverter (INT tn_width = 0.42, INT tp_width = 0.84){

    # Creating transistors
 6  TN tn_1 W=tn_width L=0.18 R90 AT (0, 0);
 7  TP tp_1 W=tp_width L=0.18 R90 AT (0, 2);

    # Making their contacts
10  MNDIFF mndiff_d W=tn_width L=0.18 R90 AT tn_1.d;
11  MNDIFF mndiff_s W=tn_width L=0.18 R90 AT tn_1.s;
12  MPDIFF mpdiff_d W=tp_width L=0.18 R90 AT tp_1.d;
13  MPDIFF mpdiff_s W=tp_width L=0.18 R90 AT tp_1.s;

    # Wiring transistors together
16  WIRE POLY tn_1.gr HOR VER tp_1.gl;
17  WIRE MET1 mndiff_d HOR VER mpdiff_d;

    # Declaring terminals
20  IN POLY input AT tn_1.gr;
21  OUT MET1 output AT mndiff_d;
22  GND MET1 gnd AT mndiff_s;
23  VDD MET1 vdd AT mpdiff_s;
    }
```

***Program 1.*** *Inverter description in L language.*

Different instances, such as transistors (6, 7), wires (16, 17), contacts (10-13), polygons and terminals (20-23) can be crated in L language. Every instance except wires and polygons must have a name (*tn_1* for example). In wires, terminals and polygons material level must be specified (*MET1, MET2, POLY, NDIFF*). Also, different features like width and length, orientation (*R90, R180*, *R270, RX, RY*) and location (*AT*) must be given within a single instance declaration. Contacts and transistors can be wired together and wire direction can be specified with special words like *HOR, VER.* Wire's path can also be laid more precisely using other special keywords: *UP=<distance>, DOWN=<distance>, LEFT=<distance>, RIGHT=<distance>*. Terminals serve as a communication path between cells in complex designs as well as power and I/O ports to outside world. All allowed material levels, design rules and other constraints are specified in a technology file (1). It is declared in a first row with a special character *L::* indicating that this is an L-file.

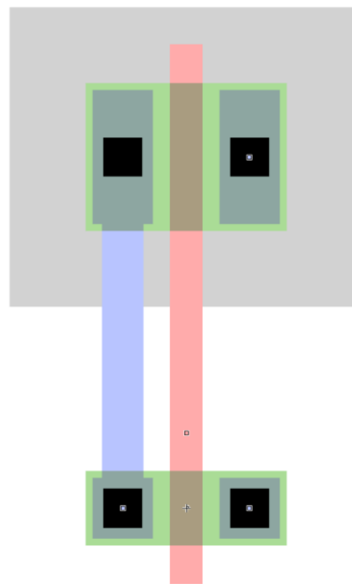

***Figure 3.*** *Layout representation of inverter from Program 1.*

Cell is called a generator if parameters are given inside parentheses within a cell declaration (3). Such generator can be called from "Led" and parameters are given by the user or it can be called from the code in an L-file. Repetitive structures and cell instantiations can be done by means of while loop and condition checking by if-else statements.

All described features are only a small part of all possibilities that "Led" and L language can utilize. However, given information should be enough to have an idea on how a layout of complex circuits such as SRAM can be implemented.

## 2.3 The Method of Logical Effort

The Method of Logical Effort was developed by Sutherland and Sproull in 1991 and is used to estimate a delay in digital circuits [2]. It also can be used to find a suitable number of logic gates in a stage and how to scale them appropriately to achieve minimal delay. Generator developed in this work is not able to add or remove logic gates, only change their size. Therefore, only relative information and basic concepts from applicative point of view will be provided in this chapter. All following information in this chapter is primarily based on Sutherland's and Sproull's book [2] as well as [8, pp. 155-171].

Every logic gate consists of transistors. Typically, signal transition from "0" to "1" is accomplished by a pull-up network that consists of pMOS transistors while signal transition from "1" to "0" is done by a pull-down network which consist of nMOS transistors. Simplest logic gate is an inverter which has one nMOS and one pMOS transistor in series (Figure 3). Hole mobility is lower than electron mobility in a diffusion and thus to achieve equal fall and rise times, pMOS transistors are usually made wider than nMOS transistors. How much wider depends on a particular technology but typical values for pMOS to nMOS width ratio lie between 1.5 and 3. If logic gate is driving some capacitive load, be it wire or another gate, it has to be scaled appropriately; otherwise it will require long time to charge or discharge the load. Scaling means changing a width of transistors. If for example inverter *x1* has nMOS transistor of width 400 nm and pMOS transistor of width 800 nm then threefold inverter *x3* will have nMOS of width 1200 nm and pMOS of 2400 nm. Scaling in the case of NAND gates is a bit trickier. *N*-input NAND gate consists of *N* nMOS transistors in series and *N* pMOS transistors in parallel. In order to deliver same current as inverter despite number of inputs, transistors in series have to be *N*-times wider than in a unit inverter.

In the Logical Effort Method delays are expressed in technology-independent, normalised with respect to inverter values (1):

$$d = \frac{t_{pd}}{\tau}, \quad (1)$$

where $t_{pd}$ is the absolute propagation delay of a logic gate and $\tau$ is the absolute propagation delay of an inverter respectively. Delay of a logic gate in a stage depends on several factors. First, on its size and the load that it drives.

This value is called the *electrical effort* or the *fanout.* It is denoted by letter $h$ and can be expressed as:

$$h = \frac{C_{out}}{C_{in}}, \tag{2}$$

where $C_{out}$ is a capacitance of the load that gate drives and $C_{in}$ is its input capacitance. As was stated earlier, $h$ is dependent on the size of a gate and Equation (2) operates with capacitances. The fact is that logic gate input capacitance is proportional to its size and by knowing the size, capacitance can be estimated. If for example inverter *x1* is driving four times larger inverter *x4* then fanout of *x1* is equal to $\frac{4C_{in}}{C_{in}} = 4$.

The gate delay is also dependent on how complex its inner structure is. This value is called the *logical effort.* It shows how much worse the logic gate is in a delivering output current compared to an unit inverter's output current if they both have the same input capacitance. From another point of view, it shows how larger gate's input capacitance (compared to a unit inverter) should be, in order to deliver the same amount of current. Mathematically it is expressed like:

$$g = \frac{R_g C_g}{R_{inv} C_{inv}}, \tag{3}$$

where $R_g$ and $C_g$ are logic gate's input resistance and capacitance. Inverter's input resistance and capacitance are denoted by $R_{inv}$ and $C_{inv}$. One way to estimate the logical effort is to simulate delays of logic gate that drives different loads and fit the line to obtained results. The slope of the obtained line then must be divided by a similarly found slope of an inverter. Point where slope crosses the delay axis (equivalently delay value where load is equal to zero) is called the *parasitic delay* and is denoted by $p$. By putting it all together, formula for calculating gate delay can be expressed as:

$$d = gh + p \tag{4}$$

Term $gh$ is also called the *stage effort* and is denoted by $f$.

When several gates in series and/or parallel are forming a network, their *path effort* is expressed as:

$$F = GBH, \tag{5}$$

where $B$ is the path's *branching effort* and $G$ and $H$ are the path logical and electrical efforts respectively.

Each of these terms can be calculated as a product of every individual term in a stage (6).

$$F = \prod g_i b_i h_i \tag{6}$$

If some stage in the network consists of several gates in parallel, then its stage branching effort $b_i$ can be found using Formula (6):

$$b_i = \frac{C_{onpath} + C_{outpath}}{C_{onpath}}, \tag{7}$$

where $C_{onpath}$ is the input capacitance of a logic gate where delay to be calculated, while $C_{outpath}$ is the total input capacitance of parallel logic gates in a stage. Delay along the whole path is equal to (8):

$$D = \sum d_i = \sum f_i + \sum p_i \tag{8}$$

It turns out, that the path electrical effort $H$ is equal to the ratio of network's output to input capacitance, because the input capacitance of a stage $i$ is the output capacitance of stage $i-1$. Thus, all terms except $C_{in,1}$ and $C_{out,n}$ in a chain of $n$ gates are cancelled out.

The following example clarifies all abovementioned equations. The inverter from Figure 4 in the Stage 2 has input capacitance of 3 of some unit capacitance *C*. The NAND gate is designed so that it delivers the same amount of current as the inverter and their effective resistances are equal. In such a case its input capacitance will be equal to *4C.* The whole network is driving capacitive load of value *100C*. There are two NANDs in parallel in the input node and thus input capacitance $C_{in}$ is equal to *2*4C=8C.* Therefore, the path electrical effort is equal to $H = \frac{100C}{8C} = \frac{25}{2}$. The branching effort of the Stage 1 is equal to $b_1 = \frac{4C+4C}{4C} = 2$ and branching effort of the Stage 2 to $b_2 = \frac{(3C+3C)+3C}{3C} = 3$. Overall path branching effort is therefore $B = 3 \cdot 2 = 6$. The logical effort of the Stage 1 is $g_1 = \frac{R_{NAND} C_{NAND}}{R_{inv} C_{inv}} = \frac{C_{NAND}}{C_{inv}} = \frac{4C}{3C} = \frac{4}{3}$. The logical effort of the inverter is equal to 1 by definition. By combining all obtained values, the path effort of the network can be count as $F = \left(\frac{4}{3} \cdot 1\right)(2 \cdot 3)\left(\frac{25}{2}\right) = 100$.

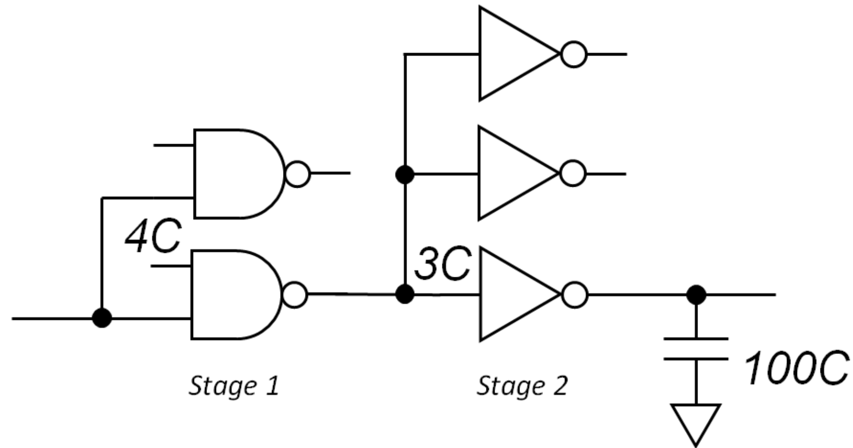


***Figure 4.*** *Example network.*

Suppose that load and input capacitance (NAND gates) are fixed and only size of the inverters in the Stage 2 can be changed. This means that the path electrical effort $H$ is constant. Delay along the path, according to equation (8) is equal to

$$D = (g_1 h_1 + p_1) + (g_2 h_2 + p_2) \tag{9}$$

The electrical effort $h_2$ in the Stage 2 can be expressed through the path electrical effort $H$ as:

$$D = (g_1 h_1 + p_1) + \left(g_2 \frac{H}{h_1} + p_2\right) \tag{10}$$

To minimize the delay along the path by adjusting size of the inverter in the Stage 2, partial derivative with respect to $h_1$ to be taken and equated to zero:

$$\frac{\partial D}{\partial h_1} = h_1 + g_2 \frac{H}{h_1^2} = 0 \tag{11}$$

By substituting $H$ with $h_1 h_2$ and solving equation (11) following result is obtained:

$$g_1 h_1 = g_2 h_2 \tag{12}$$

Equation (12) tells that delay is minimized when every stage has the same effort $f$. Thus, to equalize efforts in every stage, the path effort $F$ from equations (5, 6) must be raised to the power of $\frac{1}{N}$:

$$\hat{f} = F^{1/N}, \tag{13}$$

where $N$ is a number of stages.

By inserting the stage effort formula $f = gh$ into equation (2) and by replacing $f$ with the equalized effort $\hat{f}$, the capacitance transformation formula can be obtained:

$$C_{in,i} = \frac{C_{out,i} \cdot g_i}{\hat{f}} \tag{14}$$

This formula can be used to adjust logic gate input capacitance (or equivalently its size) if the number of stages, network's input and output capacitances and logical efforts are known. To adjust the inverter size from Figure 4, the equalized stage effort must be found by rising $F$ to the power of $\frac{1}{2}$ (because there are 2 stages). Obtained value $\hat{f} = \sqrt{F} = \sqrt{100} = 10$ can then be used in the capacitance transformation formula (14) to find the input capacitance of the inverter:

$$C_{in,2} = \frac{100C \cdot 1}{10} = 10C \tag{15}$$

Result from equation (15) tells, that in order to achieve the least delay in the network from Figure 4, the input capacitance of the inverter in the Stage 2 has to be 10*C* instead of 3*C*. This means that inverter must be more than three times larger than in Figure 4 to achieve the least delay.

# 3. METHODOLOGY

Before actual chip fabrication, simulation remains the main tool of circuit validation and verification. All simulations were performed in ELDO simulator. Supply voltage was equal to 1.8 V and temperature parameter 25 °C unless otherwise mentioned. Some fixed-sized logic gates were made in a full-custom manner in “Led”, while scalable gates and repetitive parts and structures were implemented as generators in L language. Implemented design was intended for the 180 nm technology.

## 3.1 SRAM implementation

As was mentioned earlier, choice of the architecture depends on many factors. One of studied SRAM cells in [6] was chosen to be used in this work due to its low power consumption and ability to read and write values simultaneously. The selected cell allows to design relatively simple peripheral circuitry (Figure 5) with no need in sense amplifiers and precharging bit lines. Corresponding layout of generated SRAM can be seen in Figure 6.

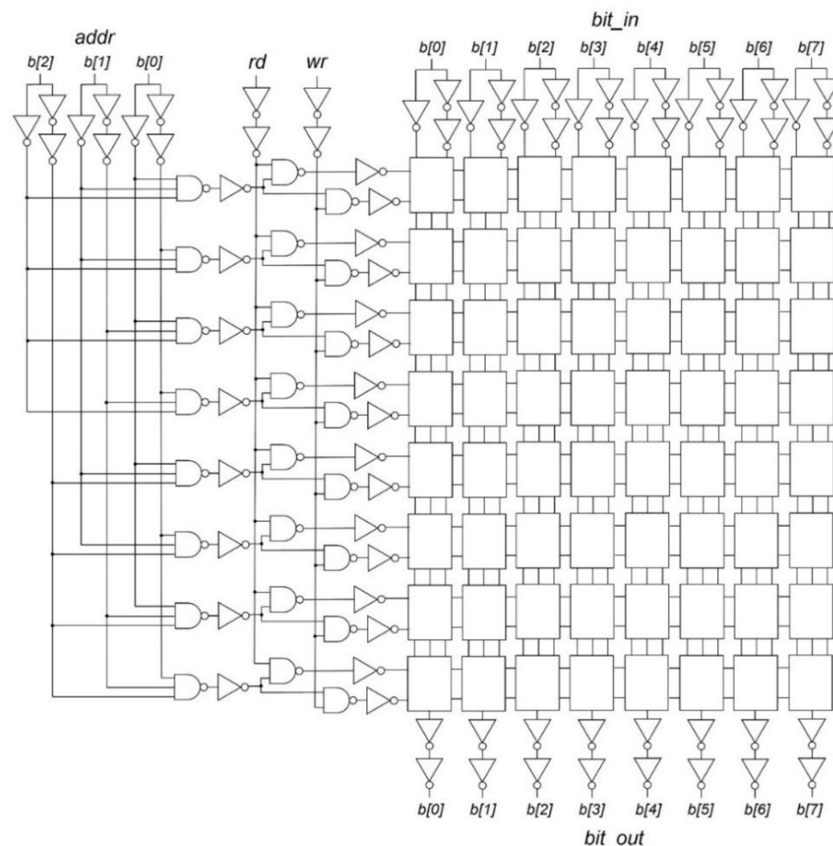


***Figure 5.*** *SRAM architecture used in this work (64-bit example).*

One of the words (8 in this case) is chosen by applying some bit sequence to *addr* and word to be written to *bit_in.* In order to write bits to chosen word, *rd* is pulled high. They can be read in the same time by pulling *wr* high as well and written data will appear in *bit_out*.

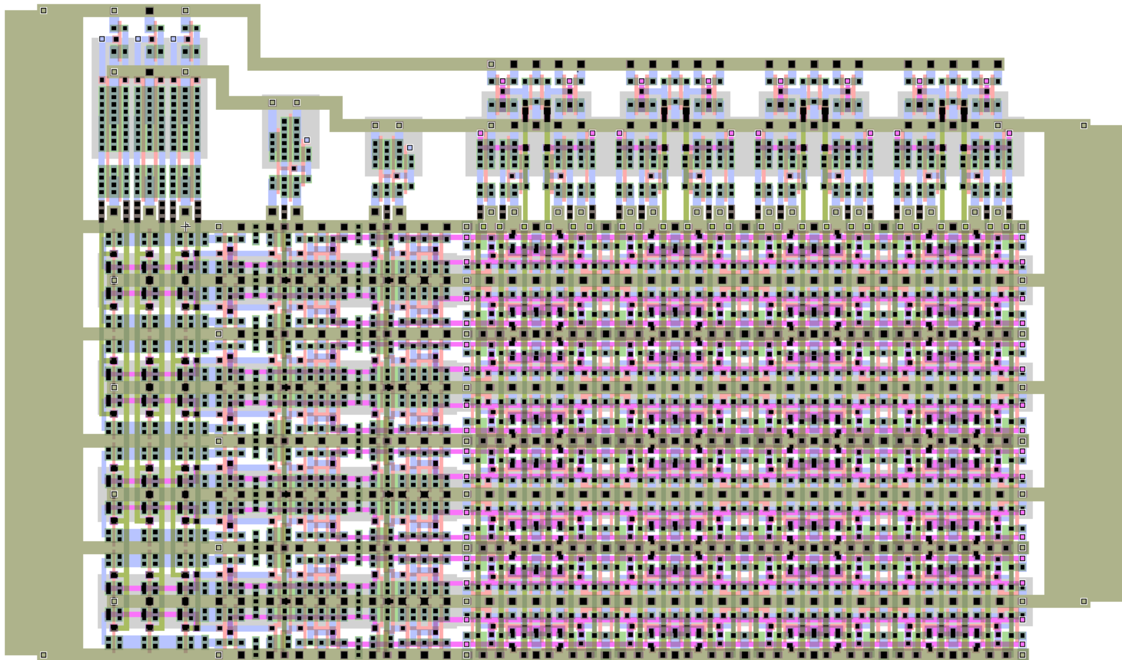

***Figure 6.*** *Layout of the generated SRAM (64 bits).*

When a user decides to call a generator from “Led”, the number of words and the number of bits per word must be specified. Generator automatically chooses optimal logic gate sizes based on a model described in Chapter 3.2.1. If time constraints are not tight, area and power consumption can be reduced by relaxing size parameters manually in the source code.

### 3.1.1 SRAM cell

In this work slightly modified SRAM cell from Figure 1 was used. In [6] different cell topologies were studied and compared. Most efficient and proposed one (10T) was used in this work. In this cell additional inverter is added to node *Q_b* (Figure 1). Complement of the stored value goes through inverter and true value appears after it. In order to control that appearance, transition gate is added. Transition gate consists of one nMOS and one pMOS transistor in parallel and controlled by two complement values. It allows to pass both strong “0”s and “1”s through a line (meaning without a voltage drop caused by threshold voltage of either transistor) [11, p. 866].

Memory cell is a main area consumer on a chip die because it is repeated hundreds, thousands and millions of times. Even small saving in a cell layout can have a huge impact on the overall area. For this reason, transition gate in proposed 10T cell was replaced with a typical nMOS access transistor. The modified cell schematic and implemented layout can be found below (Figure 7).

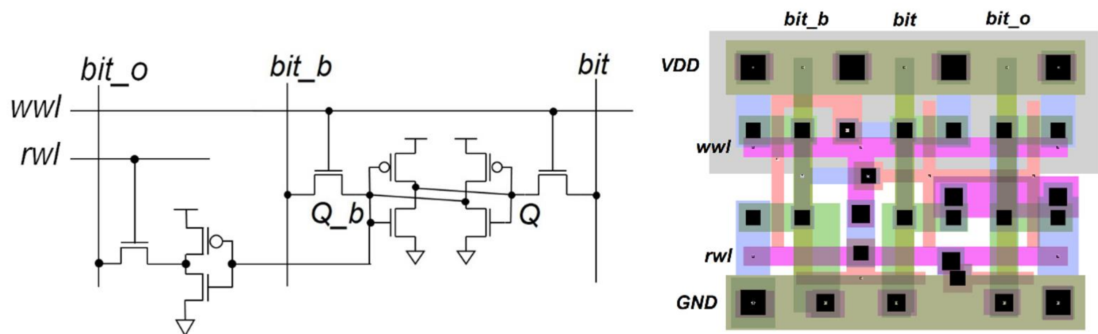


***Figure 7.*** *Modified SRAM-cell and corresponding layout.*

Access nMOS transistor alone cause some voltage drop due to its threshold voltage while passing “1”. This however should not be a problem if number of rows stay relatively small and inverter that accepts this signal is made as small as possible to further reduce capacitance.

### 3.1.2 Row circuitry

Row circuitry consists of a decoder, write word line (*wwl*) drivers and read word line (*rwl*) drivers. Lines *wwl* and *rwl* are driven by simple inverters placed next after NAND gates. In order to simplify connectivity and avoid excessive area, row circuitry have to be the same height as the memory array. This means that all inverters and NANDs in each row should be the same height as the height of the memory cell. To use the Method of Logical Effort in search of fastest topology, logic gates must be scalable. In the case of inverters, they can be easily scaled in both vertical and horizontal directions separately (Figure 8).

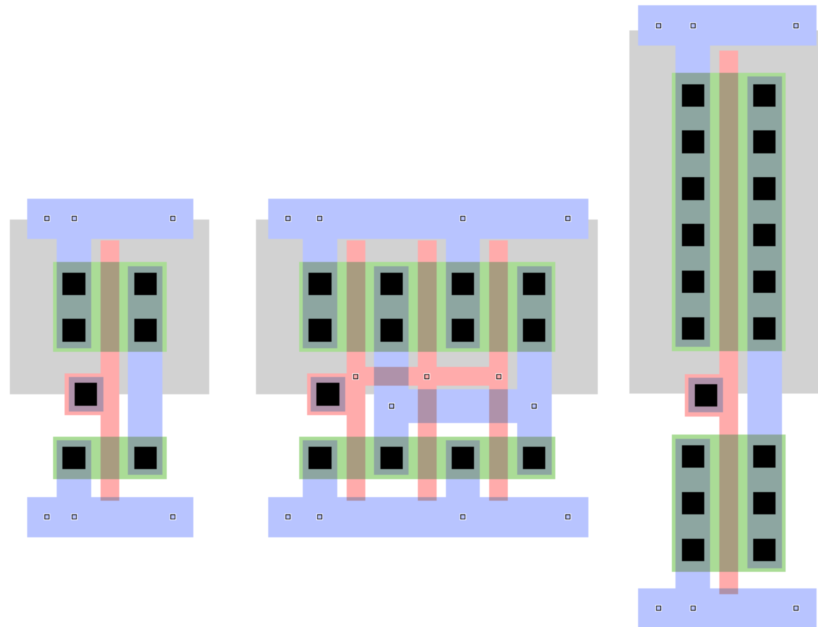

***Figure 8.*** *From left to right: x1 inverter, x3 inverter (folded), x3 inverter (unfolded).*

However, in case of NAND gates, they start to grow in both directions with increasing number of inputs if scaled appropriately [12, p. 55]. For all abovementioned reasons, only inverters were implemented as scalable logic gates in folded style.

### 3.1.3 Column circuitry

Column circuitry mainly consists of inverter drivers. Drivers for *wr, rd* and *bit_out* consist of two inverters in series thus producing a true value and can be treated as buffers. First inverters in all these drivers are made as small as possible to reduce input capacitance. In case of *wr* and *rd* this is done due to the fact, that signals that drive them are coming from outside circuits and wires that carries these signals, in turn, can have a large capacitance. In case of *bit_out* bus, as was mentioned earlier, stored “1” whenever read is experiencing voltage drop due to access transistor’s threshold voltage and long wire’s parasitic resistance and capacitance. For these reasons, inverter that is driven by this “weak 1” has to be as small as possible to reduce the overall input capacitance. Address and *bit_in* busses must produce both true and complementary values and thus consist of two inverters in series with one inverter in parallel. First inverters in a true side were made small for the same reasons as for *wr* and *rd*.

## 3.2 SRAM optimization

To make SRAM delay-optimized within a given number of words and number of bits per word, logic gates must be scalable and scaled appropriately. Because of some constraints that origin from the SRAM's topology, only inverters can be made scalable. This chapter shows what needs to be done in order to apply the Method of Logical Effort to find optimal inverter sizes. It also shows how to use straightforward method for finding these sizes. This method, however, suffers from its own constraint and is time consuming. It is still applicable to given SRAM topology and is primarily used to confirm the Logical Effort Method workability.

### 3.2.1 Building empirical model for optimal inverter sizing

To ensure that the Method of Logical Effort will work, a more reliable, empirical and comparative model had to be developed. The most simple and straightforward way is to simulate read and write delays with different memory and logic gate sizes and choose size that gives the least delay. As an example, to find an optimal inverter size that drives *wwl* with a given number of bits in a word, a circuit with differently scaled inverters must be generated. Then delays between assertion of *wr* and appearance of a bit in a cell has to be measured and inverter size that gives minimal delay must be chosen. Operation then must be repeated with another number of bits in a word.

It is worth mentioning that such method will only work fine if the number of gates with variable size is equal to one. In the case from Figure 9 NAND that drives inverter has the fixed size and thus the fixed input capacitance. Inverter that follows after NAND drives fixed load which is accumulated from SRAM cells. Therefore, there is only one logic gate which size can be varied. If there were for example two, three or more inverters in series, then, with a such brute force method, every combination of different inverter sizes would have had to be simulated; which is impractical. In developed circuit, there are such stages with two inverters in series. These are *rd*, *wr* and inverters in true side in *addr* and *bit_in*. For the sake of simplicity, first inverters in these stages were scaled to 1 and only second inverters' sizes could be varied.

Once optimal buffer sizes are found they can be used in a function estimation. This function can be used in a generation of memory of different sizes. As the Method of Logical Effort (13, 14) and measured data points (Table 1 and 2) would suggest that inverter and memory size will have a non-linear dependency.

The Method of Logical Effort tells that the stage effort is the root function of a number of stages (13). Thus, the root function will be a fitting function (16).

$$Inverter\ size = a\sqrt{Load} \tag{16}$$

The coefficient $a$ is a scaling factor to be found and $Load$ is an accumulative load capacitance of a given number of words or bits/word. The total square error between measured data points (Table 1 and 2) and fitting function can be expressed as:

$$E = \sum \left(a\sqrt{Load} - Measured\ inverter\ size\right)^2 \tag{17}$$

Finally, to minimize the error, equation (17) must be partially derivated with respect to $Load$ and equated to zero.

### 3.2.2 Adapting the Method of Logical Effort for optimal inverter sizing

In the previous section, in order to find optimal logic gate sizes, circuit had to be simulated many times while varying its size and adjusting logic gates until fastest size was found. In the present work's case where memory size is relatively small and there is only one variable gate in series, it was not over time-consuming. However, this method becomes labour intensive with large circuits and with more than one gate in series.

One possible solution to address this problem is the Logical Effort Method. In this work, it was adapted only for inverters that drive *wwl*, *rwl* (Figure 9) and *rd* (Figure 5) lines. However, this method can be used to any (scalable) logic gate in the circuit.

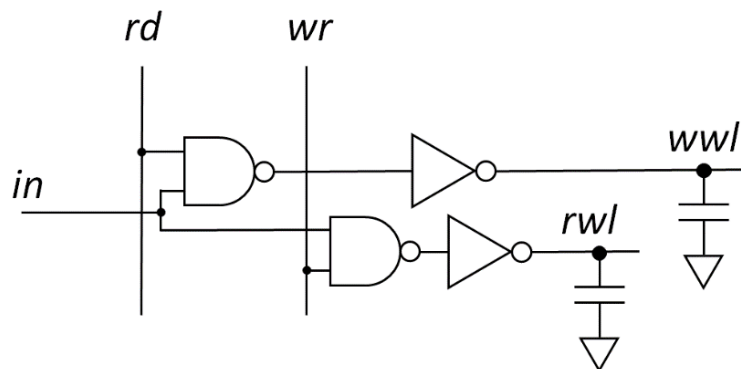


***Figure 9.*** *Part that comes after row decoder and responsible for reading and writing bits into cells.*

First, capacitances of logic gates and load should be estimated. Inverter's and NAND's input capacitances can be found using the testbench from [8, p. 308].

Capacitive load for both *wwl* and *rwl* from Figure 9 consist of two components: the parasitic capacitance of the metal wire that connect SRAM cells in a word and the diffusion capacitance of access transistors. To estimate these load capacitances for both *wwl* and *rwl* in a single cell testbench (Figure 10) was developed and used.

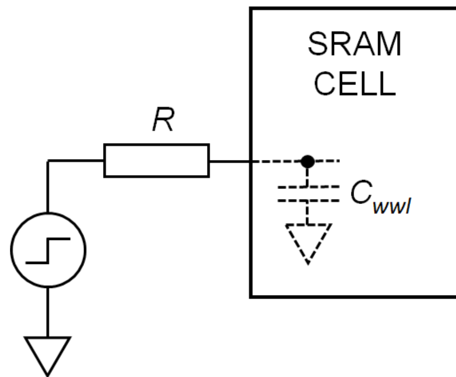


***Figure 10.*** *Testbench for estimating load capacitances of wwl and rwl in SRAM cell.*

Testbench consists of the step function voltage source and the resistor with some known resistance. The cell's *wwl* (or *rwl*) load capacitance can be calculated from the propagation delay as follows:

$$t_{pd} = \ln 2 \cdot RC \tag{16}$$

$$C = \frac{t_{pd}}{\ln 2 \cdot R} \tag{17}$$

Finally, as formula (14) would suggest, NAND's logical effort must be known. As a reminder, the logical effort of an inverter is equal to 1. To find the logical effort of NANDs, placed before inverters that drive *wwl* and *rwl* lines, testbench from [8, p. 316] was used. All obtained results and calculations based on them are presented in next chapter (Chapter 4).

# 4. RESULTS AND DISCUSSION

Optimal inverter sizes were found for a different number of words and bits/word. Size of some of the gates depends only on the number of words but does not depend on the number of bits/word. Example of such inverter is one that drives global *wr* or *rd* signals. On the contrary, sizes of *wwl* or *rwl* drivers depend only on the number of bits. In other words, inverters that drive vertical lines depend on a number of words. All these line drivers consist of two inverters in series. Results presented in Table 1 relate to second inverters, while first one was scaled to 1.

***Table 1.*** *Optimal sizes of second inverter in different lines depending on load*

| # words | *rd* | *wr* | *addr* | *bit_in* |
|---|---|---|---|---|
| 8 | 3 | 3 | 3 | 2 |
| 16 | 4 | 4 | 4 | 2 |
| 32 | 6 | 5 | 6 | 3 |
| 64 | 8 | 8 | 8 | 4 |

Sizes of horizontal line drivers depend on the number of bits. These drivers are *rwl* and *wwl* and their sizes can be found in Table 2.

***Table 2.*** *Optimal sizes of inverter in different lines depending on load*

| # bits/words | *rwl* | *wwl* |
|---|---|---|
| 8 | 3 | 4 |
| 16 | 4 | 6 |
| 32 | 7 | 8 |
| 64 | 10 | 11 |

Inverter that comes after address NAND in the decoder is also driving horizontal lines (2 NAND's from Figure 9). However, its size does not depend (as the Logical Effort Method and empirical data would suggest) on the size of the memory, because it lies between two fixed sized NAND's. The optimal size of this inverter, regardless how many bits and how many words there are in the memory, is always equal to 2.

Absolute read and write delay values for the fixed-sized memory (32 words and 32 bits/word) and differently sized inverters that drive *wwl* and *rwl* lines were measured as well. Before measurement, the memory array was filled with sequences of ones and zeros in such way that: 1 word – 1010…1, 2 word – 0101…0 and so on. The read delay was measured from asserting address on *addr* and appearance of the stored value

on *bit_out*. During the write delay, the word with complement values was asserted in *bit_in* and time difference between address assertion and value appearance in a cell was measured. All obtained values can be found in Table 3.

***Table 3.*** *Absolute read and write delay values for 32x32 SRAM memory with differently sized inverter drivers*

| Inverter size | Delay (ns) | |
|---|---|---|
| | ***rwl*** | ***wwl*** |
| 3 | 1.0440 | 0.64903 |
| 4 | 1.0261 | 0.61255 |
| 5 | 1.0210 | 0.59426 |
| 6 | 1.0201 | 0.58498 |
| 7 | 1.0231 | 0.58098 |
| 8 | 1.0270 | 0.58013 |
| 9 | 1.0324 | 0.58143 |
| 10 | 1.0381 | 0.58425 |
| 11 | 1.0432 | 0.58808 |
| 12 | 1.0484 | 0.59275 |

It can be seen from Table 3 that delay variation caused by different inverter sizes is not crucial. Moreover, circuits commonly experience different sorts of environmental impacts. Some of these, such as temperature rise inside or outside a chip or a voltage drops may slow down a circuit operation. To see how delay, due to these impacts, is comparable to delay caused by non-optimal inverter size same circuit with optimal inverter sizes were simulated with lower voltages and higher temperatures (Table 4). Delay, as previously, was measured in ns.

***Table 4.*** *Delay variations caused by temperature rise and voltage drop.*

| | Vdd \ t | 25 °C | 50 °C | 75 °C |
|---|---|---|---|---|
| *rwl* | **1.8 V** | 1.0231 | 1.0460 | 1.0639 |
| | **1.7 V** | 1.1067 | 1.1293 | 1.1414 |
| | **1.6 V** | 1.2203 | 1.2376 | 1.2491 |
| *wwl* | **1.8 V** | 0.58013 | 0.59585 | 0.61135 |
| | **1.7 V** | 0.61716 | 0.63338 | 0.64914 |
| | **1.6 V** | 0.66263 | 0.67883 | 0.69464 |

It can be seen, that even small voltage drop and/or rise in temperature has more significant impact on the delay than erroneously chosen inverter size. Thus, for example, if temperature rises from 25 °C to 50 °C and supply voltage drops from 1.8 V to 1.7 V, it will cause the delay rise in optimally chosen inverter that drives *wwl* from 0.58013 ns to 0.63338 ns which is 9,1%. Simultaneously, if the inverter size is two times smaller than optimal (4 instead of 8), the delay grows from 0.58013 ns to 0.61255 ns which is 5,6%.

Finally, the Logical Effort Method was adapted to inverters that drive *wwl*, *rwl* (Figure 8) and *rd* (Figure 5). All measurements required for the adaptation were extracted from simulations using testbenches mentioned in previous chapter (Chapter 3). These results with explanations can be seen in Table 5.

***Table 5.*** *Extracted testbench results needed for the Logical Effort Method*

| Description | Symbol | Value |
|---|---|---|
| **Unit inverter's input capacitance** | $C_{in,inv}$ | 2.4687 fF |
| **NAND's first input capacitance (global)** | $C_{in1,NAND}$ | 3.0796 fF |
| **NAND's second input capacitance** | $C_{in2,NAND}$ | 2.4769 fF |
| **SRAM cell's *wwl* capacitance** | $C_{wwl,CELL}$ | 2.5183 fF |
| **SRAM cell's *rwl* capacitance** | $C_{rwl,CELL}$ | 1.5331 fF |
| **NAND's logical effort** | $g_{NAND}$ | 1.043 |

By using all simulated values, sizes of inverters that drive *wwl* depending on the number of words can be found in the following way. First, the path effort (5) must be found. There are two identical NAND's in parallel so the branching effort $B$ is equal to 2. The logical effort of the NAND was $g_{NAND} = 1.043$ and the logical effort of the inverter is 1 so the total path logical effort is $G = 1.043 \cdot 1 = 1.043$. The path electrical effort is equal to output to input capacitance ratio. The output capacitance is a number of cells in a word times cell's *wwl* capacitance. If there are $x$ cells in a word, then the output capacitance is $C_{out} = 2.5183x$ fF. The input capacitance consists of the doubled NAND capacitance because there are two NAND's connected to the same node and is $C_{in} = 2 \cdot 2{,}4769 = 4{,}9538$ fF. Therefore, the path electrical effort is $H = \frac{C_{out}}{C_{in}} = \frac{2.5183x}{4{,}9538} = 0.5083x$ and the path effort is $F = GBH = 1.043 \cdot 2 \cdot 0.5083x = 1.0603x$. There are two stages so the equalized stage effort is $\hat{f} = \sqrt{F} = \sqrt{1.0603x} = 1.03\sqrt{x}$. Finally, to find the optimal inverter size, the capacitance transformation formula (14) must be used. To drive a word with $x$ cells with least delay, the inverters' input capacitance must be equal to $C_{in,inv} = \frac{C_{out} \cdot g_{inv}}{\hat{f}} = \frac{2.5183x \cdot 1}{1.03\sqrt{x}} = 2.44\sqrt{x}$ fF. To map the capacitance back to size, calculated value must be divided by a unit inverter's capacitance and thus $Inverter\ size_{wwl} = \frac{2.44\sqrt{x}}{2.4687} = 0.9883\sqrt{x}$. By making similar actions the same value for *rwl* was found to be $Inverter\ size_{rwl} = 0.5465\sqrt{x}$. In case of *rd,* the capacitive load contributes to the NAND's input capacitance and depends on the number of words.

Thus, if there are $x$ words, then load is equal to $3.0796x$ fF. If first inverter is scaled to 1, then the path effort is $H = \frac{3.0796x}{2.4687} = 1.2474x$. There are only two inverters in series and thus the branching effort $B = 1$, $G = 1$ and $\hat{f} = \sqrt{1.2474x} = 1.1169\sqrt{x}$. Size of the second inverter must be $Inverter\ size_{rd} = \frac{3.0796x}{1.1169\sqrt{x}} : 2.4687 = 1.1168\sqrt{x}$.

To validate above calculated results, model, based upon experimental data from Table 2 and partially from Table 3, can be used. The square root fitting function is in a form of $Inverter\ size = a\sqrt{n_{bits}}$, where $a$ is a coefficient to be found. Cumulative difference between this function and data points from Table 2 in square is a total error (16) to be derivated and equated to zero. For *rwl* it can be found as follows:

$$E = \left(a\sqrt{8} - 3\right)^2 + \left(a\sqrt{16} - 4\right)^2 + \left(a\sqrt{32} - 7\right)^2 + \left(a\sqrt{64} - 10\right)^2 \qquad (18)$$

By taking partial derivative of (18) with respect to $a$ and equating it to zero following equation is obtained:

$$120a - 68\sqrt{2} - 192 = 0 \qquad (19)$$

This, in turn, leads scaling coefficient $a_{rwl}$ to be 1.2. The same procedure was repeated for *wwl* and coefficient was equal to $a_{wwl}$=1.4. Finally, for *rd* it was $a_{rd}$=1.02.

All obtained results for *wwr*, *rwl* and *rd* can be seen below (Figures 12, 13 and 14).

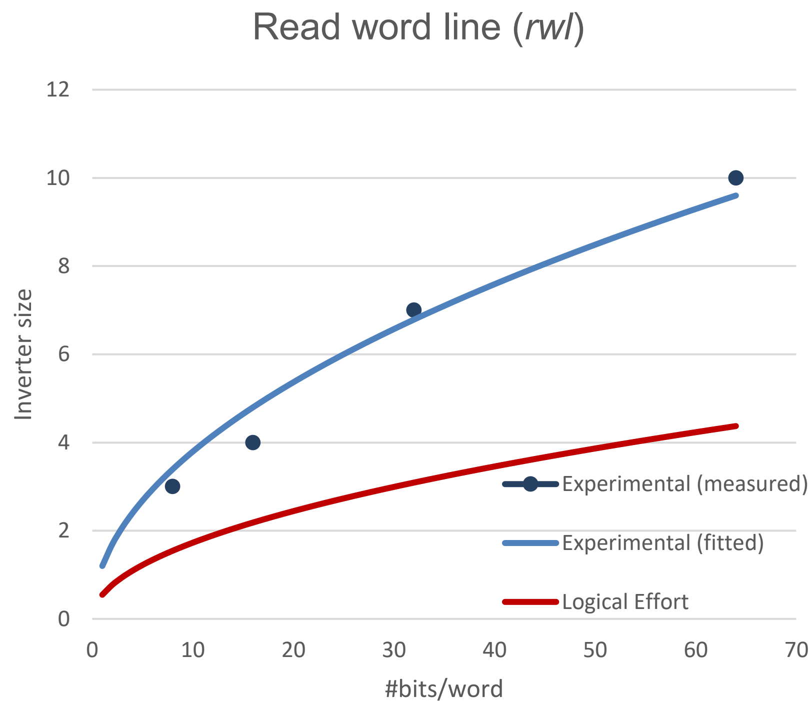


***Figure 11**. Comparative results for rwl.*

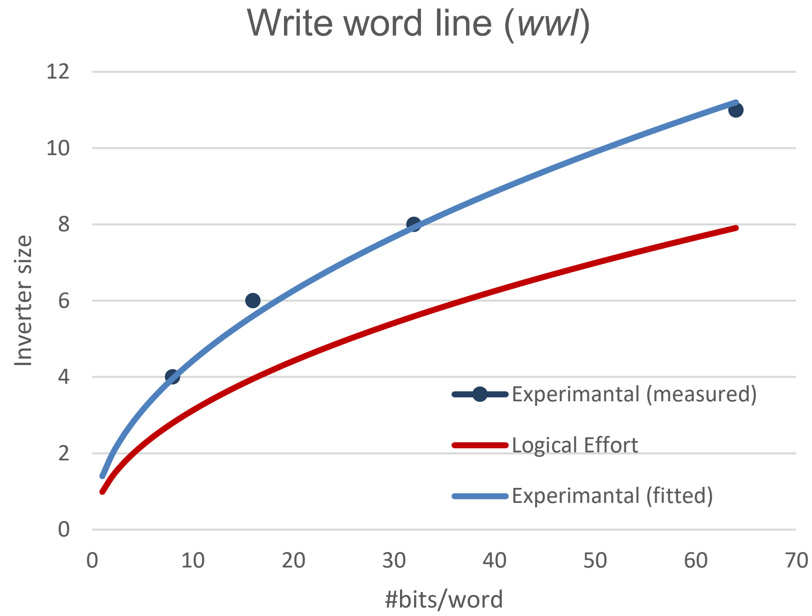


***Figure 12.*** *Comparative results for wwl*

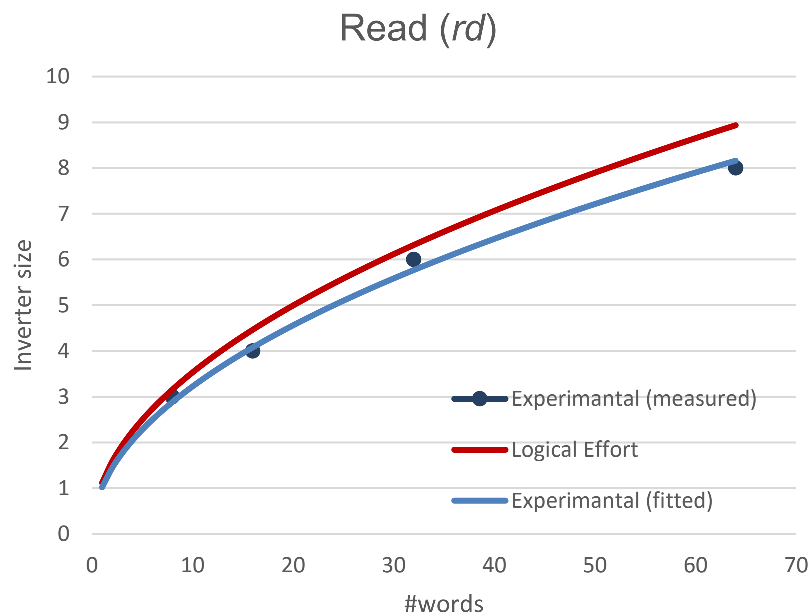


***Figure 13.*** *Comparative results for rd.*

It is evident from Figures 11 and 12 that results are not ideal. The Logical Effort Method always suggests making inverter size smaller than optimal. However, in case of *rd,* curves are close enough to each other.

The results for *rwl*, *wwl* and *rd*, and how capacitances for each part were extracted, would suggest, that parasitic capacitances of the cell were underestimated. Memory cell has a very tight and complex morphology. This probably leads to a capacitive coupling and other parasitic effects such as Millers capacitance [13]. For these reasons, obtaining of more precise results, would require development of a more sophisticated testbench.

# 5. CONCLUSION

In this work configurable SRAM generator with scalable logic gates was implemented. Needed capacitances and other values of logic gates and memory cell were estimated by simulation using testbenches described in [8]. The Logical Effort Method was adapted to developed SRAM and its architectural constraints. The additional empirical model was developed upon simulation results in order to verify the Logical Effort Method's workability. The difference in results was studied and its significance was evaluated.

The results for the second inverter in *rd* line have shown that the Logical Effort Method gives accurate estimation for delay-optimal logic gate sizes. Even if size is not fully optimal, additional delay caused by it is in order of picoseconds and thus is not significant. In addition, the delay caused by a small voltage drop and temperature rise have caused more significant impact on a delay, than non-optimal logic gate size. For example, temperature rise from 25 °C to 50 °C with supply voltage drop from 1.8 V to 1.7 V, have caused delay growth in 9,1%. In the meantime, inverter that is two times smaller than optimal (4 instead of 8) have caused delay growth only in 5,6%.

However, the results for *rwl* and *wwl* line drivers were not that accurate and inverters were undersized. Even though model have not produced fully optimal inverter sizes, they are still applicable and will not cause large additional delay if word's length stays relatively small.

All abovementioned results and methods used to estimate capacitances suggest that *rwl* and *wwl* driver sizes were deviated because SRAM cell's parasitic capacitances were underestimated. Thus, the more sophisticated method in parasitic capacitance extraction is required. As an example, approach used in the testbench for extracting capacitances from logic gates may be used. Parallel inverters driven by same driver should be connected to a line with interested parasitic capacitance and test capacitor. Parasitic capacitance can be found by sweeping test capacitor value and comparing delays until they are equal. Also, to make results more realistic, word of 16 bits should be generated. Then sequence of "1"s and "0"s should be written into the word and then obtained value for the capacitance can be divided by 16.

All in all, the Logical Effort Method can be used in a design of a custom or semi-custom circuit such as SRAM, where scalable logic gates can be adjusted to achieve the fast-

est topology. This work can serve as a guide during development of such circuit. However, care should be taken during capacitance extraction from involved logic gates and structures that act as load. Although even rough estimation can still be applicable if timing is not critical. Finally, obtained results can be easily validated and model can be adjusted accordingly by sweeping size of the interested logic gate in both directions and by comparing obtained delays.

# REFERENCES


[1] J. Sell, The Xbox One X Scorpio Engine, IEEE Micro, Vol. 38, No. 2, Mar/Apr 2018, pp. 53-60.

[2] D. Harris, R. F. Sproull, I. Sutherland, Logical Effort: Designing Fast CMOS Circuits, San Francisco, Calif: Morgan Kaufmann Publishers, 1999.

[3] S. N. Panda, S. Padhi, V. Phanindra, U. Nanda, S. K. Pattnaik and D. Nayak, Design and implementaton of SRAM macro unit, 2017 International Conference on Trends in Electronics and Informatics (ICEI), Tirunelveli, 2017, pp. 119-123.

[4] C. Barry, Modern Embedded Computing: Designing Connected, Pervasive, Media-Rich Systems. Modern Embedded Computing, St. Louis: Elsevier Science & Technology, 2012, pp. 66-67.

[5] P. Athe and S. Dasgupta, A comparative study of 6T, 8T and 9T decanano SRAM cell, 2009 IEEE Symposium on Industrial Electronics & Applications, Kuala Lumpur, 2009, pp. 889-894.

[6] P. N. V. Kiran and N. Saxena, Design and analysis of different types SRAM cell topologies, 2015 2nd International Conference on Electronics and Communication Systems (ICECS), Coimbatore, 2015, pp. 1060-1065.

[7] P. Sharma, R. Anusha, K. Bharath, J. K. Gulati, P. K. Walia and S. J. Darak, Quantification of figures of merit of 7T and 8T SRAM cells in subthreshold region and their comparison with the conventional 6T SRAM cell, 2016 20th International Symposium on VLSI Design and Test (VDAT), Guwahati, 2016, pp. 1-2.

[8] D. Harris, N. Weste, CMOS VLSI Design: a Circuits and Systems Perspective, 4th ed., Boston (MA): Pearson/Addison-Wesley, 2010.

[9] H. Kaeslin, Digital integrated circuit design: from VLSI architectures to CMOS fabrication, Cambridge University Press, 2008.

[10] H. Yamauchi, A Discussion on SRAM Circuit Design Trend in Deeper Nanometer-Scale Technologies, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 18, No. 5, May 2010, pp. 763-774.

[11] R. Baker, CMOS: Circuit Design, Layout, and Simulation, Third Edition, Vol. 17, Wiley, 2010.

[12] D. Clein, G. Shimokura, CMOS IC layout: concepts, methodologies, and tools, Newnes, 2001.

[13] K. Brzozowski, The miller effect in digital CMOS gates and power consumption analysis, 2012 International Conference on Signals and Electronic Systems (ICSES), IEEE, Sep. 2012, pp. 1–6.