

University of Windsor

Scholarship at UWindor

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

2006

Optimizing power, delay and reliability for digital logic circuits with CMOS and single-electron technologies.

Jialin Mi

University of Windsor

Follow this and additional works at: <https://scholar.uwindsor.ca/etd>

Recommended Citation

Mi, Jialin, "Optimizing power, delay and reliability for digital logic circuits with CMOS and single-electron technologies." (2006). *Electronic Theses and Dissertations*. 2954.

<https://scholar.uwindsor.ca/etd/2954>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

Optimizing Power, Delay and Reliability for Digital Logic
Circuits with CMOS and Single-Electron Technologies

by

Jialin Mi

A thesis

Submitted to the Faculty of Graduate Studies and Research
through the Department of Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada

2006

© Jialin Mi



Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*
ISBN: 978-0-494-17106-6
Our file *Notre référence*
ISBN: 978-0-494-17106-6

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.


Canada

Abstract

In this thesis, we present two low power approaches with consideration of delay and/or reliability. The first approach is based on CMOS (Complementary Metal-Oxide Semiconductor) technology. Given a gate level topology of digital circuits and a target library, we propose a greedy algorithm for delay budgeting in order to optimize power dissipation. The algorithm is implemented with JAVA SDK. The developed software tool estimates how much power dissipation (percentage) can be saved without increasing the circuit delay, and the potential of power savings by relaxing the circuit's timing constraints.

The second low power approach is proposed with SET (Single Electron Tunneling) technology. We focus on an elementary logic structure called threshold gate, and present a standard procedure of logic implementation, with analysis of delay, power and reliability due to background charge effect. As an application example, an FSM (Finite State Machine) for RFID (Radio Frequency Identification) system is designed and simulated successfully.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Chunhong Chen for his constant support, guidance and motivation.

I am grateful to the committee members Dr. Mohammed Khalid and Dr. Xueyuan Nie for providing valuable feedbacks.

I would like to thank Ms. Andria Turner and Ms. Shelby Marchand for seamless administrative assistance.

This research work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

Contents

Abstract	iii
Acknowledgements	iv
List of Figures	viii
List of Tables	x
1. Introduction to Power Analysis and Optimization	1
1.1 Needs for Low Power Integrated Circuits	1
1.2 Dynamic and Static Power Dissipation	2
1.3 Levels of Power Analysis and Optimization	3
1.4 Organization of Thesis	6
2. Power-Oriented Delay Budgeting for CMOS Circuits	8
2.1 Introduction to Delay Budgeting	9
2.1.1 Zero-Slack Algorithm	11
2.1.2 Maximal Independent Set Algorithm	11
2.1.3 Greedy Algorithm	13
2.2 Delay Budgeting Problem Formulation	15
2.3 Power-Oriented Delay Budgeting Algorithm	16
2.4 Experiment on Benchmark Circuits	21

2.5	Summary	24
3.	Single-Electron Tunneling (SET) Threshold Logic	25
3.1	Introduction to Single-Electron Tunneling	25
3.1.1	Theoretical Background	26
3.1.2	Fabrication Limitation	27
3.2	SET Threshold Gate Structure	27
3.3	Feasible Parameters of SET Threshold Logic	29
3.4	Capacitance Parameter Selection	31
3.5	Parameter Optimization for Reliability	32
3.5.1	Reliability Analysis with Background Charges of Uniform Probability Distribution	34
3.5.2	Reliability Analysis with Background Charges of Normal Probability Distribution	37
3.5.3	Relationship between Gate Reliability and Circuit Parameters	40
3.5.4	Parameter Scaling	42
3.6	Summary	42
4.	A Finite State Machine (FSM) Implementation with SET Threshold Gates	44
4.1	Radio Frequency Identification (RFID)	45
4.2	RFID Anti-Collision Protocols	46
4.2.1	Binary-Tree Scheme	47
4.2.2	Query-Tree Scheme	47
4.3	SET Implementation for Binary Tree Protocol	48
4.3.1	Binary-Tree Protocol FSM Structure	48
4.3.2	SET Positive Edge-Triggered D-Flip-Flop (DFF)	50
4.4	Simulation with SIMON (SIMulation Of Nanostructure)	51
4.5	Delay and Power Estimation	56
4.6	Summary	56

5. Conclusions and Future Work	57
Appendix A Program 1 – Power Oriented Delay Budgeting Greedy Algorithm with GUI (Chapter 2)	59
Appendix B Program 2 – SET Threshold Gates Parameter Selection and Optimization (Chapter 3, 4)	136
References	154
VITA AUCTORIS	159

List of Figures

Fig.1.1. Low power analysis accuracy and optimization potential at various abstraction levels	5
Fig.2.1. Two delay budget assignments to the same circuit	10
Fig.2.2. Slack distribution and slack assignment	11
Fig.2.3. Effect of node delay on the slack of its fanin and fanout	12
Fig.2.4. An eight-node subcircuit example	19
Fig.2.5. Java Implementation of power-oriented delay budgeting greedy algorithm	23
Fig.3.1. SET threshold logic structure	28
Fig.3.2. SET buffer/inverter	29
Fig.3.3. The selected values of buffer/invertor from [22]	31
Fig.3.4. Reliability of a 2- input AND gate with uniform distribution of background charges	37
Fig.3.5. Reliability of a 2- input NOR gate with uniform distribution of background charges	37
Fig.3.6. Reliability of a 2- input AND gate with normal distribution of background charges	40

Fig.3.7. Reliability of a 2- input NOR gate with uniform distribution of background charges	40
Fig.3.8. Reliability-versus- C_b curves with different probability distribution of background charges	41
Fig.4.1. State diagram for Binary-tree protocol	49
Fig.4.2. Logic-level implementation of the state machine for Binary-tree protocol	50
Fig.4.3. Positive edge-triggered D-flip-flop	51
Fig.4.4. SIMON 2.0 Graphic User Interface	52
Fig.4.5. SET implementation of the state machine for Binary-tree protocol	53
Fig.4.6. Simulation result of Fig.4.5.	54

List of Tables

Table.2.1. The available library information for example graph of Fig.2.4.	19
Table.2.2. Demonstration of the Greedy Algorithm on Fig.2.4.	20
Table.2.3. Power consumption with the proposed algorithm on a set of benchmark circuits	22
Table.2.4. Power consumption using the algorithm from [2]	22
Table.3.1. Threshold expressions	30
Table.3.2. Capacitance parameters for different gates given in Table.3.1.	32
Table.3.3. Reliabilities of 2-input AND gate with uniform distribution of background charges	36
Table.3.4. Reliabilities of 2-input NOR gate with uniform distribution of background charges	36
Table.3.5. Reliabilities of 2-input AND gate with normal distribution of background charges	39
Table.3.6. Reliabilities of 2-input NOR gate with normal distribution of background charges	39
Table.4.1. State assignment	49

Table.4.2. Threshold logic gates in Fig.4.3.	51
Table.4.3. Capacitance parameters ($\times 10^{-19}$ F) for threshold gates in Fig.4.5.	55
Table.4.4. State transitions in Fig.4.6.	55

Chapter 1

Introduction to Power Analysis and Optimization

The art of power analysis and optimization of integrated circuits is now appearing in the mainstream design community affecting all aspects of the design process. The interests in low-power chips and systems are driven by both business and technical needs. The industry for low power consumer electronic products is booming with a rapidly expanding market. At the same time, newer generations of semiconductor processing technologies present more stringent requirements to the power dissipation of digital chips due to increased device density, speed and complexity. [1]

1.1 Needs for Low Power Integrated Circuits

Integrated circuits were first introduced by Texas Instruments half a century ago. In the early years, device density and operating frequency were so low that no one would consider power dissipation to be a constraining factor in a chip. Besides the environment and energy concern of power consumption of a chip, high power dissipation makes a chip

so hot that it simply burns itself out without extra efforts to cool it down, such as fans and liquid cooling pumps. And those cooling technologies have their own limitations and require further development. Simply put, the needs for low power solutions arise from the evolution of integrated circuits itself.

Another factor that fuels the needs for low power integrated circuits is the huge and still increasing demand for portable consumer electronics powered by batteries. This market ranges from laptop computers, cellular phones to personal GPS systems. For those portable devices, if the batteries cannot support their power consumption for a certain long enough period, the facilities brought by those devices will be much less meaningful, if not totally meaningless. Furthermore, for portable devices, the power consumption of integrated circuits can be more important than the running speed, if a tradeoff has to be made. Take laptop computer CPUs for example, mobile CPUs normally specify with lower operating frequency and lower voltage supply compared with desktop computer CPUs. The growing portable market hungrily demands low power integrated circuits.

The demand for low power integrated circuits also comes from the emerging technologies, such as passive mode devices. A passive mode circuit is not equipped with a battery. It receives power, along with data and command signals, from electromagnetic waves. For instance, an RFID (Radio Frequency Identification) system requires passive mode circuits on its tags. As there are environmental regulations all around the world, which set limitations to maximum strength of electromagnetic field covering all frequency bands, the only attainable solution to supply enough power to passive mode circuit is to make it consume less. Therefore, low power integrated circuits are critical in their implementation.

1.2 Dynamic and Static Power Dissipation

All power dissipation can be generally classified into two types: dynamic and static, according to how and where it happens.

Dynamic power dissipation is caused by switching activities of the circuits. As each switching activity consumes certain amount of energy, a higher operating frequency leads to more frequent switching activities in the circuits and increases dynamic power dissipation, which is energy dissipation per switching activity multiplied by frequency. The most significant source of dynamic power dissipation in circuits is the charging and discharging of capacitance, including those of capacitor components and parasitic capacitance of interconnection wires and transistors.

Static power dissipation is, on the contrary, not related to the switching activities. When a CMOS digital circuit holds its logic states, it still consumes certain amount of power due to the current leakage. Static power dissipation is not strongly related to the circuit operating frequency.

1.3 Levels of Power Analysis and Optimization

An IC design process is abstracted into hierarchical levels in a standard top-down design flow. In order to achieve low power dissipation for final product, power estimate, analysis and optimization is applied to all levels with different methodologies.

Device Level: For a digital circuit design, each fabrication technology standard is presented by a digital cell library. To build a new library is by no means a trivial project, and normally beyond the concern of a circuit designer. With the development of electronics industry, new generations of digital library will emerge. For a specific circuit design project, a predetermined digital library leaves nearly no space for an individual designer to do any power analysis and optimization at this level. New technologies,

however, do improve power dissipation performance considerably. For instance, applied onto the same gate level netlist, CMOS $0.18\ \mu\text{m}$ @ 1.8V may consume less than 1/5 of that of $0.35\ \mu\text{m}$ @ 3.3V . Furthermore, new devices other than MOS transistors, such as single electron transistors (SETs), resonant tunneling diodes (RTDs), and quantum dots and quantum cellular automata (QCA) have been proposed to bring ultra-low power integrated circuits in the future. In this thesis, we discuss SET device and logic in Chapter 3 and Chapter 4.

Circuit Level: A circuit is built up with primitive elements such as resistors, capacitors, and inductors. More complex device models such as diodes and transistors are constructed from the basic components. SPICE (Simulation Program with IC Emphasis)-like application software packages are heavily used to simulate the circuit and estimate the power consumption. On this level, devices parameters are adjustable and can be used to reduce power dissipation. For digital circuit design, given a cell library, power reduction may be achieved by transistor and gate sizing, equivalent pin ordering, network restructuring, building special latches and flip-flop, etc. In Chapter.2, we propose a power oriented delay budgeting greedy algorithm to do gate sizing given a gate level netlist and a standard cell library. At the circuit level, percentage power reduction in the teens is considered good.

Logic Level: There are always more than one gate level implementations available to fulfill the same logic functions, however, probably with different switching activities. More switching activities introduce more dynamic power dissipation, which is normally dominant in a digital circuit. In logic level, reduction of switching activities is the most prevalent theme of power optimization techniques. Some switching activities are the result of undefined behavior of a logic system. Such switching activities not relating to the proper operation of the circuit should be reduced or eliminated if possible. Techniques

such as gate reorganization, signal gating, logic encoding and state machine encoding have significant effects on the power dissipation of a digital circuit.

Architecture and System Level: This level includes RTL (register transfer level) and macro hardware block level. In today’s digital design flow, design functionalities are described by circuit designers with a hardware description language such as Verilog or VHDL. Any decision made in this level will heavily affect the lower logic, circuit, device levels, and thus largely effect the circuit power consumption. Power management of performance and throughput of a system based on its computation needs, choice of clocking strategy, component organization, parallel architecture with voltage reduction, etc., are issues to be considered at this level.

As shown in Fig.1.1., generally speaking, on a higher level, power analysis is less accurate than on a lower level, since less detailed information about the design is decided on a higher level. However, because of the same reason, on a higher level, we have more optimization potential to improve the low power performance.



Abstraction level	Analysis accuracy	Optimization potential
Algorithm	Worst	Most
Software and system		
Hardware behavior		
Register transfer		
Logic		
Circuit		
Device	Best	Least

Fig.1.1. Low power analysis accuracy and optimization potential at various abstraction levels

1.4 Organization of Thesis

In this thesis, we present two low power approaches with consideration of delay and/or reliability, one for CMOS technology, and the other for SET technology.

Chapter 2 is dedicated to CMOS technology. For a digital circuit, given its gate level topology and a cell library, we propose a greedy algorithm for delay budgeting in order to optimize power dissipation. The algorithm is implemented with JAVA SDK with GUI. With a specified circuit gate level topology and a chosen digital layout library as input, the software tool can estimate how much power dissipation (percentage) can be saved within a given critical path circuit delay. Also, the software tool can further estimate the potential of power savings by relaxing the critical path delay in the circuits. Based on this information, the tool can draw a circuit level power-delay tradeoff curve, which can be valuable for a designer before plunging into the details of every effort to optimize the circuit power dissipation.

In Chapter 3, we study a relatively new SET (Single Electron Tunneling) technology, which is a prospective candidate to partly replace CMOS technology in the future. We focus on an elementary logic structure called threshold gate, and present a standard procedure of logic implementation, with analysis of delay, power and reliability due to background charge effect.

In Chapter 4, as an example of digital circuit implementation based on SET threshold gate, we build and simulate an FSM (Finite State Machine) for RFID (Radio Frequency Identification) system. Assuming a clock frequency of 100 MHz, our proposed FSM consumes about 10 pW power, compared to $\sim 100 \mu\text{W}$ with a typical $0.18 \mu\text{m}$ CMOS

implementation.

Chapter 5 gives the conclusions and comments on possible research work in the future.

Chapter 2

Power-Oriented Delay Budgeting for CMOS Circuits

In the design flow of today's digital systems, circuit area, delay, and power consumption are three major concerns. Generally speaking, in order to reduce the area or power of a circuit, an increased delay is the price one has to pay, or vice versa. Recent research shows that this delay penalty can be avoided or reduced to a minimum because the circuit delay is independent of the non-critical paths where there are some delay budgets available [2-6].

However, previous research either looks at a pure delay budgeting problem for layout synthesis without keeping power optimization in mind, or explores a pure power-delay tradeoff without investigating the potential ability of optimizing the power consumption through delay budgets. In this thesis, we present a power-oriented delay budgeting algorithm which is intended to provide the best power-delay tradeoff for any given combinational circuits. As a sequential circuit can always be divided into several

combinational sub-circuits, this power-oriented delay budgeting algorithm can generally apply to any digital circuits.

Experiments on a variety of benchmark circuits show that the potential power savings from the current synthesis flow are significant with an average improvement rate of 35%.

2.1 Introduction to Delay Budgeting

Under a given timing constraint, delay budget is the extra delay a component can tolerate such that no timing constraint is violated for the whole circuit.

The delay budgeting problem was formulated as a slack distribution/assignment problem for which many algorithms have been proposed. [7] presented a zero-slack algorithm (ZSA) to assign delay budgets to signal nets for performance-driven layout. In [5], a delay upper-bound budgeting method was proposed for the net-based timing-driven placement. In [4], Kuo et al. formulated the delay-budgeting problem as a Lagrange-Multiplier-based slack assignment problem for the timing-closure-driven design. Chen et al. [2, 8] proposed the notion of potential slack and described a maximal-independent-set-based slack assignment algorithm.

To solve a delay budgeting problem, a common practice is to describe a digital circuit as a DAG (Directed-Acyclic Graph) $G = (V, E)$. A node $v \in V$ denotes a module, which could represent a logic gate or a functional block composed by a set of gates. And an edge $e \in E$ from node v_i to v_j if v_j is an immediate fanout of v_i . There is a delay $d(v)$ associated with each node v , which stands for the delay of node v itself plus the interconnection delay. A vector of delays $D(V) = [d(v_1) d(v_2) \dots d(v_n)]$ is called delay distribution of the circuit.

A well-known procedure to compute arrival time, $a(v)$, and required time, $r(v)$, for each node v is given recursively by

$$\begin{cases} a(v) = \max_{u \in FI(v)} (a(u) + d(v)) \\ r(v) = \min_{w \in FO(v)} (r(w) - d(w)) \end{cases} \quad (2.1)$$

where $FI(v)$ is a set of fanins of node v , and $FO(v)$ a set of fanouts. Slack of node v is defined as $s(v) = r(v) - a(v)$. A vector of slacks $S(V) = [s(v1) \ s(v2) \ \dots \ s(vn)]$ is called slack distribution of the circuit.

For a single node v , its slack $s(v)$ can be considered as a delay budget, which can be partly or fully released to increase the delay of node v from its original delay $d(v)$ to a new delay within the range of $[d(v), d(v) + s(v)]$, without breaking the circuit timing constraint. However, due to the dependency between the nodes, not all nodes can release their slacks as delay budgets at the same time. Fig.2.1 shows two delay budget assignments to the same circuit, while with different total budgets.

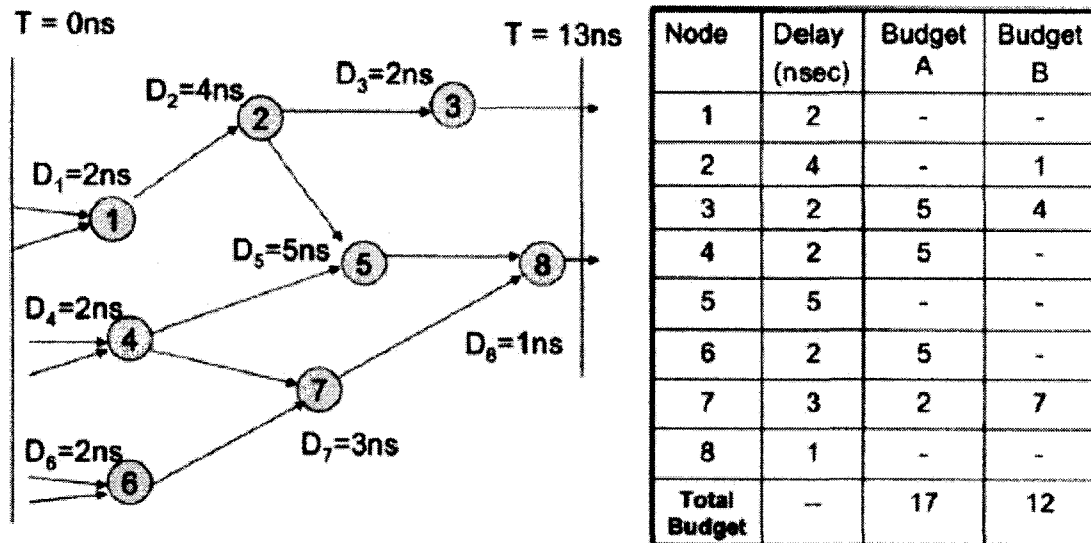


Fig.2.1. Two delay budget assignments to the same circuit

A traditional delay budget algorithm is a scheme to generate a slack assignment which brings an acceptably large, if not maximum, total budget.

2.1.1 Zero-Slack Algorithm

Given a graph, ZSA starts with nodes of minimum positive slack and locally performs slack assignment such that their slacks become zero. At each iteration, ZSA identifies a path on which all nodes have minimum slack s_{\min} , then assigns each node an additional delay s_{\min} / N_{\min} , where N_{\min} is the number of nodes on the path. In Fig.2.2, for example, path $\{v1, v3, v4\}$ is first identified, and each node on the path is assigned an additional delay $5/3$. Slacks of all nodes in the figure except $v2$ become zero, while slack of $v2$ is updated as $5/3$. After assigning additional delay of $5/3$ again to $v2$, the algorithm terminates with effective slack of $20/3$, while the maximum effective slack for this circuit is 10. This example shows that while ZSA is simple and easy to implement, it is far from optimal slack assignment.

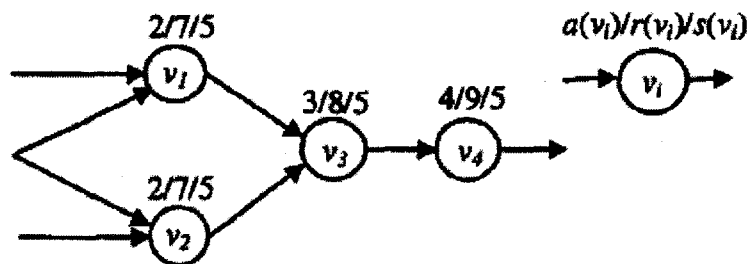


Fig.2.2. Slack Distribution and slack assignment

2.1.2 Maximal Independent Set Algorithm

Opposite to minimum slack selected in ZSA, MISA first selects nodes with maximum slack s_m , to construct a slack-equalization graph $G_m = (V_m, E_m)$, where $V_m = \{v | v \in V$

and $s(v) = s_m$ } and $E_m = \{(u,v)|(u,v) \in E, \text{ and } u, v \text{ are slack-sensitive}\}$. A transitive slack-equalization graph $G_t = (V_m, E_m)$ of G_m is a directed graph such that there is an edge $(u, v) \in E$ if and only if there is a directed path from u to v in G_m . In this way, a maximal independent set (MIS) of G_t corresponds to a set of nodes $V_{MIS} \subseteq V_m$ such that the number of nodes which are slack-sensitive to any node $v \in V_{MIS}$ is minimized.

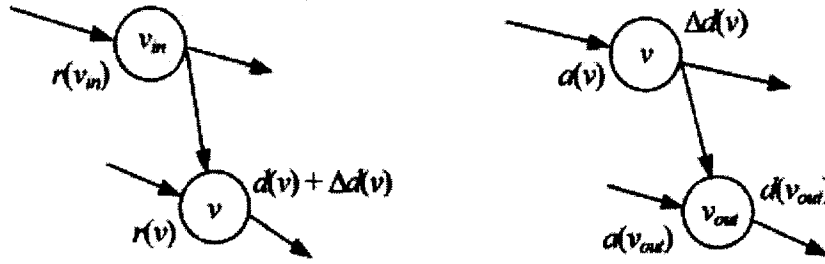


Fig.2.3. Effect of node delay on the slack of its fanin and fanout

Fig.2.3. demonstrates how an additional delay $\Delta d(v)$ applied to node v may affect the slack of fanin/ fanout nodes. The slack of v_{in} and v_{out} will be reduced only if

$$\begin{cases} r(v) - r(v_{in}) - d(v) < \Delta d(v) \\ a(v_{out}) - a(v) - d(v_{out}) < \Delta d(v) \end{cases} \quad (2.2)$$

Thus, if we choose $\Delta d(v) = s_m - s_{m-1}$, where s_{m-1} is the second largest slack in G , no nodes with slack less than s_{m-1} are slack sensitive to node v , thus the slack penalty is minimized.

Maximum-Independent-Set-Algorithm (MISA) {

Input: Graph $G = (V, E)$

and the given timing constraints

Output: Slack assignment $\Delta D(V)$

Begin


```

Compute slack for each node  $v \in V$ ;
Initialize  $\Delta D(V) = [\Delta d(v_1), \Delta d(v_2), \dots, \Delta d(v_n)] = 0$ ;
Find  $s_m$  and  $s_{m-1}$  in  $G$  and let  $\delta = s_m - s_{m-1}$ ;
While ( $\delta > 0$ ){
    Construct transitive slack-equalization graph  $G_t$ ;
    Find maximum independent set  $MIS$  of  $G_t$ ;
    Assign additional delay  $\delta$  to each node in  $MIS$ :
         $\Delta d(u) \leftarrow \Delta d(u) + \delta, \forall u \in MIS$ ;
    Update node slacks and  $\delta$  in  $G$ ;
}
End
}

```

MISA maximizes effective slack—the slack can be used to do delay budgeting. The time complexity of MISA is $O(K \cdot n^3)$, where n is the number of nodes and K the number of different slacks in a given graph.

2.1.3 Greedy Algorithm

Considering the fact that all immediate fanins (or fanouts) of a node v are always independent of each other in terms of slack sensitivity unless there are reconvergent directed edges (i.e., edges going from one fanin or fanout to another), a fast way of estimating potential slack (PS) is to assign an additional delay (which is equal to a specific slack) to either node or all of its fanins (or fanouts) recursively, depending on their slacks. By comparing these slacks, the nodes with the largest sum of slacks can be selected to be the best candidates for node for delay assignment. This largest sum of

slacks is called local PS with node v . The extra procedure we need to go through is to check if there is any reconvergent edge involved. When there exists a reconvergent edge between two nodes, we can ignore the node with less slack. When the local PSs are available, we select the maximum one for delay assignment and then recursively delete all the related fanins and fanouts from the graph. The time complexity of this algorithm is $O(n \log n)$.

Greedy Algorithm {

Input: Graph $G = (V, E)$

Output: Potential slack estimate PS

Begin

$PS \leftarrow 0;$

While ($G \neq \phi$) {

For each node v_i

 // find local PS and candidate set

$in \leftarrow \text{slack_sum_fanin}(v_i);$

$out \leftarrow \text{slack_sum_fanout}(v_i);$

$self \leftarrow \text{slack}(v_i);$

$local_PS(v_i) \leftarrow \max \{ in, out, self \};$

$candidate(v_i) \leftarrow$ a set of nodes with local PS ;

end For

Find v_m such that $local_PS(v_m) = \max \{ local_PS(v_i) \};$

$PS \leftarrow PS + local_PS(v_m);$

Delete all (transitive) fanins/fanouts of

```
Nodes in  $candidate(v_m)$  from  $G$ ;  
}  
End  
}
```

2.2 Delay Budgeting Problem Formulation

In a typical CMOS standard cell library (e.g., TSMC 0.13 m), the cells with same logic function may have several layouts with specific information about power, area and delay. [3] targeted the capability of optimizing the power-delay tradeoff with different cell libraries, but did not consider the delay budgets which may provide further power optimization and thus promise a better tradeoff.

The delay-budgeting is to assign an additional delay (called budget) for each node so that a specific objective can be optimized while keeping non-negative slacks for all nodes. In general, a maximum sum of budgets over all nodes is the default objective, on which we have detailed ZSA, MISA and Greedy Algorithm. MISA produces an optimal solution while takes heaviest time complexity.

However, if the power reduction acts as such an objective, the problem turns out to be power-oriented delay-budgeting. For the power savings to be maximized, some gates need to be assigned more budgets than others, depending on both circuit topology and individual power-delay characteristics of the cells. Compounding the problem is the discreteness of the cell library which may make the budgets meaningless if they are not large enough.

In the following, we briefly discuss the power-delay curves of the nodes by presenting

four different models (the impact of graph topology on the delay-budgeting process will be investigated in the next section):

- Model (a): All nodes have uniform and linear power-delay characteristics. This is an ideal case where the potential power reduction is proportional to total delay budget, and the power-oriented delay-budgeting problem is equivalent to the traditional delay-budgeting.
- Model (b): All nodes' power-delay curves are linear (or piece-wise linear) but not uniform (i.e., they may have different slopes). This requires the delay to be weighted in the budgeting process. The nodes with steeper power-delay curves should be assigned a higher weight with the expectation of more power savings to be obtained.
- Model (c): The power-delay curves for all nodes are uniform but nonlinear. In this case, different nodes may have different weights depending on their current delay, and the nodes may also need to update their weights during the budgeting process to reflect the nonlinearity.
- Model (d): All nodes have different power-delay characteristics which are generally nonlinear. This represents a more practical case for real design, where not only do the nodes have to be weighted, but their weights are also expected to be updated during the delay assignment process. Finding an optimal or near-optimal solution to this case is a non-trivial task and requires computationally efficient algorithms.

2.3 Power-Oriented Delay Budgeting Algorithm

In a standard library, the number of different cells for a logic gate is quite limited, and not all obtainable delay budgets can be used for power minimization. To deal with the

nonlinearity of power-delay curves and the discreteness of target library as discussed above, we define a local potential power savings with delay budget (LPPDB) for each node as a weighted delay budget. The weight is set to zero if the budget is less than the delay of next available gate. Otherwise, it is set to the slope of the node's power-delay tradeoff curve which can be obtained using the following power and delay models:

$$\begin{cases} d(v) \propto fanout_area(v) / area(v) \\ p(v) \propto switching(v) \times (fanout_area(v) + area(v)) \end{cases} \quad (2.3)$$

where $area(v)$ and $fanout_area(v)$ are the size of the gate corresponding to node v and the sum of sizes of all its immediate fanouts, respectively, and $switching(v)$ is the switching activity for node v , available from gate-level description. All possible pairs of (p, d) from (2.3) form the power-delay curve of node v .

We define the potential power savings with delay budget (PPDB) for node v to be the maximum among the LPPDB of node v , the sum of v 's immediate fanins' LPPDBs and the sum of v 's immediate fanouts' LPPDBs. PPDB can be used as a new criterion for the power-oriented delay budgeting. The detailed pseudo-code of the proposed greedy algorithm is given as follows:

Power-Oriented Delay Budget Greedy Algorithm {

Input: Graph $G = (V, E)$ and a given library

Output: Power Savings (PS)

Begin {

PS \leftarrow 0;

Do {

for each node v

 Find PPDB and a candidate set:

```

    in ← sum of LPPDBs for all immediate fanins of v
    out ← sum of LPPDBs for all immediate fanouts of v
    self ← LPPDB for v
    PPDB(v) ← max{in, out, self};
    candidate(v) ← a set of nodes with PPDB;
end for
Find vm such that PPDB(vm) = max{PPDB(v) | v ∈ V};
PS ← PS + PPDB(vm);
Update slacks of all (transitive) fanins/ fanouts of nodes in candidate(vm) from G;
}
While (max{PPDB(v) | v ∈ V} > 0)
}
}

```

In what follows, we demonstrate how the greedy algorithm works on an example graph shown in Fig.2.4, using the library information given in Table.2.1. In Fig.2.4, $a(v)$, $r(v)$, $s(v)$, and $d(v)$ represent the arrival time, required time, slack, and delay of node v , respectively. Assuming a minimum delay of each node is used to start with, the algorithm first identifies node v_2 that has a maximum PPDB = 8, and its candidate set which is $candidate(v_2) = \{v_3, v_4, v_5\}$ (i.e., the immediate fanouts of v_2). Then, the algorithm brings PS from 0 to 8, and updates slacks of (transitive) fanins/ fanouts of $candidate(v_2)$ (i.e., v_1, v_2, v_6, v_7 and v_8). During the second and third iterations of the algorithm, the PS is updated to $(8 + 4) = 12$ and $(12 + 2) = 14$, respectively. After that, $\max\{PPDB\}$ turns out to be 0, which breaks the algorithm out of the While loop. The details of each iterative step are shown in Table.2.2. In this example, the original power dissipation is: $(4 + 3 + 4 + 6 + 4 + 6 + 10 + 6) = 43$ units. After the power-oriented delay budgeting, the PS of 14 units represents $14/43 \approx 33\%$ power savings.

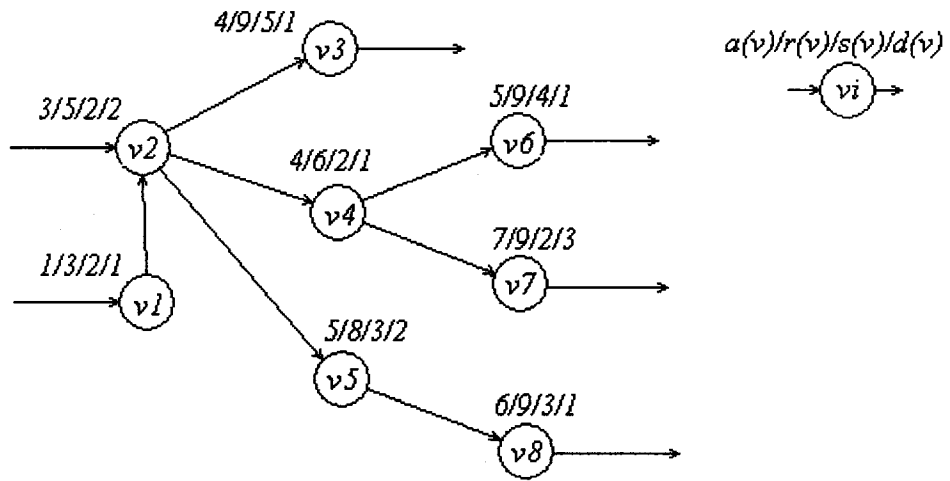


Figure.2.4. An eight-node subcircuit example

Table.2.1. The available library information for example graph of Fig.2.4

node	available layouts in a library (delay units power units)		
	1 4	2 2	4 1
v1	1 4	2 2	4 1
v2	2 3	3 2	6 1
v3	1 4	2 2	4 1
v4	1 6	2 3	6 1
v5	2 4	4 2	8 1
v6	1 6	2 3	3 2
v7	3 10	6 5	10 3
v8	1 6	3 2	6 1

Note that in the greedy algorithm, a large slack does not represent high delay budget, nor

does a high delay budget necessarily provide big potential power savings. In the above example, if we choose {v3, v4, v8} instead of {v3, v4, v5} to assign the delay budgets in the first iteration, the final PS turns out to be 16 units. In general, to guarantee a maximum PPDB, an optimal algorithm is needed which can recursively check PPDBs of all transitive fanins and fanouts for each node, leading to a prohibitively expensive computation cost. The above algorithm is greedy but much faster, making it practical for large circuits.

Table.2.2. Demonstration of the Greedy Algorithm on Fig.2.4

<i>iteration 1</i>	<i>v1</i>	<i>v2</i>	<i>v3</i>	<i>v4</i>	<i>v5</i>	<i>v6</i>	<i>v7</i>	<i>v8</i>
arrival time	1	3	4	4	5	5	7	6
required time	3	5	9	6	8	9	9	9
slack	2	2	5	2	3	4	2	3
delay	1	2	1	1	1	1	3	1
local_PPDB	2	8	3	4	4	4	3	4
candidate(s)	<i>v1</i>	<i>v3, v4, v5</i>	<i>v3</i>	<i>v6, v7</i>	<i>v8</i>	<i>v6</i>	<i>v4</i>	<i>v8</i>

<i>iteration 2</i>	<i>v1</i>	<i>v2</i>	<i>v3</i>	<i>v4</i>	<i>v5</i>	<i>v6</i>	<i>v7</i>	<i>v8</i>
arrival time	1	3	7	5	7	6	8	8
required time	2	4	9	6	8	9	9	9
slack	1	1	2	1	1	3	1	1
delay	1	2	4	2	4	1	3	1
local_PPDB	2	2	1	4	1	4	0	0
candidate(s)	<i>v1</i>	<i>v1</i>	<i>v2</i>	<i>v6, v7</i>	<i>v2</i>	<i>v6</i>	<i>v7</i>	<i>v8</i>

<i>iteration 3</i>	<i>v1</i>	<i>v2</i>	<i>v3</i>	<i>v4</i>	<i>v5</i>	<i>v6</i>	<i>v7</i>	<i>v8</i>
arrival time	1	3	7	5	7	8	8	8
required time	2	4	9	6	8	9	9	9
slack	1	1	2	1	1	1	1	1
delay	1	2	4	2	4	3	3	1
local_PPDB	2	2	1	1	1	0	0	0
candidate(s)	<i>v1</i>	<i>v1</i>	<i>v2</i>	<i>v2</i>	<i>v2</i>	<i>v6</i>	<i>v7</i>	<i>v8</i>

<i>iteration 4</i>	<i>v1</i>	<i>v2</i>	<i>v3</i>	<i>v4</i>	<i>v5</i>	<i>v6</i>	<i>v7</i>	<i>v8</i>
arrival time	2	4	8	6	8	9	9	9
required time	2	4	9	6	8	9	9	9
slack	0	0	1	0	0	0	0	0
delay	2	2	4	2	4	3	3	1
local_PPDB	0	0	0	0	0	0	0	0
candidate(s)	<i>v1</i>	<i>v2</i>	<i>v3</i>	<i>v4</i>	<i>v5</i>	<i>v6</i>	<i>v7</i>	<i>v8</i>

2.4 Experiments on Benchmark Circuits

We implemented the proposed greedy algorithm in Java programming language and tested it on benchmark circuits on top of SIS package [9]. The synthesis results with minimum delay from SIS were used as an input to our delay-budgeting problem. We also compared the results with those from [2].

For each benchmark circuit, we first estimated its original circuit delay (T_0) and power consumption (P_0). Then, we ran our algorithm using T_0 as the timing constraints to show how much power savings can be achieved by assigning the delay budgets to the nodes on non-critical paths. The results represent the potential power savings of each benchmark circuit without any delay penalty. In order to get the optimized power-delay tradeoff, we relaxed the timing constraints from T_0 to $4T_0$ with a step of $0.1T_0$, and measured the power consumption P after the algorithm was performed. The results reflect the possible power savings with certain percentage of delay penalty.

Table.2.3 summarizes the power consumption with our algorithm for different timing constraints. It can be seen that (a) an average of 35% power reduction (with a maximum of 65%) is achieved without any delay penalty, and (b) the amount of power savings for a given delay penalty varies significantly from circuit to circuit, depending upon their topologic structures. Table2.4. shows the power consumption using the algorithm from [2] for the same benchmarks. We see that the proposed algorithm provides an improvement of about 5~10%, which is significant considering the fact that the percentage is taken with respect to initial power consumption (i.e., P_0).

Table.2.3. Power Consumption with the proposed algorithm on a set of benchmark circuits

circuit	size (#gates)	power consumption (% P_0)						
		T_0	$1.5T_0$	$2T_0$	$2.5T_0$	$3T_0$	$3.5T_0$	$4T_0$
<i>cif</i>	5	100.0%	64.4%	46.6%	46.6%	32.2%	32.2%	25.0%
<i>bi</i>	7	86.2%	51.2%	41.1%	37.9%	28.2%	28.2%	25.0%
<i>mux</i>	28	97.1%	49.8%	44.2%	39.7%	35.7%	27.9%	25.0%
<i>decod</i>	30	100.0%	71.1%	64.6%	58.6%	40.0%	30.2%	28.5%
<i>cm150a</i>	32	79.6%	47.5%	38.6%	33.5%	30.9%	26.9%	25.0%
<i>r4m1</i>	33	79.9%	63.8%	48.7%	41.0%	32.4%	28.7%	26.7%
<i>cc</i>	42	48.7%	38.9%	32.4%	28.6%	32.3%	28.9%	26.0%
<i>count</i>	96	43.1%	29.5%	27.2%	25.9%	25.6%	25.6%	25.0%
<i>b9</i>	103	51.9%	37.0%	31.3%	25.5%	25.3%	25.3%	25.0%
<i>apex7</i>	151	47.7%	33.8%	29.7%	27.8%	25.4%	25.2%	25.2%
<i>xi</i>	218	40.2%	32.7%	31.1%	26.8%	25.9%	25.3%	25.0%
<i>alu2</i>	277	62.9%	47.5%	37.0%	31.8%	30.3%	26.7%	25.9%
<i>apex6</i>	487	34.8%	29.1%	26.2%	25.5%	25.3%	25.3%	25.2%
<i>i8</i>	748	34.1%	29.3%	27.2%	25.8%	25.1%	25.0%	25.0%
average		64.7%	44.7%	37.6%	33.9%	29.6%	27.2%	25.5%

Table.2.4. Power consumption using the algorithm from [2]

circuit	size (#gates)	power consumption (% P_0)						
		T_0	$1.5T_0$	$2T_0$	$2.5T_0$	$3T_0$	$3.5T_0$	$4T_0$
<i>cif</i>	5	100.0%	64.4%	46.6%	46.6%	32.2%	78.4%	25.0%
<i>bi</i>	7	87.1%	51.2%	41.1%	37.9%	28.2%	44.0%	25.0%
<i>mux</i>	28	97.1%	49.8%	44.2%	68.1%	40.9%	27.9%	25.0%
<i>decod</i>	30	100.0%	72.8%	64.6%	57.8%	43.1%	30.2%	28.5%
<i>cm150a</i>	32	79.6%	49.0%	63.2%	64.2%	54.9%	27.6%	25.0%
<i>r4m1</i>	33	80.5%	77.4%	55.1%	51.4%	43.4%	33.0%	26.7%
<i>cc</i>	42	48.8%	41.3%	32.7%	29.9%	32.3%	28.9%	25.8%
<i>count</i>	96	45.4%	32.2%	27.8%	29.6%	28.6%	36.4%	25.3%
<i>b9</i>	103	54.3%	44.1%	32.6%	27.4%	25.3%	32.0%	25.0%
<i>apex7</i>	151	50.2%	44.6%	32.6%	28.4%	28.6%	28.5%	25.0%
<i>xi</i>	218	47.9%	39.4%	31.9%	33.1%	26.5%	25.6%	25.0%
<i>alu2</i>	277	65.5%	57.6%	51.5%	44.7%	39.5%	31.0%	26.0%
<i>apex6</i>	487	37.5%	31.9%	26.7%	25.8%	25.7%	25.6%	25.5%
<i>i8</i>	748	36.1%	34.9%	30.5%	32.2%	28.7%	26.2%	25.0%
average		66.4%	49.3%	41.5%	41.2%	34.1%	34.0%	25.6%

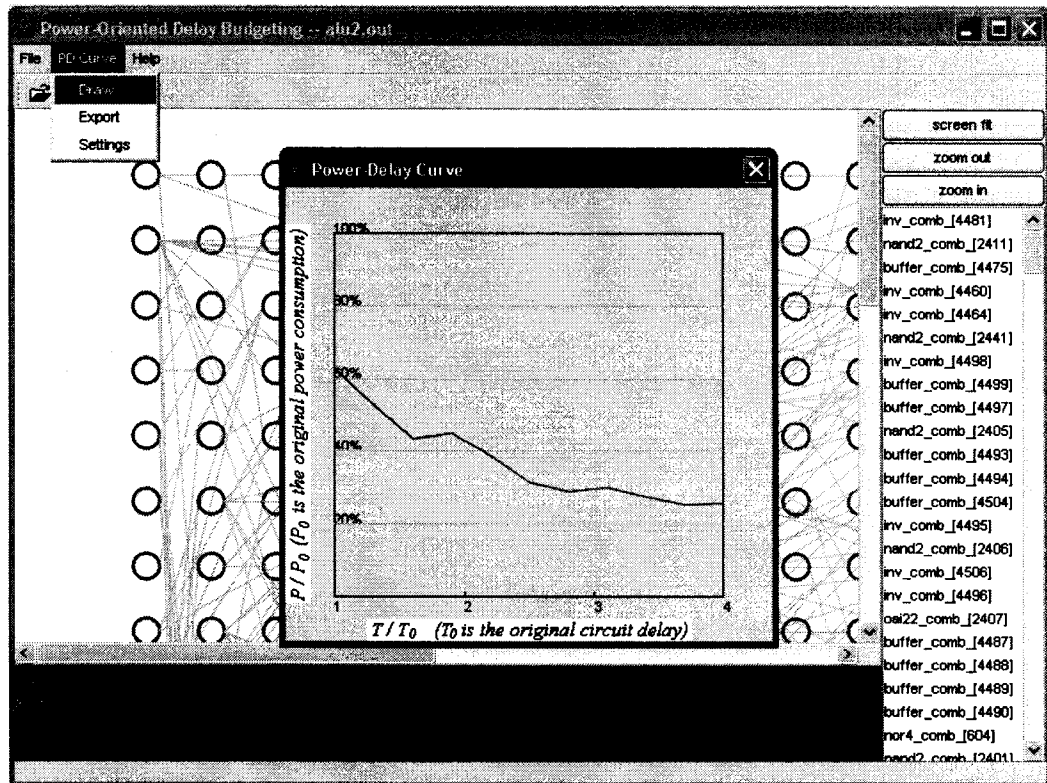
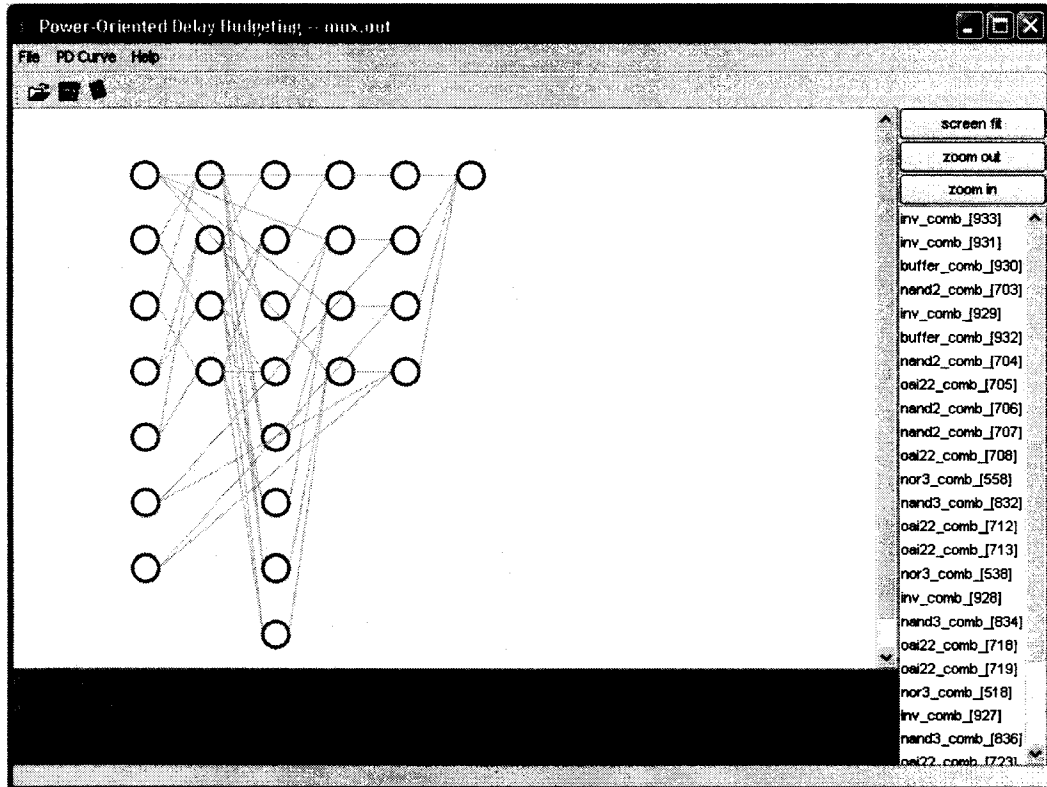


Fig.2.5. Java Implementation of power-oriented delay budgeting greedy algorithm

2.5 Summary

We have proposed a greedy algorithm to solve the power-oriented delay budgeting problem. We have shown that the topologic information of the circuits as well as cell library can be used for aggressive power optimization in order to obtain good and effective power-delay tradeoff. Experimental comparison has also been made between a pure delay-budgeting algorithm and the proposed algorithm in terms of power savings.

Part of this work was published in our recent paper titled “Power-Oriented Delay Budgeting for Combinational Circuits” which is to appear in the Proceedings of 2006 ISCAS (International Symposium on Circuits and Systems).

Chapter 3

Single Electron Tunneling (SET) Threshold Logic

As MOS transistors are expected to finally reach their physical fabrication limits for further power and/or area reduction, various new devices and architectures at nano-scale have recently been proposed, such as single electron transistors (SETs), resonant tunneling diodes (RTDs), and quantum dots and quantum cellular automata (QCA).

From this chapter, we will focus on SET devices and logic which have been shown to be a promising candidate for future ultra-low power integrated circuits.

3.1 Introduction to Single-Electron Tunneling

Single-electron devices are such devices functional based on the controllable transfer of single electrons between small conductors called 'islands'. The main advantages of single-electron devices are their fast and ultra-low power operation. While the prospect of CMOS components being completely replaced by SET counterparts remains to be seen,

the SET logic circuits receive increasing attention in the research community.

3.1.1 Theoretical Background

For single-electron devices, a tunnel event is defined as the transport of a single electron through the tunnel junction. The concept of single-electron control can be explained by the Coulomb Blockade theory. After a tunnel event, the net charge Q of the island is $-e$ (1.6×10^{-19} Coulomb), and the resulting electric field repulses the following electrons which might be added.

A tunneling event is a random event with a probability formulated by an ‘orthodox’ theory:

$$\Gamma(\Delta W) = (1/e)I(\Delta W/e)[1 - \exp\{-\Delta W/k_B T\}]^{-1} \quad (3.1)$$

where $I(V)$ is the dc I - V curve of the tunnel barrier in the absence of single-electron charging effects. $\Delta W = e(V_i + V_j)/2$, where V_i and V_j are voltage drops across the barrier before and after the tunneling event, respectively.

At a very low temperature, the voltage across a tunnel junction, V_j , is the only factor to determine the possibility of a tunnel event. The critical voltage V_c of a tunnel junction is given by [22]:

$$V_c = \frac{q_e}{2(C_e + C_j)} \quad (3.2)$$

where C_j is the capacitance of the tunnel junction, C_e the equivalent capacitance viewed from the tunnel junction’s perspective, and q_e the charge of an electron ($1.602 \times 10^{-19} C$). A tunnel event will occur through a junction only if the absolute value of V_j is larger than V_c , i.e., $|V_j| > V_c$. Single-electron devices utilize the SET tunnel junction’s ability to control

the transport of individual electrons, and provide an alternative to implement digital logic. An electron tunneling through a junction in the quantum scale is a stochastic process. If we define P_{error} as the probability that the desired transport does not occur, the switching delay t_d can be expressed as:

$$t_d = \frac{-\ln(P_{error})q_e R_t}{|V_j| - V_c} \quad (3.3)$$

where R_t is the tunnel resistance.

3.1.2 Fabrication Limitation

In order to implement SET devices, it requires a reproducible fabrication of very small conducting particles, and their accurate positioning with respect to external electrodes. Current nanofabrication technologies have made possible small SET devices. However, the time for fabricating large SET circuits is still expected to be 40 years away. Most current research on SET circuit level designs, including ours, cannot be verified by current nanofabrication technologies. Some simulators have been developed to support circuit level design. To do simulation, we use SIMON (SIMulation Of Nanostructures) [27].

3.2 SET Threshold Gate Structure

A generic SET threshold logic structure has been reported [22, 29, 30], and is depicted in Fig.3.1. In this threshold logic structure, the critical voltage V_c acts as the ‘threshold’. If the voltage across the tunnel junction is higher than the critical voltage, one electron tunnels through the junction. This can be specified as a ‘high’ output. Otherwise, the output is meant to be logic ‘low’. Input voltages V_p and V_n , weighted by their input capacitors C^p and C^n respectively, are added across the junction (the superscripts p

and n stand for positively and negatively weighted inputs, respectively). V_b is the bias voltage weighted by its input capacitance C_b .

In order to provide enough driving ability and stability, a buffer/inverter, as shown in Fig.3.2, should follow the threshold logic structure. Hereafter, we define a threshold gate as a threshold logic structure plus a buffer/inverter following it. Since a buffer/inverter inverts the output of the original threshold logic, one needs to reverse the positively- and negatively-weighted inputs in the logic function accordingly.

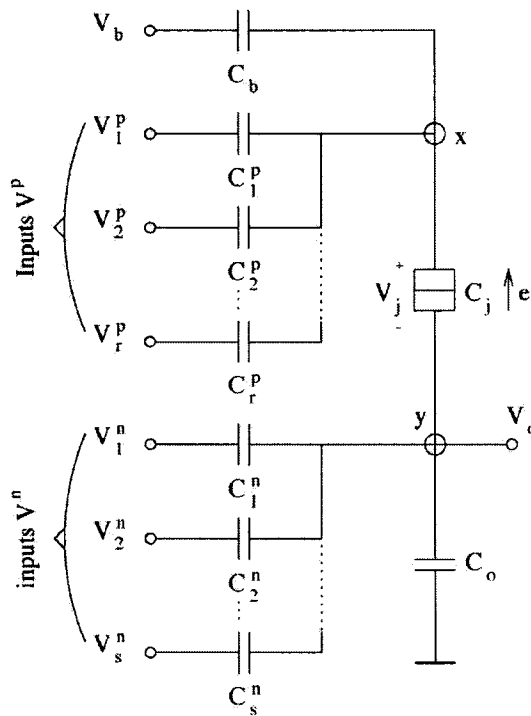


Fig.3.1. SET threshold logic structure

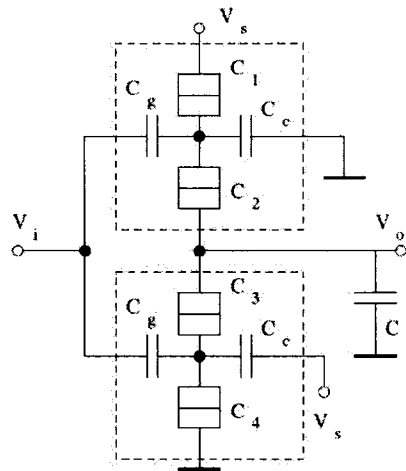


Fig.3.2. SET buffer/inverter

3.3 Feasible Parameters of SET Threshold Logic

It has been shown [22, 29] that when weighted properly, a SET threshold gate can implement different combinational logic (such as AND, NOR) gates. More specifically, one can choose all parameters in the logic structure carefully to obtain a meaningful logic function. A group of parameters that promise a specific logic function are called feasible parameters for the threshold logic. By using a number of threshold gates with feasible parameters, more complex logic components (such as flip-flops) could also be implemented.

The logic function for a variety of gates can be represented by a threshold expression. The threshold expressions for some commonly-used 2-input logic gates are shown in Table.3.1, where a and b are inputs.

Table.3.1. Threshold expressions

Gate	Threshold expression
AND(a, b)	$\text{sgn}\{a + b - 1.5\}$
OR(a, b)	$\text{sgn}\{a + b - 0.5\}$
NAND(a, b)	$\text{sgn}\{-a - b + 1.5\}$
NOR(a, b)	$\text{sgn}\{-a - b + 0.5\}$

The capacitance parameters of a threshold gate can be determined by the relationship between the tunnel critical voltage, V_c , and the voltage across the tunnel, V_j . The critical voltage can be written as

$$V_c = \frac{q_e}{2\left\{C_j + \frac{(\sum C^n + C_o + C_{buffer})(C_b + \sum C^p)}{\sum C^n + C_o + C_{buffer} + C_b + \sum C^p}\right\}} \quad (3.4)$$

where C_j is the tunnel capacitance, and C_{buffer} is the equivalent capacitance of buffer/inverter viewed from the output of a generic threshold structure. If we define V_{cc} as the power supply voltage shared by all V_p , V_n and V_b , the voltage across the junction can be expressed as

$$V_j = V_{cc} \left\{ \frac{C_b + \sum C^p(1)}{C_b + \sum C^p} - \frac{\sum C^n(1)}{\sum C^n + C_o + C_{buffer}} \right\} - V_{buffer} \quad (3.5)$$

where V_{buffer} is the voltage drop back on the output of the generic threshold structure, caused by the power supply of the buffer/inverter, and $\sum C^p(1)$ (or $\sum C^n(1)$) represents the sum of C^p 's (or C^n 's) that are connected to a high input voltage (i.e., logic value of "1"). Since both V_c and V_j are a function of all capacitance parameters (i.e., C^p , C^n , C_b and C_o), the feasible parameters for a given threshold expression are not unique.

As an example, if we choose $V_{cc} = 16\text{mV}$ and use the parameter values of the buffer

given in [22] (as shown in Fig.3.3), we can get $C_{buffer} = 0.9$ aF, and $V_{buffer} = 1.17$ mV. By choosing C^p and C^n to be 0.5 aF, and fixing the total loading capacitance of the threshold logic, $\sum C^n + C_o + C_{buffer}$, to be 10 aF, we leave C_b to be the only parameter to be determined, with its value depending upon the specific threshold logic expression.

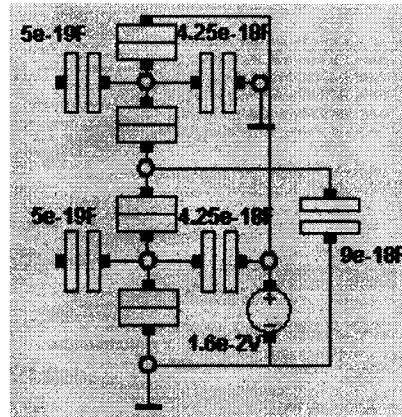


Fig.3.3. The selected capacitance values of buffer/inverter from [22]

3.4 Capacitance Parameter Selection

The general procedure of selecting capacitance parameters in the threshold logic consists of three steps as follows:

Step 1: Determine the structure of threshold gate with positively weighted inputs connected to C^n 's and negatively-weighted inputs connected to C^p 's. Take a 2-input AND gate as an example with the threshold expression $y = \text{sgn}\{a + b - 1.5\}$, where y is the output, and a and b are the inputs which are connected to C_1^n and C_2^n , respectively, with the same value of 0.5 aF for each. Given the total loading capacitance $\sum C^n + C_o + C_{buffer} = 10$ aF, we select C_o to be 8.1 aF.

Step 2: Find two input patterns with the combined value closest to the threshold (one of them gives a value higher than the threshold, the other lower than the threshold). For the case of 2-input AND gate with four different input patterns $\{a, b\} = \{0, 0\}, \{0, 1\}, \{1, 0\}$, or $\{1, 1\}$, the combined value $(a + b)$ (refer to the threshold expression $y = \text{sgn}\{a + b - 1.5\}$) is 0, 1, 1, or 2, respectively, and the threshold value is 1.5. Therefore, we select two input logic patterns $\{1, 1\}$ and $\{1, 0\}$ (or $\{0, 1\}$).

Step 3: Solve two inequalities, $V_j > V_c$ and $V_j < V_c$, for the above two input patterns to get the upper and lower bounds of acceptable value of C_b . A tunnel event happens if and only if $V_j > V_c$. In the 2-input AND gate example, we require $V_j > V_c$ for the input pattern $\{1, 0\}$, and $V_j < V_c$ for the input pattern $\{1, 1\}$. This gives the value range of C_b : 12.8 aF \sim 14.6 aF. The results have been verified with SIMON [27].

The above procedure applies to various threshold logic gates. Table.3.2 shows the capacitance parameters for different gates given in Table.3.1.

Table.3.2. Capacitance parameters for different gates given in Table.3.1

Gate	C^p (aF)		C^n (aF)		C_o (aF)	C_b (aF)
AND(a,b)			0.5	0.5	8.1	(12.8,14.6)
OR(a,b)			0.5	0.5	8.1	(11.3,12.7)
NAND(a,b)	0.5	0.5			9.1	(10.3,11.4)
NOR(a,b)	0.5	0.5			9.1	(11.5,12.5)

3.5 Parameter Optimization for Reliability

It is observed that during the fabrication process, certain amount of charges could appear on nodes of a SET logic gate. These charges are referred to as background charges. Research shows that background charge is random and less likely to be eliminated with current technologies. Background charges generate a biased voltage which contributes to the total voltage across a SET logic gate and may affect the logic behavior.

Let q_x and q_y be the background charge on nodes x and y , respectively, in Fig.3.1 which is to be followed by a SET buffer of Fig.3.2. The bias voltage (denoted by $V_{background}$) across the junction due to q_x and q_y is:

$$V_{background} = \frac{q_x}{C_b + \sum C^p} - \frac{q_y}{\sum C^n + C_o + C_{buffer}} \quad (3.6)$$

This extra voltage will add to the original voltage drop V_j across the junction, yielding a new junction voltage, V_j' :

$$\begin{aligned} V_j' &= V_j + V_{background} \\ &= V_{cc} \left\{ \frac{C_b + \sum C^p(1)}{C_b + \sum C^p} - \frac{\sum C^n(1)}{\sum C^n + C_o + C_{buffer}} \right\} - V_{buffer} \\ &\quad + \frac{q_x}{C_b + \sum C^p} - \frac{q_y}{\sum C^n + C_o + C_{buffer}} \end{aligned} \quad (3.7)$$

If a given set of feasible parameters for a SET threshold gate with V_j are still feasible for the gate with V_j' , the logic behavior of the gate remains correct even with the background charges. Otherwise, the circuit will malfunction. In this paper, the reliability of a SET threshold gate is defined to be the probability that the gate still performs the desired logic, with possible background charges.

From Table.3.2 with assumption of no background charges, C_b can be any value between

its upper bound and lower bound for the gate to function correctly. The question is: Given a probability distribution of background charge, what is the optimal value of Cb so that the reliability of the underlying gates can be maximized?

From equation (3.5), we can obtain the probability distribution $p(V_{background})$ for the extra voltage $V_{background}$. Depending on the structure and parameters of a threshold gate, a maximum value of $V_{background}$ (denoted by $\max V_{background}$) and a minimum value of $V_{background}$ (denoted by $\min V_{background}$) can be found from equation (3.6) in order to keep the desirable logic. The problem is to determine the value of Cb , which maximizes the reliability: $P(V_{background}) = \int_{\min V_{background}}^{\max V_{background}} p(V_{background}) dV_{background}$.

In the reliability analysis that follows, we take 2-input AND gate and NOR for example to show how the reliability of a SET threshold logic gate varies with the parameter C_b .

Throughout our discussions, two assumptions are also made:

- 1) Background charges on different nodes are independent;
- 2) Background charges take either uniform or normal probability distribution.

3.5.1 Reliability Analysis with Background Charges of Uniform Probability Distribution

In the threshold logic gates (see Fig.3.1), we assume both nodes x and y have background charges (i.e., q_x and q_y) of a uniform probability distribution:

$$p(q) = \begin{cases} \frac{1}{2\mu}, & -\mu \leq q \leq \mu \\ 0, & \text{otherwise} \end{cases} \quad (3.8)$$

Let $z = V_{background}$. The equation (3.5) can be rewritten as:

$$z = \frac{q_x}{m} - \frac{q_y}{n} \quad (3.9)$$

where

$$m = C_b + \sum C^p$$

$$n = \sum C^n + C_o + C_{buffer}$$

The distribution for z is:

$$p(z) = \begin{cases} \frac{mn}{\mu(m+n)} \left(1 + \frac{mn}{\mu(m+n)} z\right), & -\frac{\mu(m+n)}{mn} \leq z < 0 \\ \frac{mn}{\mu(m+n)} \left(1 - \frac{mn}{\mu(m+n)} z\right), & 0 \leq z \leq \frac{\mu(m+n)}{mn} \\ 0, & \text{otherwise} \end{cases} \quad (3.10)$$

By choosing the different values for C_b in a particular gate, we can get the different reliabilities: $P(z) = \int p(z) dz$. Table.3.3 and Table.3.4 summarize the reliabilities for 2-input AND and NOR gates, respectively, with uniform distribution of background charges. It can be seen from these two tables that with the smaller value of μ , the reliability is more sensitive to the value of C_b , meaning that the optimized value of C_b can significantly improve the reliability. We plot the reliability-versus- μ curves for 2-input AND and NOR gates in Fig.3.4 and Fig.3.5, respectively. It can be seen that as the value of μ increases, the reliability decreases quickly and the value of C_b becomes less critical.

**Table.3.3. Reliabilities of 2-input AND gate with
uniform distribution of background charges**

μ / q_e	Optimized		Worst Case	
	Cb (aF)	Reliability (%)	Cb (aF)	Reliability (%)
0.5	14.6	5.8	12.8	5.5
0.4	14.4	7.2	12.8	6.8
0.3	14.2	9.5	12.8	8.9
0.2	14.0	14.0	12.8	13.0
0.1	13.8	26.8	12.8	24.2
0.05	13.7	49.4	12.8	40.6
0.03	13.7	73.1	12.8	50.4
0.02	13.7	92.2	12.8	51.0

**Table.3.4. Reliabilities of 2-input NOR gate with
uniform distribution of background charges**

μ / q_e	Optimized		Worst Case	
	Cb (aF)	Reliability (%)	Cb (aF)	Reliability (%)
0.5	11.5	4.4	12.5	4.2
0.4	11.5	5.4	12.5	5.2
0.3	11.6	7.2	12.5	6.8
0.2	11.7	10.6	12.5	10.1
0.1	11.9	20.6	12.5	19.0
0.05	12.0	38.8	12.5	33.6
0.03	12.0	59.4	11.5	45.8
0.02	12.0	79.2	11.5	48.2

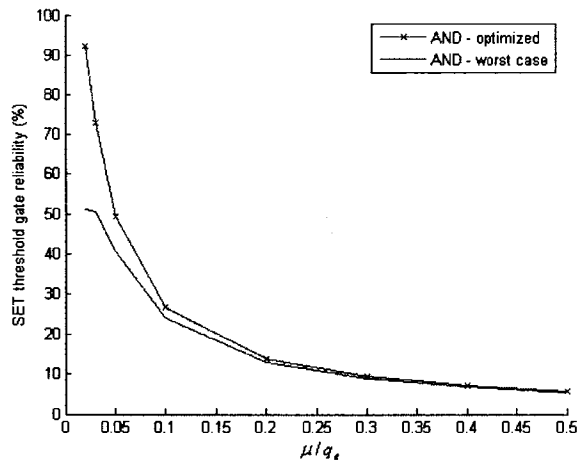


Fig.3.4. Reliability of a 2- input AND gate with uniform distribution of background charges

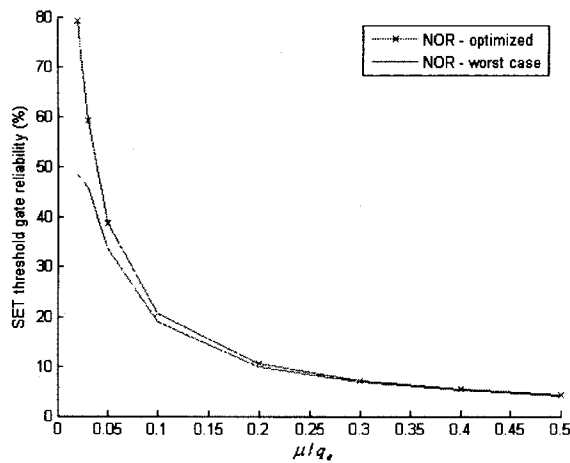


Fig.3.5. Reliability of a 2- input NOR gate with uniform distribution of background charges

3.5.2 Reliability Analysis with Background Charges of Normal Probability Distribution

If the random background charges q_x and q_y take a normal distribution with a standard deviation σ :

$$p(q) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{q^2}{2\sigma^2}} \quad (3.11)$$

it can be proved that the distribution of z is also normal, but with a standard deviation of $\frac{\sqrt{m^2 + n^2}\sigma}{mn}$. By using similar analysis for the reliability with uniform distribution,

we can calculate the reliabilities of any logic gates with normal distribution of random background charges. Again, we take the 2-input AND and NOR gates as an example, and the results are shown in Table.3.5 and Table.3.6, where the parameters of Cb and σ are the key factors in determining the reliability. The reliability-versus- σ curves are shown in Fig.3.6 and Fig.3.7. It can be observed from these tables and figures that with a small value of σ , the Cb needs to be chosen carefully in order to maximize the reliability.

Table.3.5. Reliabilities of 2-input AND gate with normal distribution of background charges

σ / q_e	Optimized		Worst Case	
	Cb (aF)	Reliability (%)	Cb (aF)	Reliability (%)
0.5	14.6	3.3	12.8	3.2
0.4	14.6	4.1	12.8	3.9
0.3	14.6	5.5	12.8	5.2
0.2	14.6	8.2	12.8	7.8
0.1	14.2	16.2	12.8	15.4
0.05	13.8	31.5	12.8	28.6
0.03	13.7	50.1	12.8	40.9
0.02	13.7	68.9	12.8	48.3
0.01	13.7	95.7	12.8	51.5

Table.3.6. Reliabilities of 2-input NOR gate with normal distribution of background charges

σ / q_e	Optimized		Worst Case	
	Cb (aF)	Reliability (%)	Cb (aF)	Reliability (%)
0.5	11.5	2.5	12.5	2.4
0.4	11.5	3.1	12.5	3.0
0.3	11.5	4.2	12.5	4.0
0.2	11.5	6.3	12.5	5.9
0.1	11.5	12.4	12.5	11.7
0.05	11.8	24.2	12.5	22.5
0.03	11.9	39.1	12.5	34.1
0.02	12.0	55.7	12.5	43.6
0.01	12.0	87.5	11.5	48.2

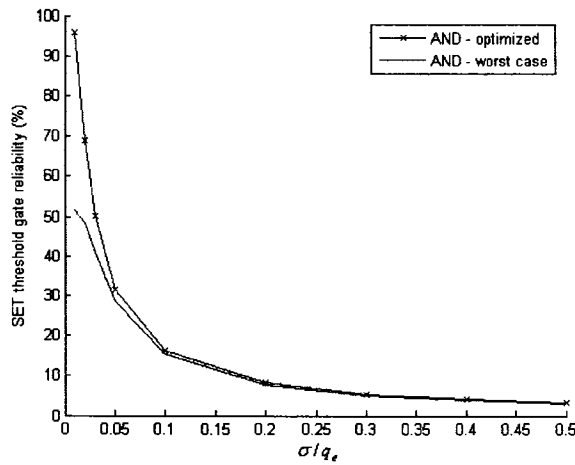


Fig.3.6. Reliability of a 2- input AND gate with normal distribution of background charges

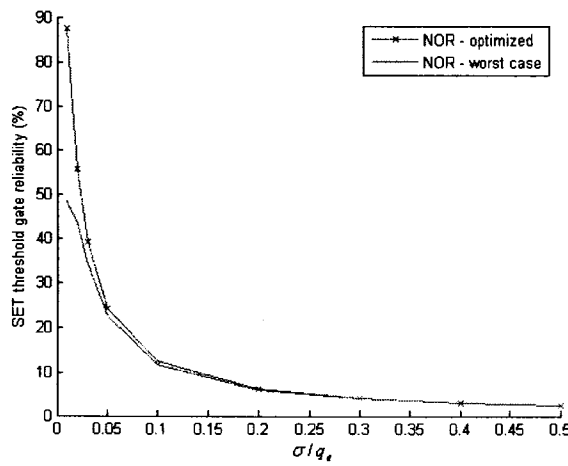
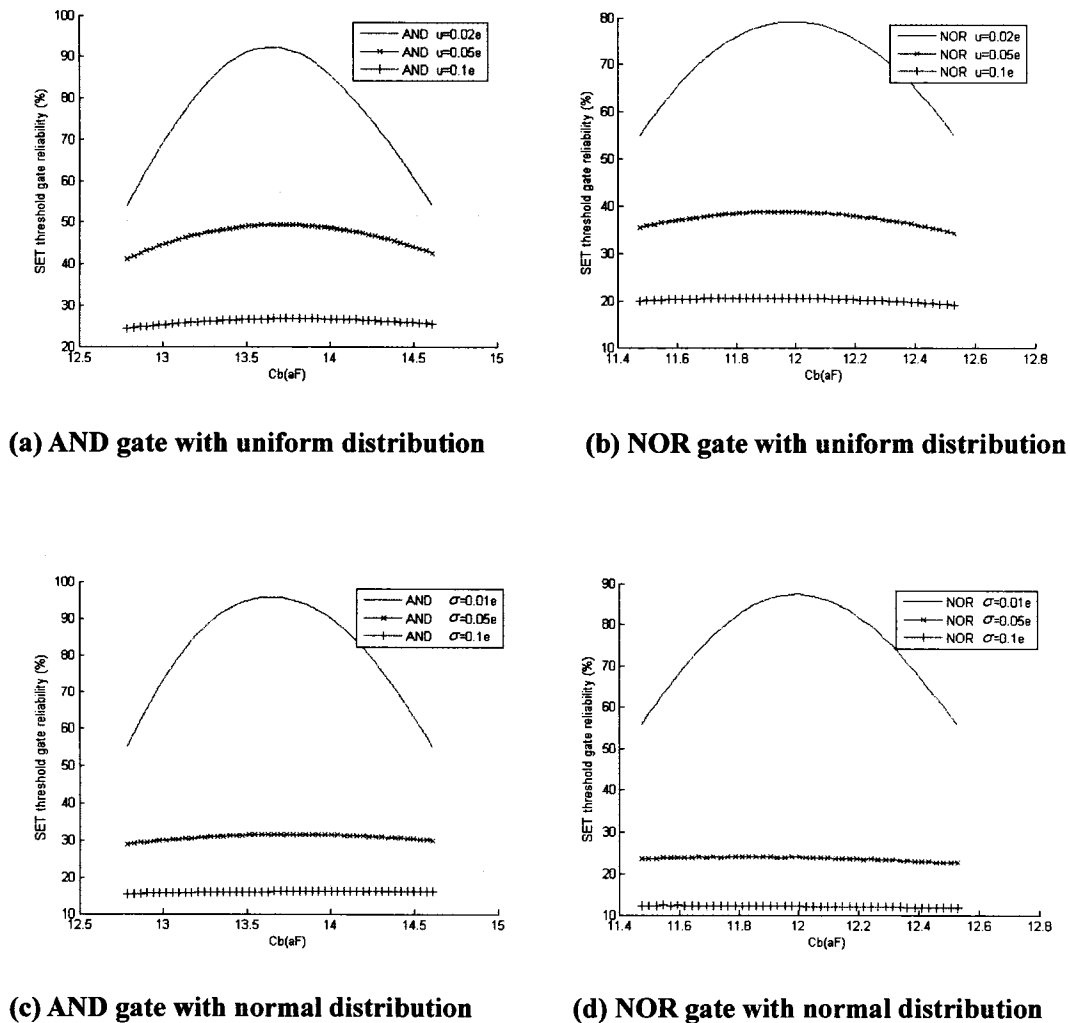


Fig.3.7. Reliability of a 2- input NOR gate with uniform distribution of background charges

3.5.3 Relationship between Gate Reliability and Circuit Parameters

The above analysis shows that if the probability distribution curves are flat (i.e., with a large value of μ and σ for uniform and normal distribution, respectively), the gate

reliability turns out to be low. Under such cases, the parameter optimization does little to improve the reliability. Instead, as the value of μ or σ gets smaller, the gate reliability strongly depends on Cb . To demonstrate this sensitivity, we plot the reliability-versus- Cb curves in Fig.3.8. As a rule of thumb, the value of Cb should be somewhere in the middle between its upper bound and lower bound in order to maximize the reliability, given either uniform or normal distribution of background charges.



**Fig.3.8. Reliability-versus- Cb curves
with different probability distribution of background charges**

3.5.4 Parameter Scaling

Theoretically, all parameters in threshold logic gates can be scaled while keeping the same logic behavior. From equation (3.2), if we increase (or decrease) all the capacitance values by X times, and reduce (or raise) the power supply voltage by X times, the threshold gate will have an output voltage level which is reduced (or increased) by X times after scaling, with the logic behavior unchanged. However, too large or too small values may not be possible in practice with current technologies. Too large value (say, larger than 100 aF) of capacitances requires the circuit to operate at extremely low temperature (say, 1~10 K) in order to avoid thermally induced random tunneling events. On the other hand, too small value (say, less than 0.1 aF) of capacitances requires the device size to be extremely small (say, 1~5 nm). This poses a great challenge to, and may not be likely with, current nano-fabrication technology. In this paper, we set a minimum value of capacitances (such as C^p and C^n) to be 0.5 aF, and the largest capacitance in the gates is Cb which is within the range of 10~15 aF.

Delay and energy consumption are another issue which may have to be considered during the parameter selection or optimization. With X -time scaling, the delay of a threshold gate will increase by X times, as can be seen from equation (3.2). Also, the energy consumed by a single tunnel event is $\Delta E = q_e (|V_j| - V_c)$, which will be reduced by X times after scaling. However, the reliability of the threshold gate remains the same, given a certain probability distribution of the random background charges.

3.6 Summary

In this chapter, we propose a strategy to select parameters and analyze the reliability for

single-electron threshold logic with emphasis on two typical logic gates (AND and NOR). The effect of possible background charges has been taken into account for the reliability improvement. While a uniform or normal distribution of random background charges is assumed throughout the paper, the proposed method can apply to any other distributions. Further work is also needed to deal with the correlated distribution of background charges, which could occur in the real world.

Chapter 4

A Finite State Machine (FSM) Implementation with SET Threshold Gates

Finite state machine (FSM) is well understood as a typical module for most sequential circuit designs [16]. It can be described by a state transition graph, where each vertex represents a state and each directed edge represents the transition from one state to another. Implementation of an FSM involves (a) the logic design which includes the state assignment followed by combinational logic synthesis, and (b) the circuit realization with specific technology. Due to the increasing complexity and power dissipation of digital systems, a lot of research works have focused on FSM synthesis towards low power requirement [17-19]. As MOS transistors are expected to finally meet their physical fabrication limits in further power and/or area reduction, various new revolutionary device architectures have been recently proposed at nano-scale [21], with an increasing interest in Single Electron Tunneling (SET) devices as promising candidates for future ultra-low power integrated circuits [20-22].

Single-electron devices utilize one-electron-precision charge transfer based on the Coulomb blockade effect, and provide an alternative to implement digital logic. Single-electron transistor and single-electron memory are such an example [20, 22, 23]. The main advantages of single-electron devices are their fast and ultra-low power operation, simply because they use only a few electrons to accomplish logic or arithmetic operations. While the prospect of CMOS components being completely replaced by SET remains to be seen, more and more SET circuits are emerging.

In this chapter, we present an FSM implementation using SET technology, and apply our approach to radio-frequency identification (RFID) protocols where the power dissipation is a critical concern. We simulate the proposed circuit and provide its power estimation to show its particular advantages over its CMOS counterpart.

4.1 Radio Frequency Identification (RFID)

Radio Frequency Identification systems consist of electromagnetic readers and radio frequency (RF) tags. The reader communicates with tags within its read range and tries to obtain unique IDs of each tag.

Predictions have been on business journals and magazines for long time that RFID systems will sooner or later replace the current heavily-used barcode systems. By thoroughly changing the mode of logistics, manufacture and retailing, RFID will have a positive impact on the worldwide economy and change our every day life.

However, before that time comes, it is also predicted that the cost of one tag must be reduced to less than 1 cent in order to initialize a considerable market volume. Also, the size of a tag is required to be very small to fit anywhere. To do this, the circuit function on a tag should be made as simple as possible.

Passive mode naturally comes into designers' mind. A passive mode circuit is a circuit without battery power supply itself. A passive mode tag receives power from the reader and work as a passive response. Non-battery seems to provide an overall solution to both size and cost. In addition, problems like battery-life will not exist if there is no battery on tags and make an RFID system almost maintenance free.

In order to get all those profits a passive tag may bring, however, power consumption is an unpreventable obstacle in the way. The electromagnetic power a tag receives from a reader is given by:

$$S = \frac{\lambda^2 P G_s G_r}{(4\pi R)^2} \quad (4.1)$$

where λ is the wavelength of the emitted electro-magnetic wave, P the power injected into the reader's antenna, R the distance between the reader and the tag, G_s the gain of the reader's antenna, and G_r the gain of the tag's antenna.

For a frequency of 900MHz, the working distance of 1 meter makes tags fall into the far field region of the reader's antenna. The energy received by the tag is less than $100 \mu W$. Furthermore, if we need an RFID system suitable for real application, warehouse management for example, a working distance of more than 10 meters should be expected. In that way, the energy received by the tag will be less than $0.1 \mu W$.

4.2 RFID Anti-Collision Protocols

For the power received by the tags from the electro-magnetic waves, consumption can be considered into two parts. One part supplies for sending signals from the tag's antenna

back to the reader. Another is for logic calculating for what data it should send back based on the signals received from the reader.

For a typical RFID system, a reader should have the ability to identify all tags within its working zone. A collision problem may arise when there are more than one tags sending series of data in response to the same reader's inquiry. Without an anti-collision mechanism, the reader will receive a mixture of scattered data and cannot work properly to identify individual IDs of tags. The digital logic circuit implemented on a tag is basically an anti-collision scheme. For anti-collision protocols, there are two major schemes right now—Binary-Tree Scheme and Query-Tree Scheme.

4.2.1 Binary –Tree Scheme

In a binary-tree protocol, there is a pointer with each tag. Each time a tag is reset, the pointer points to the first bit of the tag's ID and, with the ongoing of inquiring, it moves toward the last bit. During the inquiring, the reader sends one ID bit at a time. The tags with the pointed bit the same as the inquiring bit send back their next bits to the reader, while the rest convert to a state of 'quiet' and will not answer the remaining inquiries in this round until one tag has been completely identified then 'killed' and all the remaining tags have been reset. If the reader detects a non-collision answer, the answer bit will be used as the next inquiring bit. Otherwise, the reader sends '0' as the next inquiring bit. The procedure will continue until all tags within the working zone of a reader are identified (then 'killed').

4.2.2 Query –Tree Scheme

Query-tree scheme is a memoryless protocol in which the tags do not have to remember

their inquiring history, unlike binary-tree scheme in which a pointer position has to be remembered by each tag. Instead of sending one bit for each inquiring, the reader sends a prefix with a length range from 1 to the length of ID minus 1. The tags will send back the remaining bits of their IDs if their IDs have the same prefix. From the tags' response, the reader can tell the bit at which the collision is occurring. After that, the reader overrides the non-collision bits and extends the prefix directly to the collision bit.

Generally speaking, when tags' ID length is short and there are a large number of tags within the working zone, binary-tree scheme is more efficient; when tags' ID length is long and the number of tags within the working zone is quite limited, query-tree scheme will take the advantage.

4.3 SET Implementation for Binary Tree Protocol

Hereafter, we demonstrate our procedure of implementing a RFID Binary-tree protocol FSM. The procedure is actually applicable to any digital logic circuits.

4.3.1 Binary-Tree Protocol FSM Structure

Table.4.1 describes the state assignment for a binary-tree scheme FSM, and Fig.4.1 shows the state diagram of the protocol (for more details about the protocols, refer to [25, 26]). In this figure, signals *NULL* and *CLK* come from the reader. The *CLK* is the synchronized clock shared with all tags. The *NULL* reboots all tags with the reader's working range into a new ID recognition procedure. When a tag receives the *NULL* signal, its next state will be set to "S1" no matter what the current state is. The signal *LSB* indicates whether the tag's ID bit under checking is the last bit. The signal *DIFF* shows whether or not the data bit received from the reader matches the current ID bit on the tag. Q1 and Q0 are the two-bit output signals of the state machine, representing 4 states shown in Table.4.1. A

logic-level implementation of the state machine is shown in Fig.4.2.

Table.4.1. State assignment

State		Q1	Q0
S0	quiet	0	0
S1	receiving	0	1
S2	sending	1	1
S3	disabled	1	0

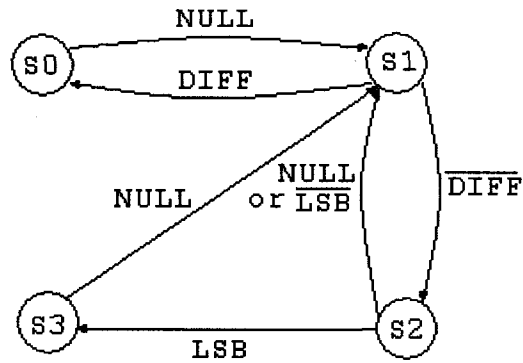


Fig.4.1. State diagram for Binary-tree protocol

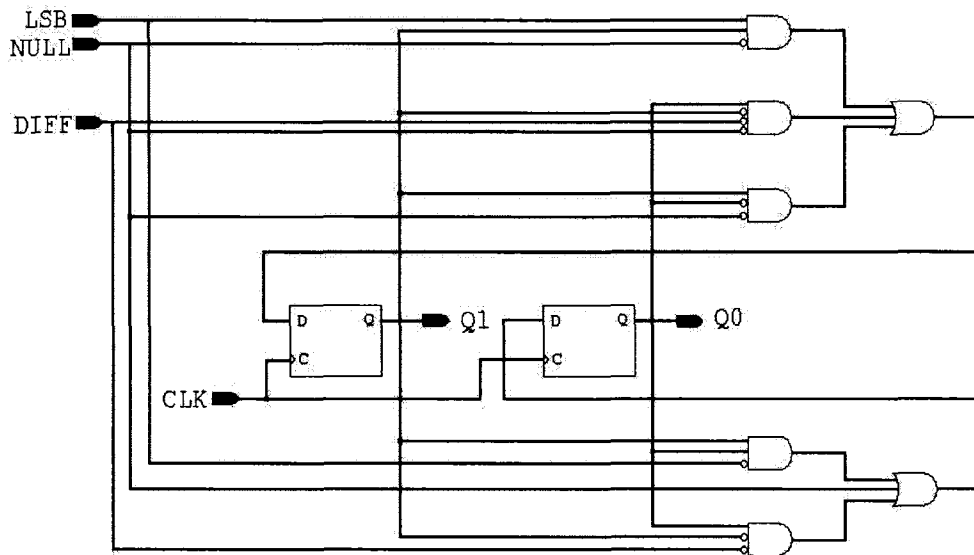


Fig.4.2. Logic-level implementation of the state machine for Binary-tree protocol

In Fig.4.2, there are 7 logic gates and 2 positive edge-triggered D-Flip-Flops (DFFs). As for those 7 logic gates, even though we may not find a counterpart gate type from a CMOS digital library, we can implement each of them with one SET threshold gate, directly following our procedure described in Chapter. 3.

With the same procedure, we can also implement a positive edge-triggered DFF, a typical sequential logic function, by using 4 SET threshold gates.

4.3.2 SET Positive Edge-Triggered D-Flip-Flop (DFF)

An edge-triggered D-flip-flop is shown in Fig.4.3, where a circle denotes a threshold logic gate, and the numbers appearing at the inputs of the gate represent the values of positive (+) or negative (-) weights. The detailed description of all gates is given in Table.4.1.

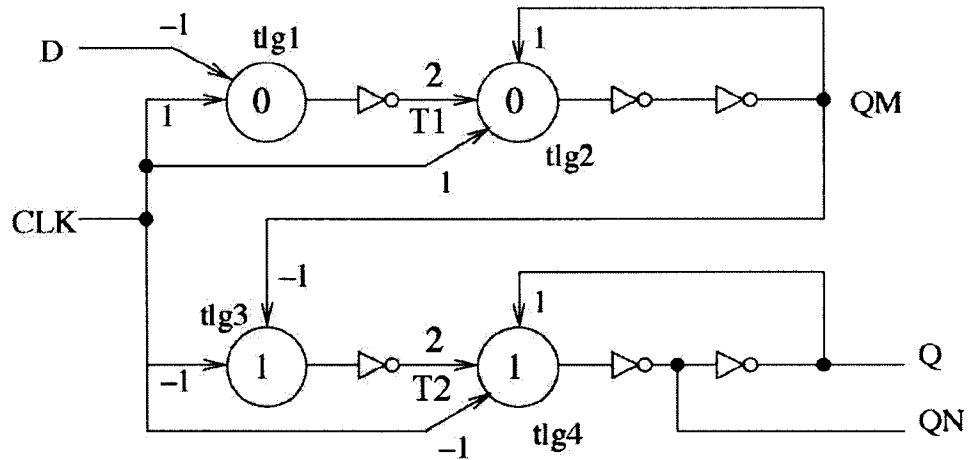


Fig.4.3. Positive edge-triggered D-flip-flop

Table.4.2. Threshold logic gates in Fig.4.3

Gate	Threshold logic
tlg1	$\text{sgn}\{D - \text{CLK}\}$
tlg2	$\text{sgn}\{-\text{CLK} - 2T1 - \text{QM} + 1.5\}$
tlg3	$\text{sgn}\{\text{CLK} + \text{QM} - 1.5\}$
tlg4	$\text{sgn}\{\text{CLK} - 2T2 - \text{Q} + 0.5\}$

4.4 Simulation with SIMON (SIMulation of Nanostructure)

SIMON is a most commonly used single-electron tunnel devices and circuits simulator based on a Monte Carlo method. SIMON is a PC windows platform program, with a graphic user interface shown in Fig.4.4.

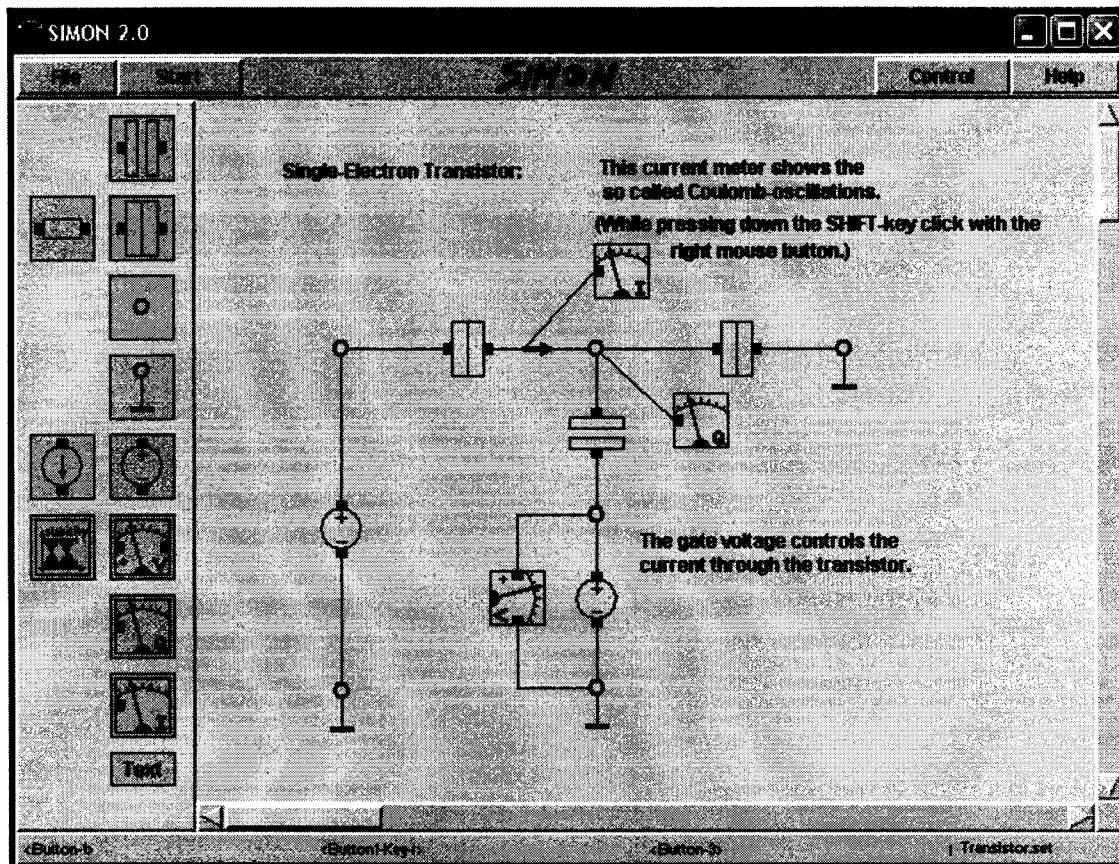


Fig.4.4. SIMON 2.0 Graphic User Interface

The implementation of RFID binary-tree protocol FSM using SET devices is shown in Fig.4.5, where logic '1' = 16mV, logic '0' = 0V and, for all tunnel junctions, $R_t = 105 \Omega$, $C_t = 10^{-19} \text{ F}$. As mentioned in Chapter.3, SET-only architectures require the buffers because of low driving capability with SET gates. A structure of the buffer with SET junctions is shown in Fig.3.3, where typical values of parameters are also given. This structure is needed in almost every threshold gate in our circuit, as can be seen in Fig.4.5 where all gates are labeled from 'A' through 'O'. The capacitance parameters ($\times 10^{-19} \text{ F}$) for these individual gates are summarized in Table.4.3.

The SET circuit of Fig.4.5 was simulated using SIMON (Simulation Of Nanostructure) [27]. The simulation time step was set to be 0.1 ms. Fig.4.6 shows a 4-bit ID

identification procedure. The simulation starts with state S1 (i.e, Q1 = '0', Q0 = '1'). The state transitions are shown partially in Table.4.4. We see that in clock cycles #2 through #8, the state changes between S1 and S2, and ends up with S3. This process represents the successful identification of a 4-bit ID tag. A 'high' *DIFF* at clock cycle #13 puts the machine into state S0 which holds until a next 'high' *NULL* signal arrives.

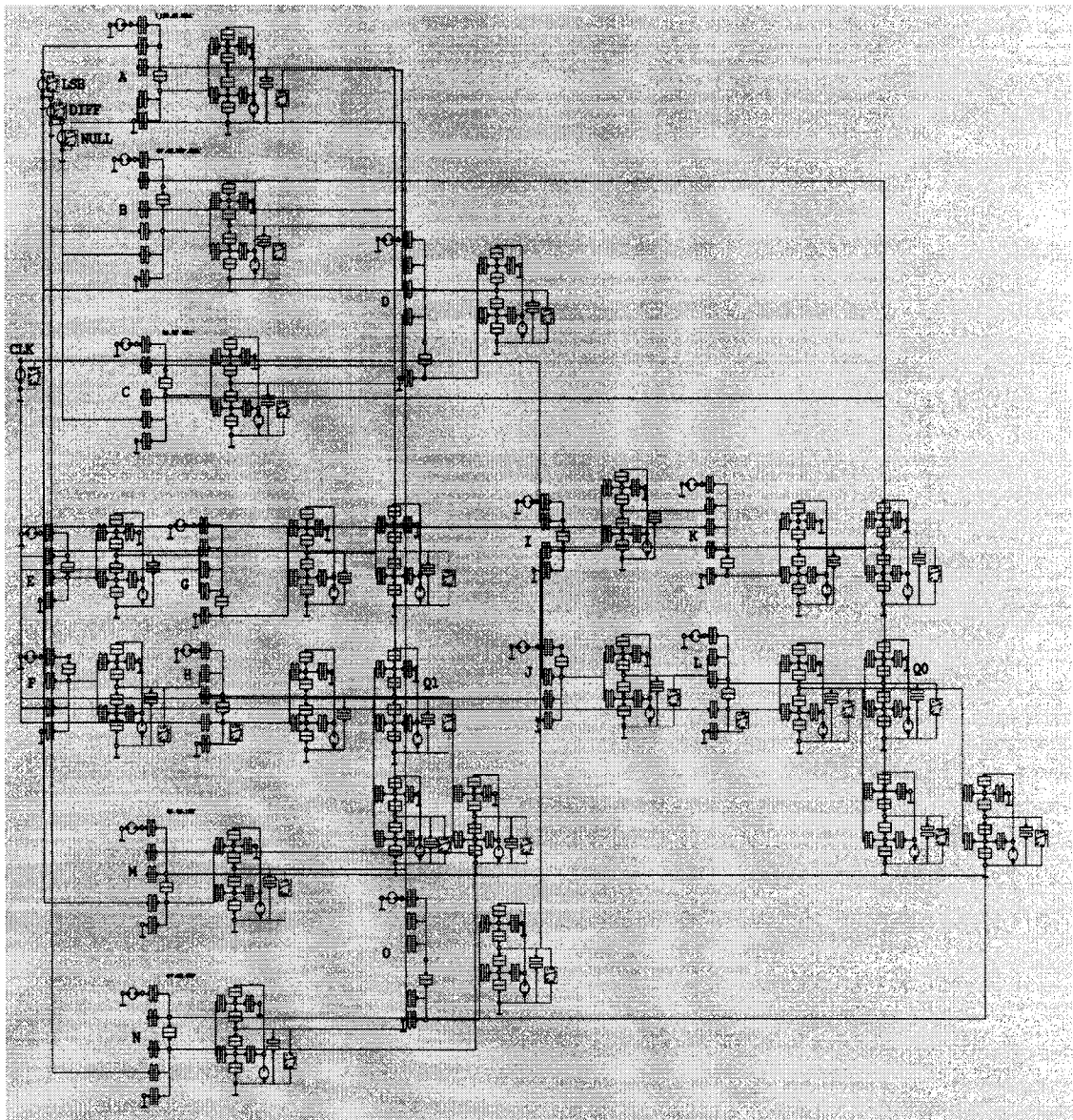


Fig.4.5. SET implementation of the state machine for Binary-tree protocol

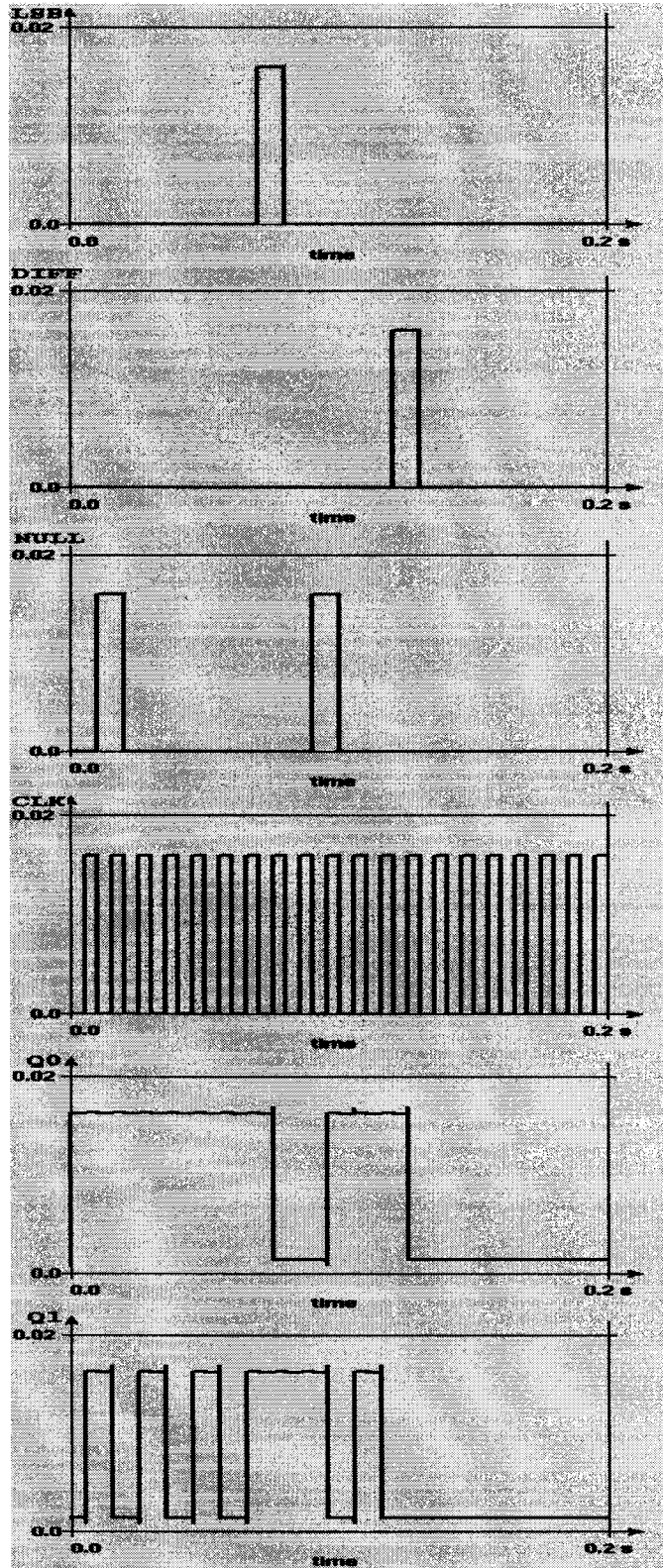


Fig.4.6. Simulation result of Fig.4.5

Table.4.3. Capacitance parameters ($\times 10^{-19}$ F) for threshold gates in Fig.4.5

Gate	C_b	C_o	C^p			C^n		
A	117	80	5	5	5			
B	110	80	5			5	5	5
C	117	80	5			5	5	
D	119	80	5	5	5	-		
E	120	86	5			5		
F	132	80	-			5	5	
G	117	90	10	5	5			
H	126	86	10	5		5		
I	120	86	5			5		
J	132	80	-			5	5	
K	119	90	10	5	5	-		
L	126	86	10	5		5		
M	117	80	5	5		5		
N	117	80	5			5	5	
O	117	80	5	5		5		

Table.4.4. State transitions in Fig.4.6.

Clock Cycle	State	Q1	Q0	State transition Condition
#1	S2	1	1	DIFF = '0'
#2	S1	0	1	NULL = '1'
#3	S2	1	1	DIFF = '0'
#4	S1	0	1	LSB = '0'
#5	S2	1	1	DIFF = '0'
#6	S1	0	1	LSB = '0'
#7	S2	1	1	DIFF = '0'
#8	S3	1	0	LSB = '1'
#9	S3	1	0	NULL = '0'
#10	S1	0	1	NULL = '1'
#11	S2	1	1	DIFF = '0'
#12	S1	0	1	LSB = '0'
#13	S0	0	0	DIFF = '1'

4.5 Delay and Power Estimation

The critical path of the circuit in Fig.4.5 starts from the CLK to D-flip-flop, and ends at the output of the two-level combinational logic. By using the estimation method from [22], the delay can be calculated as around $0.3 | \ln(P_{error}) |$ ns which turns out to be 5.53 ns when $P_{error} = 10^{-8}$ is assumed.

In order to estimate the power dissipation of Fig.4.5, we have to select a sequence of state transitions that occur during the tag identification process, because the switching energy varies for different states. If we have only one 4-bit ID tag in the reader's working range, the state machine will have the following sequence of state transitions: $S1 \rightarrow S2 \rightarrow S1 \rightarrow S2 \rightarrow S1 \rightarrow S2 \rightarrow S3$. With the power analysis method of [22], the total switching energy can be calculated to be about 702 meV. Assuming a clock frequency of 100 MHz, one can expect the total power consumption to be about 10 pW, compared to a typical $0.18 \mu\text{m}$ CMOS implementation which may consume power in the order of magnitude of $\sim 100 \mu\text{W}$ at the same frequency.

4.6 Summary

In this chapter, we presented an FSM implementation using single electron encoded logic in SET technology with the goal of obtaining ultra-low power solution. We have taken the RFID binary-tree protocol as an example to show the details about implementation procedure together with its power and delay analysis. The simulation results have shown significant power savings, compared with the traditional CMOS circuits.

Part of this work was reported in our recent paper titled "Finite State Machine Implementation with Single-Electron Tunneling Technology" which has been accepted by 2006 ISVLSI (International Symposium on VLSI).

Chapter 5

Conclusions and Future Work

In this thesis, we presented two low power approaches. One for CMOS technology, and the other for single electron tunneling devices.

For CMOS technology, we proposed a greedy algorithm to solve the power-oriented delay budgeting problem for combinational circuits. The topological information of the circuits as well as the cell library are used for aggressive power optimization in order to obtain good power-delay tradeoff. A Java Implementation with Graphic User Interface was developed. Experimental comparison has also been made between a pure delay-budgeting algorithm and the proposed algorithm in terms of power savings. As a sequential circuit can be transformed into several combinational sub-circuits, our greedy algorithm with the software tool can be further developed to do power-oriented delay budgeting for any digital circuits, both combinational and sequential. Also, further study is needed to apply the delay-budgeting method to high-level design, and develop one single unified metric (instead of separately-treated metrics of area, delay and power) for future design.

For SET circuit, we proposed a strategy to select parameters and analyze the reliability for single-electron threshold logic. Two typical logic gates (AND and NOR) were detailed with power and delay analysis as examples. The effect of possible background charges has been taken into account for the reliability improvement. While a uniform or normal distribution of random background charges is assumed throughout the paper, the proposed method can apply to any other distributions. Further work is also needed to deal with the correlated distribution of background charges.

As an application example, we presented an FSM implementation with the goal of obtaining ultra-low power solution. We have taken the RFID binary-tree protocol as an example to show the details about implementation procedure together with its power and delay analysis. The simulation results have shown significant power savings, compared with the traditional CMOS circuits.

Appendix A

Program 1 — Power Oriented Delay Budgeting Greedy Algorithm with GUI (Chapter 2)

```

/*****
*Functions Profile:
With a specified circuit gate level topology and a chosen digital layout library as input, this
tool can estimate how much power dissipation (percentage) can be saved within a given critical
path circuit delay. Also, the program can further estimate how much power saving potential by
relaxing the critical path delay a specified circuit has. Based on this information, the tool
can draw a circuit level power-delay curve.

*Usage:
java com.Greedy.GreedyApp;

*Package structure:
Package com.Greedy
    |--- Class GreedyApp      // Application launcher
    |--- Class GreedyFrame   // Greedy Algorithm with Graphic User Interface (GUI)
    |--- Class Out2Gates     // SIS benchmark circuits (.out) file input
    |--- Class Gates2Graph   // DAG (directed-acyclic graph) holder
    |--- Class Node         // DAG Node holder
*****/
```

```

//GreedyApp.java
package com.Greedy;
import java.awt.*;
import javax.swing.UIManager;

public class GreedyApp{
    boolean packFrame = false;
    public GreedyApp() {
        GreedyFrame frame = new GreedyFrame();

        //Pack frames that have useful preferred size info, e.g. from their layout
        //Validate frames that have preset sizes
        if (packFrame)
            frame.pack();
        else
            frame.validate();

        // Center the frame
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height)
            frameSize.height = screenSize.height;
        if (frameSize.width > screenSize.width)
            frameSize.width = screenSize.width;
        frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
frameSize.height) / 2);
        frame.setVisible(true);
    }

    static public void main(String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch(Exception e) {
            e.printStackTrace();
        }
        new GreedyApp();
    }
}

```



```

//GreedyFrame.java
package com.Greedy;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.ImageIcon;
import javax.swing.event.*;
import java.awt.image.*;
import javax.imageio.*;
import java.io.*;
import java.awt.geom.*;

public class GreedyFrame extends JFrame {
    private double zoom = 1;
    private double dimension_coefficient = 50;
    private double radius = 0.2;

    private int from_T = 1;
    private int to_T = 4;
    private int sample_number = 10;

    JPanel contentPane;
    BorderLayout borderLayout1 = new BorderLayout();
    JMenuBar jMenuBar1 = new JMenuBar();
    JMenu jMenuFile = new JMenu();
    JMenuItem jMenuItemFileOpen = new JMenuItem();
    JMenuItem jMenuItemFileExit = new JMenuItem();
    JMenu jMenuHelp = new JMenu();
    JMenuItem jMenuItemHelpAbout = new JMenuItem();
    JMenu jMenuCurve = new JMenu();
    JMenuItem jMenuItemCurveDraw = new JMenuItem();
    JMenuItem jMenuItemCurveExport = new JMenuItem();
    JMenuItem jMenuItemCurveSettings = new JMenuItem();

    JToolBar jToolBar1 = new JToolBar();
    JPanel jPanel1 = new JPanel();
    JButton jButton1 = new JButton();
    JButton jButton2 = new JButton();
    ImageIcon image1;
    ImageIcon image2;
    ImageIcon image3;

```

```

JFileChooser jFileChooser1 = new JFileChooser();
BorderLayout BorderLayout2 = new BorderLayout();
JPanel jPanel2 = new JPanel();
JPanel jPanel3 = new JPanel();
JPanel jPanel4 = new JPanel();
BorderLayout BorderLayout4 = new BorderLayout();
JLabel statusBar = new JLabel();
JPanel jPanel5 = new JPanel();
BorderLayout BorderLayout3 = new BorderLayout();
GridLayout GridLayout1 = new GridLayout();
JButton jButton5 = new JButton();
JPanel jPanel6 = new JPanel();
BorderLayout BorderLayout5 = new BorderLayout();
JPanel jPanel7 = new JPanel();
BorderLayout BorderLayout6 = new BorderLayout();
JList jList1 = new JList();
BorderLayout BorderLayout7 = new BorderLayout();
JButton jButton4 = new JButton();
JPanel drawpanel;
JTextArea jTextArea1;
private Gates2Graph graph;
private Out2Gates out;
JButton jButton3 = new JButton();
JButton jButton6 = new JButton();

/**
 * Construct the frame
 */
public GreedyFrame() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    runGreedyAlgorithm();
    try {
        jbInit();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

public void runGreedyAlgorithm(){
    graph = new Gates2Graph();

```

```

graph.run_once_releaseable_power(0);
//graph.write_power_plot_file(0.1, 31);
}

private void jbInit() throws Exception {
    image1 = new ImageIcon(GreedyFrame.class.getResource("openFile.gif"));
    image2 = new ImageIcon(GreedyFrame.class.getResource("closeFile.gif"));
    image3 = new ImageIcon(GreedyFrame.class.getResource("help.gif"));
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(borderLayout1);
    this.setTitle("Power-Oriented Delay Budgeting -- " + graph.get_title());
    this.setJMenuBar(jMenuBar1);
    this.setSize(new Dimension(800,600));

    //jToolBar1.setAlignmentY((float) 0.5);
    jButton1.setIcon(image1);
    jButton1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            jButton1_actionPerformed(e);
        }
    });
    jButton1.setToolTipText("Open File");

    jButton2.setIcon(image3);
    jButton2.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            jButton2_actionPerformed(e);
        }
    });
    jButton2.setToolTipText("About");

    jPanel1.setLayout(borderLayout2);
    jPanel4.setLayout(gridLayout1);
    jPanel2.setLayout(borderLayout4);
    jPanel5.setLayout(borderLayout3);

    jButton5.setText("zoom in");
    jButton5.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
            jButton5_actionPerformed(e);
        }
    });
}

```

```

});
jPanel3.setLayout(borderLayout5);
jPanel6.setLayout(borderLayout7);
jPanel7.setLayout(borderLayout6);
jButton4.setText("zoom out");
jButton4.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jButton4_actionPerformed(e);
    }
});
//menu file
jMenuFile.setText("File");
jMenuFileExit.setText("Exit");
jMenuFileExit.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jMenuFileExit_actionPerformed(e);
    }
});
statusBar.setToolTipText("");
statusBar.setText(" ");
jMenuFileOpen.setText("Open");
jButton3.setToolTipText("About");
jButton3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jButton3_actionPerformed(e);
    }
});
jButton3.setIcon(image2);
jButton6.setToolTipText("");
jButton6.setText("screen fit");
jButton6.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jButton6_actionPerformed(e);
    }
});
jMenuFile.add(jMenuFileOpen);
jMenuFileOpen.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jMenuFileOpen_actionPerformed(e);
    }
});

```

```

jMenuFile.add(jMenuFileExit);
//menu pd curve
jMenuCurve.setText("PD Curve");
jMenuCurve.add(jMenuCurveDraw);
jMenuCurveDraw.setText("Draw");
jMenuCurveDraw.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jMenuCurveDraw_actionPerformed(e);
    }
});
jMenuCurve.add(jMenuCurveExport);
jMenuCurveExport.setText("Export");
jMenuCurveExport.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jMenuCurveExport_actionPerformed(e);
    }
});
jMenuCurve.add(jMenuCurveExport);

jMenuCurveSettings.setText("Settings");
jMenuCurveSettings.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jMenuCurveSettings_actionPerformed(e);
    }
});
jMenuCurve.add(jMenuCurveSettings);
//help
jMenuHelp.setText("Help");
jMenuHelpAbout.setText("About");
jMenuHelpAbout.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jMenuHelpAbout_actionPerformed(e);
    }
});
jMenuHelp.add(jMenuHelpAbout);
jMenuBar1.add(jMenuFile);
jMenuBar1.add(jMenuCurve);
jMenuBar1.add(jMenuHelp);
contentPane.add(jToolBar1, BorderLayout.NORTH);
jToolBar1.add(jButton1);

```

```

jToolBar1.add(jButton3, null);
jToolBar1.add(jButton2);
contentPane.add(jPanel1, BorderLayout.CENTER);
jPanel1.add(jPanel2, BorderLayout.CENTER);
jPanel1.add(jPanel3, BorderLayout.EAST);
jPanel3.add(jPanel6, BorderLayout.NORTH);
jPanel6.add(jButton6, BorderLayout.NORTH);
jPanel6.add(jButton5, BorderLayout.SOUTH);
jPanel6.add(jButton4, BorderLayout.CENTER);
jPanel3.add(jPanel7, BorderLayout.CENTER);
jPanel7.add(new JScrollPane(jList1), BorderLayout.CENTER);
contentPane.add(statusBar, BorderLayout.SOUTH);
jPanel2.add(jPanel4, BorderLayout.SOUTH);
jPanel2.add(jPanel5, BorderLayout.CENTER);

//jPanel7 -- list
listNodes();
jList1.addListSelectionListener(new NodeListSelectionListener());

//jPanel4 -- info
//String teststring = new String("this is the text string with enough length, so that it
can test the wrapability");
jTextAreal= new JTextArea(graph.show_graph_info());
//textpanel = new JTextArea(teststring);
jTextAreal.setBackground(Color.black);
jTextAreal.setForeground(Color.orange);
jTextAreal.setLineWrap(true);
jTextAreal.setRows(4);
jPanel4.add(new JScrollPane(jTextAreal));

//jPanel5 -- draw
drawpanel = new JPanel(new BorderLayout());
drawpanel.addMouseListener(new NodeMouseListener());
drawpanel.add(new GreedyDraw(), null);
jPanel5.add(new JScrollPane(drawpanel));

//jScrollPane1.setViewport().setViewPosition(new Point(50, 50));
setVisible(true);
}

/**

```

```

* Overridden so we can exit when window is closed
*
* @param e WindowEvent
*/
protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
        System.exit(0);
    }
}

/**
* File | Exit action performed
*
* @param e ActionEvent
*/
public void jMenuItemFileOpen_actionPerformed(ActionEvent e) {
    fileOpen();
}

public void jMenuItemFileExit_actionPerformed(ActionEvent e) {
    System.exit(0);
}

public void jMenuItemCurveDraw_actionPerformed(ActionEvent e) {
    drawCurve();
}

public void jMenuItemCurveExport_actionPerformed(ActionEvent e) {
    exportCurve();
}

public void jMenuItemCurveSettings_actionPerformed(ActionEvent e) {
    initCurve();
}

/**
* Help | About action performed
*
* @param e ActionEvent
*/
public void jMenuItemHelpAbout_actionPerformed(ActionEvent e) {
    String message = "Power Oriented Delay Budgeting Greedy Algorithm\nversion 1.0.0";
    String message_title = "Help";
}

```

```

        JOptionPane.showMessageDialog(this, message, message_title,
                                     JOptionPane.DEFAULT_OPTION);
    }

    void jButton2_actionPerformed(ActionEvent e) {
        String message = "Power Oriented Delay Budgeting Greedy Algorithm\nversion 1.0.0";
        String message_title = "Help";
        JOptionPane.showMessageDialog(this, message, message_title,
                                     JOptionPane.DEFAULT_OPTION);
    }

    //inner class NodeMouseListener
    protected class NodeMouseListener extends MouseAdapter {
        public void mouseClicked(MouseEvent evt) {
            if ((evt.getModifiers() & InputEvent.BUTTON1_MASK) != 0) {
                processLeft(evt.getX(), evt.getY());
            }
            if ((evt.getModifiers() & InputEvent.BUTTON2_MASK) != 0) {
                //processMiddle(evt.getPoint());
            }
            if ((evt.getModifiers() & InputEvent.BUTTON3_MASK) != 0) {
                //processRight(evt.getPoint());
            }
        }

        public void processLeft(int x, int y){
            int[] select = graph.findSelectedNodes(x, y, zoom, dimension_coefficient,
radius);
            graph.setSelectedNodes(select);
            jList1.setSelectedIndices(select);
            jTextAreal.setText(graph.setTextArea(select));
            redraw(zoom);
        }
    }

    // inner class GreedyDraw
    protected class GreedyDraw extends JPanel
    {
        protected void paintComponent(Graphics g)
        {
            super.paintComponent(g);
            Graphics2D g2 = (Graphics2D)g;

```



```

g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                    RenderingHints.VALUE_ANTIALIAS_ON);
setBackground(Color.white);
graph.draw_graph(false, //boolean selected_part_only,
                g2, //Graphics2D g2,
                zoom,
                radius,
                dimension_coefficient,
                Color.black, //Color node_color,
                Color.lightGray //Color line_color
                );
graph.draw_graph(true, //boolean selected_part_only,
                g2, //Graphics2D g2,
                zoom,
                radius,
                dimension_coefficient,
                Color.blue, //Color node_color,
                Color.red //Color line_color
                );

}

/**
 * informs parent what size this component needs for it's display
 */
public Dimension getPreferredSize()
{
    return graph.draw_size(zoom, dimension_coefficient);
}

}

protected class NodeListSelectionListener implements ListSelectionListener {
    // This method is called each time the user changes the set of selected items
    public void valueChanged(ListSelectionEvent evt) {
        // When the user release the mouse button and completes the selection,
        // getValueIsAdjusting() becomes false
        int index = evt.getFirstIndex();
        if (!evt.getValueIsAdjusting()) {
            // Get all selected items
            int[] selected = jList1.getSelectedIndices();

```

```

        graph.setSelectedNodes(selected);
        jTextArea1.setText(graph.setTextArea(selected));
        redraw(zoom);
    }
}

void jButton5_actionPerformed(ActionEvent e) {
    redraw(zoom * 2);
}

void jButton4_actionPerformed(ActionEvent e) {
    redraw(zoom * 0.5);
}

void redraw(double zoom){
    this.zoom = zoom;
    //jPanel5 -- draw
    drawpanel.removeAll();
    drawpanel.add(new GreedyDraw(), null);
    jPanel5.removeAll();
    jPanel5.add(new JScrollPane(drawpanel));
    setVisible(true);
}

void jButton1_actionPerformed(ActionEvent e) {
    fileOpen();
}

void fileOpen() {
    String circuit;
    // Use the OPEN version of the dialog, test return for Approve/Cancel
    try{
        if (JFileChooser.APPROVE_OPTION == jFileChooser1.showOpenDialog(this)) {
            openFile(jFileChooser1.getSelectedFile().getPath());
        }
        circuit = jFileChooser1.getSelectedFile().getName();
        graph.set_title(circuit);
        this.setTitle("Power-Oriented Delay Budgeting" + " -- " + circuit);
        this.repaint();
    }
}

```

```

    }catch (Exception e) {
    }
}

/**
 * Open named file; read text from file into JTextArea1; report to
 * statusBar.
 *
 * @param fileName String
 */
void openFile(String fileName) {
    try
    {
        out = new Out2Gates();
        out.read_out_write_switching(fileName);
        out.read_out_write_gate(fileName);
        runGreedyAlgorithm();
        listNodes();
        JTextArea1.setText(graph.show_graph_info());
        redraw(1);
    }
    catch (Exception e)
    {
        statusBar.setText("Error opening " + fileName);
    }
}

void listNodes(){
    jList1.setListData(graph.getNodeList());
}

protected class GreedyCurveDraw extends JPanel
{
    protected void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

        int w = getWidth();
        int h = getHeight();
    }
}

```

```

this.setBackground(Color.white);

//get x, y dimension labels
Image image_x = Toolkit.getDefaultToolkit().getImage("x.gif");
int width_x = image_x.getWidth(null);
int height_x = image_x.getHeight(null);
Image image_y = Toolkit.getDefaultToolkit().getImage("y.gif");
int height_y = image_y.getHeight(null);
g2.drawImage(image_x, (w - width_x)/2, h - height_x, this);
g2.drawImage(image_y, 0, (h - height_y)/2, this);

float width = w;
float height = h;
float draw_width = width/10*8;
float draw_height = height/10*8;

g2.drawString("100%",width/10, height/10);
g2.drawString("80%",width/10, height/10 + draw_height/5);
g2.drawString("60%",width/10, height/10 + draw_height/5*2);
g2.drawString("40%",width/10, height/10 + draw_height/5*3);
g2.drawString("20%",width/10, height/10 + draw_height/5*4);

Image h3 = Toolkit.getDefaultToolkit().getImage("h3.gif");

for (int index = from_T; index < to_T + 1; index++){
    g2.drawString(Integer.toString(index),width/10 + draw_width/(to_T - from_T)*(index
- from_T), height/10 + draw_height + 12);
    g2.drawImage(h3, (int)(width/10 + draw_width/(to_T - from_T)*(index - from_T)),
(int)(height/10 + draw_height - 3), this);
}

double step_length = ((double)to_T - (double)from_T ) /sample_number;
graph.write_power_plot_file(step_length, sample_number + 1);
graph.drawCurve(g2, w, h);

// g2.setPaint(Color.red);
// g2.draw(new Line2D.Double(0, 0, w, h));
// g2.drawOval(0, 0, w, h);
}

```

```

}

void drawCurve() {
    JPanel panel = new JPanel(new BorderLayout());
    panel.add(new GreedyCurveDraw(), null);

    JDialog f = new JDialog();
    f.setTitle("Power-Delay Curve");
    //f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.getContentPane().add(new JScrollPane(panel));
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    f.setSize(400, 400);
    Dimension framesize = f.getSize();
    if (framesize.height > screenSize.height/2){
        framesize.height = screenSize.height/2;
        framesize.width = framesize.height;
    }
    if (framesize.width > screenSize.width/2){
        framesize.width = screenSize.width/2;
        framesize.height = framesize.width;
    }
    f.setSize(framesize);
    f.setLocation( (screenSize.width - framesize.width) / 2,
        (screenSize.height - framesize.height) / 2);
    f.setVisible(true);
}

void exportCurve() {
    try{
        int w= 400;
        int h = 400;

        BufferedImage bufferedImage = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
        // Create a graphics contents on the buffered image
        Graphics2D g2 = bufferedImage.createGraphics();
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        g2.setColor(Color.white);
        g2.fillRect(0, 0, w, h);
        g2.setColor(Color.black);
        //Image image_x = Toolkit.getDefaultToolkit().getImage("x.gif");
    }
}

```

```

Image image_x = new javax.swing.ImageIcon("x.gif").getImage();
int width_x = image_x.getWidth(null);
int height_x = image_x.getHeight(null);
//Image image_y = Toolkit.getDefaultToolkit().getImage("y.gif");
Image image_y = new javax.swing.ImageIcon("y.gif").getImage();
int height_y = image_y.getHeight(null);
g2.drawImage(image_x, (w - width_x)/2, h - height_x, null);
g2.drawImage(image_y, 0, (h - height_y)/2, null);

float width = w;
float height = h;
float draw_width = width/10*8;
float draw_height = height/10*8;

g2.drawString("100%",width/10, height/10);
g2.drawString("80%",width/10, height/10 + draw_height/5);
g2.drawString("60%",width/10, height/10 + draw_height/5*2);
g2.drawString("40%",width/10, height/10 + draw_height/5*3);
g2.drawString("20%",width/10, height/10 + draw_height/5*4);

//Image h3 = Toolkit.getDefaultToolkit().getImage("h3.gif");
Image h3 = new javax.swing.ImageIcon("h3.gif").getImage();

for (int index = from_T; index < to_T + 1; index++){
    g2.drawString(Integer.toString(index),width/10 + draw_width/(to_T - from_T)*(index -
from_T), height/10 + draw_height + 12);
    g2.drawImage(h3, (int)(width/10 + draw_width/(to_T - from_T)*(index - from_T)),
(int)(height/10 + draw_height - 3), null);
}

double step_length = ((double)to_T - (double)from_T ) /sample_number;
graph.write_power_plot_file(step_length, sample_number + 1);
graph.drawCurve(g2, w, h);

FileDialog fd = new FileDialog(this, "Save as JPEG", FileDialog.SAVE);
fd.setFile("pd_curve.jpg");
fd.show();
String name = fd.getFile();

ImageIO.write(bufferedImage, "jpg", new File(fd.getDirectory() + name));

```

```

    }catch (Exception e) {
    }
}

void initCurve() {
    int to_temp = to_T;
    String input_holder;// A local variable to hold the name.
    input_holder = JOptionPane.showInputDialog(null, "Timing Constraints (T/T0)");
    try{
        if (input_holder != null)
            to_temp = Integer.parseInt(input_holder);
        if (to_temp > 1)
            to_T = to_temp;
    } catch (Exception e) {
    }
}

void jButton3_actionPerformed(ActionEvent e) {
    fileSave();
}

void fileSave() {
    // Use the OPEN version of the dialog, test return for Approve/Cancel
    try{
        FileDialog fd = new FileDialog(this, "Save as JPEG", FileDialog.SAVE);
        fd.setFile("circuit.jpg");
        fd.show();
        String name = fd.getFile();
        int width = (int)graph.draw_size(zoom, dimension_coefficient).getWidth();
        int height = (int)graph.draw_size(zoom, dimension_coefficient).getHeight();

        BufferedImage bufferedImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        // Create a graphics contents on the buffered image
        Graphics2D g2 = bufferedImage.createGraphics();
        g2.setColor(Color.white);
        g2.fillRect(0, 0, width, height);
        graph.draw_graph(false, //boolean selected_part_only,
                        g2, //Graphics2D g2,
                        zoom,
                        radius,

```

```

        dimension_coefficient,
        Color.black, //Color node_color,
        Color.lightGray //Color line_color
    );
    graph.draw_graph(true, //boolean selected_part_only,
        g2, //Graphics2D g2,
        zoom,
        radius,
        dimension_coefficient,
        Color.blue, //Color node_color,
        Color.red //Color line_color
    );
    ImageIO.write(bufferedImage, "jpg", new File(fd.getDirectory() + name));
} catch (Exception e) {
}
}

void jButton6_actionPerformed(ActionEvent e) {
    double width_screen = jPanel5.getBounds().getWidth();
    double height_screen = jPanel5.getBounds().getHeight();
    zoom = 1;
    double width_draw = graph.draw_size(zoom, dimension_coefficient).getWidth();
    double height_draw = graph.draw_size(zoom, dimension_coefficient).getHeight();
    while ((width_draw > width_screen) || (height_draw > height_screen)){
        zoom = zoom / 2;
        width_draw = graph.draw_size(zoom, dimension_coefficient).getWidth();
        height_draw = graph.draw_size(zoom, dimension_coefficient).getHeight();
    }
    redraw(zoom);
}
}

//Gates2Graph.java
package com.Greedy;
import java.io.*;
import java.util.*;
import java.lang.reflect.Array;
import java.awt.Dimension;
import java.awt.Graphics2D;
import java.awt.Color;

```



```

import java.awt.geom.*;

public class Gates2Graph{
    private List graph_node_list;
    private String title;

    private double initial_power = 0;
    private double T0 = 0;
    private double PS = 0;
    private double power_PS = 0;
    private List candidate_node_list;

    private int output_pos = 5;
    private List description_list;
    private List primary_input_list;
    private List primary_output_list;
    private List gate_switching_list;
    private Dimension graph_size;

    private double[] plot_array;

    public static void main(String[] args){
        Gates2Graph g = new Gates2Graph();
        g.run_once_releaseable_power(3.00);
        //g.run_once_releaseable_delay(2.99);
        //g.write_power_plot_file(0.1, 31);
        //g.write_delay_plot_file(0.5, 7);
    }

    public void write_power_plot_file(double percentage_step, int point_number){
        try {
            BufferedWriter out = new BufferedWriter(new FileWriter("Plots"));
            plot_array = new double[point_number];
            for(int index = 0; index < point_number; index++){
                clear();
                initial_graph(percentage_step * index);

                releaseable_power_greedy_algorithm();
                plot_array[index] = (1 - get_power_PS() / initial_power);
                out.write(index + "      " + (1 - get_power_PS() / initial_power) + "\n");
            }
        }
    }
}

```

```

        out.close();
    }catch (IOException e) {
    }
}

public void write_delay_plot_file(double percentage_step, int point_number){
    try {
        BufferedWriter out = new BufferedWriter(new FileWriter("Plots"));

        for(int index = 0; index < point_number; index++){
            clear();
            initial_graph(percentage_step * index);

            releaseable_greedy_algorithm();
            out.write((1 - get_power_PS() / initial_power) + " ");
        }
        /****
        clear();
        initial_graph(3);

        releaseable_power_greedy_algorithm();
        out.write( "\n" + (1 - get_power_PS() / initial_power) + "\n");
        /****
        out.close();
    }catch (IOException e) {
    }
}

public void clear(){
    initial_power = 0;
    PS = 0;
    power_PS = 0;
}

public void run_once_delay(double extra_slack_percentage){
    initial_graph(extra_slack_percentage);
    show_info();
    greedy_algorithm();
    System.out.println("\n*****");
    System.out.println("PS = " + get_PS());
    System.out.println("\n*****");
}

```

```

        show_info();
        System.out.println("\n*****");
        System.out.println("PS = " + get_PS());
        for (int index = 0; index < candidate_node_list.size(); index++) {
            Node node = (Node)candidate_node_list.get(index);
            System.out.println(node.get_nodename());
        }
        System.out.println("\n*****");
    }

    public void run_once_power(double extra_slack_percentage) {
        initial_graph(extra_slack_percentage); //extra_slack
        show_power_procedure();
        show_power_result();
    }

    public void run_once_releaseable_delay(double extra_slack_percentage) {
        initial_graph(extra_slack_percentage);
        show_info();
        releaseable_greedy_algorithm();
        System.out.println("\n*****");
        System.out.println("PS = " + get_power_PS());
        System.out.println("\n*****");
        show_info();
        System.out.println("\n*****");
        System.out.println("PS = " + get_power_PS());
        for (int index = 0; index < candidate_node_list.size(); index++) {
            Node node = (Node)candidate_node_list.get(index);
            System.out.println(node.get_nodename());
        }
        System.out.println("\n*****");
    }

    public void run_once_releaseable_power(double extra_slack_percentage) {
        initial_graph(extra_slack_percentage);
        show_info();
        releaseable_power_greedy_algorithm();
        System.out.println("\n*****");
        System.out.println("PS = " + get_power_PS());
        System.out.println("\n*****");
        show_info();
    }

```

```

        System.out.println("\n*****");
        System.out.println("PS = " + get_power_PS());
        for (int index = 0; index < candidate_node_list.size(); index++) {
            Node node = (Node)candidate_node_list.get(index);
            System.out.println(node.get_nodename());
        }
        System.out.println("\n*****");
    }

    public void show_power_procedure(){
        show_info();
        power_greedy_algorithm();
        System.out.println("\n*****");
        System.out.println("PS = " + get_power_PS());
        System.out.println("\n*****");
        show_info();
    }

    public void show_power_result(){
        System.out.println("\n*****");
        System.out.println("PS = " + get_power_PS());
        for (int index = 0; index < candidate_node_list.size(); index++) {
            Node node = (Node)candidate_node_list.get(index);
            System.out.println(node.get_nodename());
        }
        System.out.println("\n*****");
    }

    public double get_PS(){
        return PS;
    }

    public double get_power_PS(){
        return power_PS;
    }

    public double[] get_plot_array(){
        return plot_array;
    }

    public void new_graph_node_list(){

```

```

        graph_node_list = new ArrayList();
    }

    public void test1(){
        for (int index = 0; index < description_list.size(); index++){
            System.out.print(description_list.get(index) + " ");
        }
        System.out.println();

        for (int index = 0; index < primary_input_list.size(); index++){
            System.out.print(primary_input_list.get(index) + " ");
        }
        System.out.println();

        for (int index = 0; index < primary_output_list.size(); index++){
            System.out.print(primary_output_list.get(index) + " ");
        }
        System.out.println();

        for (int index = 0; index < gate_switching_list.size(); index++){
            System.out.print(gate_switching_list.get(index) + " ");
        }
        System.out.println();
    }

    public void read_Gates_file(){
        Node node = new Node();
        try {
            BufferedReader in = new BufferedReader(new FileReader("Gates"));

            int status = 0;//0: dull 1: work;
            String str;
            String [] gate_info;

            while ((str = in.readLine()) != null) {
                if(status == 0){
                    if(str.startsWith("name")){
                        description_list = new ArrayList();
                        gate_info = str.split(" ");
                        for (int index = 0; index < Array.getLength(gate_info); index++){
                            description_list.add(gate_info[index]);
                        }
                    }
                }
            }
        }
    }

```

```

        if (gate_info[index].equals("output"))
            output_pos = index;
    }
    status = 1;
}
else{
    continue;
}
}
else if(status == 1){
    if(str.startsWith("primary inputs:")){
        primary_input_list = new ArrayList();
        gate_info = str.substring(18).split(" ");
        for (int index = 0; index < Array.getLength(gate_info); index++){
            primary_input_list.add(gate_info[index]);
        }
        str = in.readLine();
        primary_output_list = new ArrayList();
        gate_info = str.substring(18).split(" ");
        for (int index = 0; index < Array.getLength(gate_info); index++){
            primary_output_list.add(gate_info[index]);
        }
        break;
    }
    else{
        gate_info = str.split(" ");

        node.set_gate_info(gate_info);

        node.set_nodename(get_gate_name(gate_info));
        node.set_category(get_gate_category(gate_info));
        node.set_area(get_gate_area(gate_info));
        node.set_weight(get_gate_weight(gate_info));

node.set_delay_coefficient(get_gate_delay_coefficient(gate_info));

node.set_power_coefficient(get_gate_power_coefficient(gate_info));
        graph_node_list.add(node);
        node = new Node();
        //System.out.println(gate_name + " " + gate_category + " " + gate_area);
    }
}

```

```

        }
    }
    in.close();

} catch (IOException e) {
}
}

public void read_Switchings_file(){
    gate_switching_list = new ArrayList();
    int jumped_line = 0;
    String str;
    try {
        BufferedReader in = new BufferedReader(new FileReader("Switchings"));
        while ((str = in.readLine()) != null) {
            if(jumped_line++ < primary_input_list.size() )
                continue;
            gate_switching_list.add(str);
        }
        in.close();
    } catch (IOException e) {
    }
}

public String get_gate_name(String [] gate_info){
    return gate_info[0];
}

public String get_gate_category(String [] gate_info){
    return gate_info[0].substring(0, gate_info[0].lastIndexOf("_"));
}

public double get_gate_area(String [] gate_info){
    double get_gate_area = 0;
    for(int index = 0; index < description_list.size(); index++){
        if (description_list.get(index).equals("area")){
            get_gate_area = Double.parseDouble(gate_info[index]);
            break;
        }
    }
    return get_gate_area;
}

```

```

}

public double get_gate_weight(String [] gate_info){
    double get_gate_weight = 1;
    for(int index = 0; index < description_list.size(); index++){
        if (description_list.get(index).equals("weight")){
            get_gate_weight = Double.parseDouble(gate_info[index]);
            break;
        }
    }
    return get_gate_weight;
}

public double get_gate_delay_coefficient(String [] gate_info){
    double get_gate_delay_coefficient = 1;
    for(int index = 0; index < description_list.size(); index++){
        if (description_list.get(index).equals("delay_coefficient")){
            get_gate_delay_coefficient = Double.parseDouble(gate_info[index]);
            break;
        }
    }
    return get_gate_delay_coefficient;
}

public double get_gate_power_coefficient(String [] gate_info){
    double get_gate_power_coefficient = 1;
    for(int index = 0; index < description_list.size(); index++){
        if (description_list.get(index).equals("power_coefficient")){
            get_gate_power_coefficient = Double.parseDouble(gate_info[index]);
            break;
        }
    }
    return get_gate_power_coefficient;
}

public boolean is_primary_input(String input){
    boolean is_primary_input = false;
    for(int index = 0; index < primary_input_list.size(); index++){
        if (input.equals(primary_input_list.get(index)))
            is_primary_input = true;
    }
}

```



```

        return is_primary_input;
    }

    public boolean is_primary_output(String output){
        boolean is_primary_output = false;
        for(int index = 0; index < primary_output_list.size(); index++){
            if (output.equals(primary_output_list.get(index)))
                is_primary_output = true;
        }
        return is_primary_output;
    }

    public void link_graph_node_list(){
        String [] gate_info_to;
        String [] gate_info_from;
        Node node_from = new Node();
        Node node_to = new Node();
        for(int index_out = 0; index_out < graph_node_list.size(); index_out++){
            node_to = (Node)graph_node_list.get(index_out);
            gate_info_to = node_to.get_gate_info();
            for (int index_in = 0; index_in < graph_node_list.size(); index_in++){
                node_from = (Node)graph_node_list.get(index_in);
                gate_info_from = node_from.get_gate_info();
                for (int index = output_pos + 1; index < Array.getLength(gate_info_to);
index++){
                    if (gate_info_to[index].equals(gate_info_from[output_pos])){
                        if (node_from.get_fanout_node_list() == null)
                            node_from.new_fanout_node_list();
                        node_from.get_fanout_node_list().add(node_to);
                        if (node_to.get_fanin_node_list() == null)
                            node_to.new_fanin_node_list();
                        node_to.get_fanin_node_list().add(node_from);
                    }
                }
            }
        }
    }

    public void calculate_nodes_position(){
        Node node = new Node();
        int column_max = 0;

```

```

for (int index = 0; index < graph_node_list.size(); index++){
    node = (Node)graph_node_list.get(index);
    node.calculate_column();
    if (column_max < node.get_column())
        column_max = node.get_column();
}
for (int column = 1; column <= column_max; column++){
    int row = 1;
    for (int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        if (node.get_column() == column){
            node.set_row(row++);
        }
    }
}

public void calculate_nodes_delay(){
    Node node = new Node();
    for (int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        node.calculate_delay();
    }
}

public void calculate_nodes_arrival_time(){
    Node node = new Node();
    for (int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        node.calculate_arrival_time();
    }
}

public void calculate_nodes_required_time(double extra_slack_percentage){
    Node node = new Node();
    double max_arrival_time = 0;
    for (int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        if (max_arrival_time < node.get_arrival_time())
            max_arrival_time = node.get_arrival_time();
    }
}

```

```

    }
    for (int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        node.calculate_required_time(max_arrival_time * (1 + extra_slack_percentage));
    }
    set_T0(max_arrival_time);
}

public void calculate_nodes_slack(){
    Node node = new Node();
    for (int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        node.calculate_slack();
    }
}

public void calculate_nodes_power(){
    Node node = new Node();
    for (int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        node.calculate_power();
    }
}

public void calculate_initial_power(){
    Node node = new Node();
    for (int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        initial_power += node.calculate_power();
    }
}

public void calculate_nodes_power_slack(){
    Node node = new Node();
    for (int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        node.calculate_power_slack();
    }
}

public void calculate_nodes_releaseable_slack(){

```

```

Node node = new Node();
for (int index = 0; index < graph_node_list.size(); index++){
    node = (Node)graph_node_list.get(index);
    node.calculate_releaseable_slack();
}
}

public void calculate_nodes_releaseable_power_slack(){
    Node node = new Node();
    for (int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        node.calculate_releaseable_power_slack();
    }
}

public void initial_graph(double extra_slack){
    new_graph_node_list();
    read_Gates_file();
    read_Switchings_file();
    link_graph_node_list();
    calculate_nodes_position();
    calculate_graph_size();
    calculate_nodes_delay();
    calculate_nodes_arrival_time();
    calculate_nodes_required_time(extra_slack);
    calculate_nodes_slack();
    add_switching_info();
    calculate_initial_power();
    calculate_nodes_power();
    calculate_nodes_power_slack();
    calculate_nodes_releaseable_slack();
    calculate_nodes_releaseable_power_slack();
}

public void add_switching_info(){
    Node node = new Node();
    int switching_index = 0;
    String str;
    for(int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        if((node.get_fanin_node_list() == null) && (node.get_fanout_node_list() ==

```

```

null)){
    }
    else{
        str = (String)gate_switching_list.get(switching_index);
        node.set_switching(Double.parseDouble(str));
        switching_index++;
    }
}

public Node node_max_local_PS(){
    double local_PS = 0;
    Node node_max_local_PS = new Node();
    Node node = new Node();
    for(int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        if (node.get_in_graph()){
            if (node.local_PS() >= local_PS){
                local_PS = node.local_PS();
                node_max_local_PS = node;
            }
        }
    }
    return node_max_local_PS;
}

public Node node_max_local_power_PS(){
    double local_power_PS = 0;
    Node node_max_local_power_PS = new Node();
    Node node = new Node();
    for(int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        if (node.get_in_graph()){
            if (node.local_power_PS() >= local_power_PS){
                local_power_PS = node.local_power_PS();
                node_max_local_power_PS = node;
            }
        }
    }
    return node_max_local_power_PS;
}

```

```

public Node node_max_releaseable_local_PS(){
    double releaseable_local_PS = 0;
    Node node_max_releaseable_local_PS = new Node();
    Node node = new Node();
    for(int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        if (node.get_in_graph()){
            if (node.releaseable_local_PS() >= releaseable_local_PS){
                releaseable_local_PS = node.releaseable_local_PS();
                node_max_releaseable_local_PS = node;
            }
        }
    }
    return node_max_releaseable_local_PS;
}

public Node node_max_releaseable_local_power_PS(){
    double releaseable_local_power_PS = 0;
    Node node_max_releaseable_local_power_PS = new Node();
    Node node = new Node();
    for(int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        if (node.get_in_graph()){
            if (node.releaseable_local_power_PS() >= releaseable_local_power_PS){
                releaseable_local_power_PS = node.releaseable_local_power_PS();
                node_max_releaseable_local_power_PS = node;
            }
        }
    }
    return node_max_releaseable_local_power_PS;
}

public boolean graph_empty(){
    boolean is_empty = true;
    Node node = new Node();
    for(int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        if (node.get_in_graph() == true)
            is_empty = false;
    }
}

```

```

        return is_empty;
    }

    public void greedy_algorithm(){
        Node node = new Node();
        candidate_node_list = new ArrayList();
        while(!graph_empty()){
            node = node_max_local_PS();
            PS += node.local_PS();
            switch(node.get_max_local_PS()){
                case 1: candidate_node_list.add(node);
                        break;
                case 2: candidate_node_list.addAll(node.get_fanin_node_list());
                        break;
                case 3: candidate_node_list.addAll(node.get_fanout_node_list());
                        break;
                default: break;
            }
            node.arrange_slack();
            node.delete_candidate_sub_graph();
        }
    }

    public void power_greedy_algorithm(){
        Node node = new Node();
        candidate_node_list = new ArrayList();
        while(!graph_empty()){
            node = node_max_local_power_PS();
            power_PS += node.local_power_PS();
            candidate_node_list.add(node);
            node.arrange_power_slack();
            node.delete_candidate_sub_graph();
        }
    }

    public void releaseable_greedy_algorithm(){
        Node node = new Node();
        candidate_node_list = new ArrayList();
        while(true){
            node = node_max_releaseable_local_PS();
            if (node.releaseable_local_PS() == 0)
                break;
        }
    }

```

```

        PS += node.releaseable_local_PS();

        switch(node.get_max_releaseable_local_PS()){
            case 1: power_PS += node.get_releaseable_power_slack();
                    break;
            case 2: power_PS +=
node.releaseable_delay_oriented_power_slack_sum_fanin();
                    break;
            case 3: power_PS +=
node.releaseable_delay_oriented_power_slack_sum_fanout();
                    break;
            default: break;
        }

        candidate_node_list.add(node);
        node.arrange_releaseable_slack();
        node.update_candidate_sub_graph();
    }
}

public void releaseable_power_greedy_algorithm(){
    Node node = new Node();
    candidate_node_list = new ArrayList();
    while(true){
        node = node_max_releaseable_local_power_PS();
        if (node.releaseable_local_power_PS() == 0)
            break;
        power_PS += node.releaseable_local_power_PS();
        candidate_node_list.add(node);
        node.arrange_releaseable_power_slack();
        node.update_power_candidate_sub_graph();
    }
}

public void show_info(){
    Node node = new Node();
    for(int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        List fanin_node_list = node.get_fanin_node_list();
        List fanout_node_list = node.get_fanout_node_list();
    }
}

```



```

node.show_myself();

//System.out.println(node.get_nodename());
System.out.println("switching: " + node.get_switching());
System.out.println("area: " + node.get_area());
System.out.println("delay: " + node.get_delay());
System.out.println("power: " + node.get_power());
System.out.println("arrival_time: " + node.get_arrival_time());
System.out.println("required_time: " + node.get_required_time());
System.out.println("slack: " + node.get_slack());
System.out.println("power_slack: " + node.get_power_slack());
System.out.println("lib_level: " + node.get_lib_level());
System.out.println("releaseable_slack: " + node.get_releaseable_slack());
System.out.println("releaseable_power_slack:          " +
node.get_releaseable_power_slack());
System.out.println("in_graph: " + node.get_in_graph());
System.out.println("delta_delay: " + node.get_delta_delay());
System.out.println("delta_power: " + node.get_delta_power());
System.out.println("column: " + node.get_column());
System.out.println("row: " + node.get_row());
System.out.println();
}
}

public Dimension calculate_graph_size(){
    int x = 0;
    int y = 0;
    Node node = new Node();

    for(int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        if (x < node.get_column())
            x = node.get_column();
        if (y < node.get_row())
            y = node.get_row();
    }

    graph_size = new Dimension(x, y);
    return graph_size;
}
}

```

```

public Dimension draw_size(double zoom, double dimension_coefficient){
    //around +2 space;
    Dimension draw_size = new Dimension();
    draw_size.setSize((graph_size.getWidth() + 3) * zoom * dimension_coefficient,
        (graph_size.getHeight() + 1) * zoom * dimension_coefficient);
    return draw_size;
}

public void draw_graph(boolean selected_part_only, Graphics2D g2, double zoom, double
radius, double dimension_coefficient, Color node_color, Color line_color){
    List draw_node_list = new ArrayList();
    Node node = new Node();

    if (selected_part_only){
        for(int index = 0; index < graph_node_list.size(); index++){
            node = (Node)graph_node_list.get(index);
            if (node.get_selected())
                draw_node_list.add(node);
        }
    }
    else{
        draw_node_list = graph_node_list;
    }

    for(int index = 0; index < draw_node_list.size(); index++){
        node = (Node)draw_node_list.get(index);
        node.draw_fanin_lines(g2, zoom, radius, dimension_coefficient, line_color);
        node.draw(g2, zoom, radius, dimension_coefficient, node_color);
        if(selected_part_only)
            node.draw_fanout_lines(g2, zoom, radius, dimension_coefficient,
line_color);
    }
}

public String[] getNodeList(){
    Node node;
    String[] nodeArray = new String[graph_node_list.size()];
    for(int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        nodeArray[index] = node.get_nodename();
    }
}

```

```

    }
    return nodeArray;
}

public void setSelectedNodes(int[] selected){
    Node node;
    for(int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        node.set_selected(false);
        for(int index_inner = 0; index_inner < Array.getLength(selected); index_inner++){
            if(index == selected[index_inner]){
                node.set_selected(true);
                break;
            }
        }
    }
}

public int[] findSelectedNodes(int x, int y, double zoom, double dimension_coefficient,
double radius){
    Node node;
    int[] select = new int[1];
    select[0] = -1;
    for(int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        if (node.clickInside(x, y, zoom, dimension_coefficient, radius)){
            select[0] = index;
            break;
        }
    }
    return select;
}

public void drawCurve(Graphics2D g2, int w, int h){
    double width = w;
    double height = h;
    double border_width = width/10;
    double border_height = height/10;
    double draw_height_step = height/10*8/5;
    //draw dimension
    g2.setPaint(Color.black);

```

```

        g2.draw(new Line2D.Double(border_width, border_height, border_width,
border_height*9));
        g2.draw(new Line2D.Double(border_width, border_height*9, border_width*9,
border_height*9));
        g2.draw(new Line2D.Double(border_width, border_height, border_width*9,
border_height));
        g2.draw(new Line2D.Double(border_width*9, border_height, border_width*9,
border_height*9));
        g2.setPaint(Color.lightGray);
        g2.draw(new Line2D.Double(border_width, border_height + draw_height_step,
border_width*9, border_height + draw_height_step));
        g2.draw(new Line2D.Double(border_width, border_height + draw_height_step*2,
border_width*9, border_height + draw_height_step*2));
        g2.draw(new Line2D.Double(border_width, border_height + draw_height_step*3,
border_width*9, border_height + draw_height_step*3));
        g2.draw(new Line2D.Double(border_width, border_height + draw_height_step*4,
border_width*9, border_height + draw_height_step*4));

//draw curve
g2.setPaint(Color.blue);
double x1 = border_width;
double y1 = height- (plot_array[0]* border_height*8 + border_height);
double x_step = border_width*8/(Array.getLength(plot_array)-1);
for(int index=0; index < Array.getLength(plot_array)-1; index++){
    double x2= x_step*(index+1) + border_width;
    double y2 = height - (plot_array[index+1] * border_height*8 + border_height);
    g2.draw(new Line2D.Double(x1, y1, x2, y2));
    x1 = x2;
    y1 = y2;
}
}

public String setTextArea(int[] select){
    String textarea_text = new String("");
    Node node;
    for(int index = 0; index < graph_node_list.size(); index++){
        node = (Node)graph_node_list.get(index);
        if (node.get_selected() == true)
            textarea_text += node.show_myself_tostring();
    }
}

```

```

    if (textarea_text.length() < 2)
        textarea_text += show_graph_info();
    return textarea_text;
}

public String show_graph_info(){
    String graph_info = new String("");
    if (get_title() != null)
        graph_info += "Circuit Name: " + get_title() + "\n";
        graph_info += "Number of Gates: " + graph_node_list.size() + "\n";
        graph_info += "T0: " + (float)get_T0() + " time units \n";
        //graph_info += "T0: " + String.valueOf(get_T0()).substring(0,4 +
String.valueOf(get_T0()).indexOf(new String("."))) + " time units";
        //graph_info += "
";
        graph_info += "Power dissipation: " + (float)(get_initial_power()-get_power_PS()) + "
power units";
        //graph_info += "Power dissipation: " +
String.valueOf(get_initial_power()-get_power_PS()).substring(0,4 +
String.valueOf(get_initial_power()-get_power_PS()).indexOf(new String("."))) + " power
units";
    return graph_info;
}

public void set_title(String title){
    this.title = title;
    try {
        BufferedWriter out = new BufferedWriter(new FileWriter("System"));
        out.write("Circuit Name: " + title + "\n");
        out.close();
    }catch (IOException e) {
    }
}

public String get_title(){
    if (this.title == null){
        try {
            String str;
            BufferedReader in = new BufferedReader(new FileReader("System"));
            while((str = in.readLine()) != null) {
                if(str.startsWith("Circuit Name: ")){
                    this.title = str.substring(14);
                }
            }
        }
    }
}

```

```

        }
    }

    in.close();
} catch (IOException e) {
}

}
return this.title;
}

public void set_T0(double T0){
    this.T0 = T0;
}

public double get_T0(){
    return this.T0;
}

public double get_initial_power(){
    return initial_power;
}
}

//out2gates.java
package com.Greedy;
import java.io.*;
import java.util.*;
import java.lang.reflect.Array;

class Out2Gates{
    public static void main(String[] args){
        String out_file = "b1.out";
        Out2Gates o = new Out2Gates();
        o.read_out_write_switching(out_file);
        o.read_out_write_gate(out_file);
        //o.test1();
        //o.test2();
    }
}

```

```

public void read_out_write_gate(String out_file){
    try {
        BufferedReader in = new BufferedReader(new FileReader(out_file));
        BufferedWriter out = new BufferedWriter(new FileWriter("Gates"));
        int status = 0;//0: dull 1: work;
        String str;
        String [] gate_info;
        String gate_category;
        String gate_name;
        double gate_weight = 1;
        //double gate_switching = 1;
        double gate_area;
        double gate_delay_coefficient = 1;
        double gate_power_coefficient = 1;
        while ((str = in.readLine()) != null) {
            if(status == 0){
                if(str.equals("# of failing outputs: 0")){
                    out.write("name" + " " + "area" + " "
                        + "weight" + " " /*+ "switching" + " " */+ "delay_coefficient"
                        + " power_coefficient" + " " + "output inputs"
                        + "\n");
                    status = 1;
                }
                else{
                    continue;
                }
            }
            else if(status == 1){
                if(str.startsWith("sis> ")){
                    str = str.substring(9);
                }
                if(str.startsWith("sis> primary inputs:")){
                    out.write(str.substring(5) + "\n");
                    str = in.readLine();
                    out.write(str);
                    break;
                }
                else{
                    gate_info = str.trim().split(" ");
                    gate_category = lib_match(gate_info);
                    gate_name = gate_category + "_" + gate_info[0];
                }
            }
        }
    }
}

```

```

        gate_area = gate_area(gate_category);
        //System.out.println(gate_name + " " + gate_area + " " + gate_weight +
" " + trim_gate_info(gate_info, gate_category));
        out.write(gate_name + " " + gate_area + " " /*+ gate_switching + "
*/ + gate_weight + " " + gate_delay_coefficient + " " + gate_power_coefficient + " " +
trim_gate_info(gate_info, gate_category) + "\n");
    }
}
in.close();
out.close();
} catch (IOException e) {
}
}

public void read_out_write_switching(String out_file){
    try {
        int status = 0; //0: dull; 1: work;
        String str;
        String [] str_info;
        BufferedReader in = new BufferedReader(new FileReader(out_file));
        BufferedWriter out = new BufferedWriter(new FileWriter("Switchings"));
        while ((str = in.readLine()) != null) {
            if(status == 0){
                if(str.startsWith("sis> Node")){
                    status = 1;
                    str = str.substring(str.indexOf("Switch Prob. = ") + 15,
str.indexOf("Switch Prob. = ") + 19);
                    out.write(str + "\n");
                }
            }
            else if (status == 1){
                if (!str.startsWith("Node"))
                    break;
                str = str.substring(str.indexOf("Switch Prob. = ") + 15,
str.indexOf("Switch Prob. = ") + 19);
                out.write(str + "\n");
            }
        }
        in.close();
        out.close();
    }
}

```



```

        } catch (IOException e) {
        }
    }

public double gate_area(String gate_category) {
    double area = 0;
    if (gate_category.equals("inv_comb"))
        area = 16;
    else if (gate_category.equals("buffer_comb"))
        area = 16;
    else if (gate_category.equals("nor2_comb"))
        area = 24;
    else if (gate_category.equals("nor3_comb"))
        area = 32;
    else if (gate_category.equals("nor4_comb"))
        area = 40;
    else if (gate_category.equals("nand2_comb"))
        area = 24;
    else if (gate_category.equals("nand3_comb"))
        area = 32;
    else if (gate_category.equals("nand4_comb"))
        area = 40;
    else if (gate_category.equals("and2_comb"))
        area = 32;
    else if (gate_category.equals("and3_comb"))
        area = 40;
    else if (gate_category.equals("and4_comb"))
        area = 48;
    else if (gate_category.equals("or2_comb"))
        area = 32;
    else if (gate_category.equals("or3_comb"))
        area = 40;
    else if (gate_category.equals("or4_comb"))
        area = 48;
    else if (gate_category.equals("aoi22_comb"))
        area = 40;
    else if (gate_category.equals("aoi12_comb"))
        area = 32;
    else if (gate_category.equals("oai22_comb"))
        area = 40;
    else if (gate_category.equals("oai12_comb"))

```

```

        area = 32;
    else if (gate_category.equals("ao22_comb"))
        area = 56;
    else if (gate_category.equals("ao222_comb"))
        area = 72;
    else if (gate_category.equals("ao222_2_comb"))
        area = 72;
    else if (gate_category.equals("ao222_3_comb"))
        area = 72;
    else if (gate_category.equals("ao2222_comb"))
        area = 96;
    else if (gate_category.equals("ao33_comb"))
        area = 64;
    else if (gate_category.equals("ao33_2_comb"))
        area = 64;
    else if (gate_category.equals("ao33_3_comb"))
        area = 64;
    else if (gate_category.equals("xor_comb"))
        area = 40;
    else if (gate_category.equals("xorbar_comb"))
        area = 48;
    else if (gate_category.equals("invand_comb"))
        area = 32;
    else if (gate_category.equals("invor_comb"))
        area = 32;
    else if (gate_category.equals("mux2_comb"))
        area = 48;
    else if (gate_category.equals("const1_comb"))
        area = 8;
    else if (gate_category.equals("const0_comb"))
        area = 8;
    return area;
}

public String trim_node(String node, int head, int tail){
    String trim_node = node;
    switch(head){
        case 1: trim_node = trim_node.substring(1);
                break;
        case 2: trim_node = trim_node.substring(2);
                break;
    }
}

```

```

        case 3: trim_node = trim_node.substring(3);
                break;
        default: break;
    }
    switch(tail){
        case 1: trim_node = trim_node.substring(0,trim_node.length()-1);
                break;
        case 2: trim_node = trim_node.substring(0,trim_node.length()-2);
                break;
        case 3: trim_node = trim_node.substring(0,trim_node.length()-3);
                break;
        default: break;
    }
    return trim_node;
}

public String trim_gate_info(String [] gate_info, String gate_category){
    String trim_gate_info = gate_info[0];
    if (gate_category.equals("inv_comb"))
        trim_gate_info = trim_gate_info.concat(" " + trim_node(gate_info[2], 0, 1));
    else if (gate_category.equals("buffer_comb"))
        trim_gate_info = trim_gate_info.concat(" " + gate_info[2]);
    else if (gate_category.equals("nor2_comb"))
        trim_gate_info = trim_gate_info.concat(" " + trim_node(gate_info[2], 0, 1) + "
" + trim_node(gate_info[3], 0, 1));
    else if (gate_category.equals("nor3_comb"))
        trim_gate_info = trim_gate_info.concat(" " + trim_node(gate_info[2], 0, 1) + "
" + trim_node(gate_info[3], 0, 1) + " " + trim_node(gate_info[4], 0, 1));
    else if (gate_category.equals("nor4_comb"))
        trim_gate_info = trim_gate_info.concat(" " + trim_node(gate_info[2], 0, 1) + "
" + trim_node(gate_info[3], 0, 1) + " " + trim_node(gate_info[4], 0, 1) + " " +
trim_node(gate_info[5], 0, 1));
    else if (gate_category.equals("nand2_comb"))
        trim_gate_info = trim_gate_info.concat(" " + trim_node(gate_info[2], 0, 1) + "
" + trim_node(gate_info[4], 0, 1));
    else if (gate_category.equals("nand3_comb"))
        trim_gate_info = trim_gate_info.concat(" " + trim_node(gate_info[2], 0, 1) + "
" + trim_node(gate_info[4], 0, 1) + " " + trim_node(gate_info[6], 0, 1));
    else if (gate_category.equals("nand4_comb"))
        trim_gate_info = trim_gate_info.concat(" " + trim_node(gate_info[2], 0, 1) + "
" + trim_node(gate_info[4], 0, 1) + " " + trim_node(gate_info[6], 0, 1) + " " +

```

```

trim_node(gate_info[8], 0, 1));
    else if (gate_category.equals("and2_comb"))
        trim_gate_info = trim_gate_info.concat(" " + gate_info[2] + " " + gate_info[3]);
    else if (gate_category.equals("and3_comb"))
        trim_gate_info = trim_gate_info.concat(" " + gate_info[2] + " " + gate_info[3]
+ " " + gate_info[4]);
    else if (gate_category.equals("and4_comb"))
        trim_gate_info = trim_gate_info.concat(" " + gate_info[2] + " " + gate_info[3]
+ " " + gate_info[4] + " " + gate_info[5]);
    else if (gate_category.equals("or2_comb"))
        trim_gate_info = trim_gate_info.concat(" " + gate_info[2] + " " + gate_info[4]);
    else if (gate_category.equals("or3_comb"))
        trim_gate_info = trim_gate_info.concat(" " + gate_info[2] + " " + gate_info[4]
+ " " + gate_info[6]);
    else if (gate_category.equals("or4_comb"))
        trim_gate_info = trim_gate_info.concat(" " + gate_info[2] + " " + gate_info[4]
+ " " + gate_info[6] + " " + gate_info[8]);
    else if (gate_category.equals("aoi22_comb"))
        trim_gate_info = trim_gate_info.concat(" " + trim_node(gate_info[2], 1, 1) + "
" + trim_node(gate_info[4], 0, 2) + " " + trim_node(gate_info[5], 1, 1) + " " +
trim_node(gate_info[7], 0, 2));
    else if (gate_category.equals("aoi12_comb"))
        trim_gate_info = trim_gate_info.concat(" " + trim_node(gate_info[2], 0, 1) + "
" + trim_node(gate_info[3], 1, 1) + " " + trim_node(gate_info[5], 0, 2));
    else if (gate_category.equals("oai22_comb"))
        trim_gate_info = trim_gate_info.concat(" " + trim_node(gate_info[2], 0, 1) + "
" + trim_node(gate_info[3], 0, 1) + " " + trim_node(gate_info[5], 0, 1) + " " +
trim_node(gate_info[6], 0, 1));
    else if (gate_category.equals("oai12_comb"))
        trim_gate_info = trim_gate_info.concat(" " + trim_node(gate_info[2], 0, 1) + "
" + trim_node(gate_info[3], 0, 1) + " " + trim_node(gate_info[5], 0, 1));
    else if (gate_category.equals("ao22_comb"))
        trim_gate_info = trim_gate_info.concat(" " + gate_info[2] + " " + gate_info[3]
+ " " + gate_info[5] + " " + gate_info[6]);
    else if (gate_category.equals("ao222_comb"))
        trim_gate_info = trim_gate_info.concat(" " + gate_info[2] + " " + gate_info[3]
+ " " + gate_info[5] + " " + gate_info[6] + " " + gate_info[8] + " " + gate_info[9]);
    else if (gate_category.equals("ao222_2_comb"))
        trim_gate_info = trim_gate_info.concat(" " + gate_info[2] + " " +
trim_node(gate_info[3], 1, 0) + " " + trim_node(gate_info[5], 0, 1) + " " + gate_info[7] + "
" + gate_info[8]);

```

```

        else if (gate_category.equals("ao222_3_comb"))
            trim_gate_info = trim_gate_info.concat(" " + gate_info[2] + " " +
trim_node(gate_info[3], 1, 0) + " " + gate_info[5] + " " + trim_node(gate_info[7], 0, 1));
        else if (gate_category.equals("ao2222_comb"))
            trim_gate_info = trim_gate_info.concat(" " + gate_info[2] + " " + gate_info[3]
+ " " + gate_info[5] + " " + gate_info[6] + " " + gate_info[8] + " " + gate_info[9] + " " +
gate_info[11] + " " + gate_info[12]);
        else if (gate_category.equals("ao33_comb"))
            trim_gate_info = trim_gate_info.concat(" " + gate_info[2] + " " + gate_info[3]
+ " " + gate_info[4] + " " + gate_info[6] + " " + gate_info[7] + " " + gate_info[8]);
        else if (gate_category.equals("ao33_2_comb"))
            trim_gate_info = trim_gate_info.concat(" " + gate_info[2] + " " +
trim_node(gate_info[3], 1, 0) + " " + gate_info[4] + " " + gate_info[6] + " " +
trim_node(gate_info[7], 0, 1));
        else if (gate_category.equals("ao33_3_comb"))
            trim_gate_info = trim_gate_info.concat(" " + gate_info[2] + " " + gate_info[3]
+ " " + trim_node(gate_info[4], 1, 0) + " " + trim_node(gate_info[6], 0, 1));
        //else if (gate_category.equals("xor_comb"))
        //
        //else if (gate_category.equals("xorbar_comb"))
        //
        else if (gate_category.equals("invand_comb"))
            trim_gate_info = trim_gate_info.concat(" " + trim_node(gate_info[2], 0, 1) + "
" + gate_info[3]);
        else if (gate_category.equals("invor_comb"))
            trim_gate_info = trim_gate_info.concat(" " + trim_node(gate_info[2], 0, 1) + "
" + gate_info[4]);
        return trim_gate_info;
    }

    public String lib_match(String [] gate_info){
        String gate_category = "--unknown--";
        int gate_info_length = Array.getLength(gate_info);
        switch(gate_info_length){
            case 3: if (gate_info[2].lastIndexOf("'") == gate_info[2].length() - 1)
                gate_category = "inv_comb";
                else
                    gate_category = "buffer_comb";
                break;
            case 4: if ((gate_info[2].lastIndexOf("'") == gate_info[2].length() - 1)
&& (gate_info[3].lastIndexOf("'") == gate_info[3].length() - 1))

```

```

        gate_category = "nor2_comb";
    else if ((gate_info[2].lastIndexOf("'") == gate_info[2].length() - 1)
        && (gate_info[3].lastIndexOf("'") != gate_info[3].length() - 1))
        gate_category = "invand_comb";
    else if ((gate_info[2].lastIndexOf("'") != gate_info[2].length() - 1)
        && (gate_info[3].lastIndexOf("'") != gate_info[3].length() - 1))
        gate_category = "and2_comb";
    break;
case 5: if ((gate_info[2].lastIndexOf("'") == gate_info[2].length() - 1)
        && (gate_info[3].lastIndexOf("'") == gate_info[3].length() - 1)
        && (gate_info[4].lastIndexOf("'") == gate_info[4].length() - 1))
        gate_category = "nor3_comb";
    else if ((gate_info[2].lastIndexOf("'") == gate_info[2].length() - 1)
        && (gate_info[3].equals("+"))
        && (gate_info[4].lastIndexOf("'") == gate_info[4].length() - 1))
        gate_category = "nand2_comb";
    else if ((gate_info[2].lastIndexOf("'") != gate_info[2].length() - 1)
        && (gate_info[3].equals("+"))
        && (gate_info[4].lastIndexOf("'") != gate_info[4].length() - 1))
        gate_category = "or2_comb";
    else if ((gate_info[2].lastIndexOf("'") == gate_info[2].length() - 1)
        && (gate_info[3].equals("+"))
        && (gate_info[4].lastIndexOf("'") != gate_info[4].length() - 1))
        gate_category = "invor_comb";
    else
        gate_category = "and3_comb";
    break;
case 6: if ((gate_info[2].lastIndexOf("'") == gate_info[2].length() - 1)
        && (gate_info[3].lastIndexOf("'") == gate_info[3].length() - 1)
        && (gate_info[4].lastIndexOf("'") == gate_info[4].length() - 1)
        && (gate_info[5].lastIndexOf("'") == gate_info[5].length() - 1))
        gate_category = "nor4_comb";
    else if ((gate_info[2].lastIndexOf("'") == gate_info[2].length() - 1)
        && (gate_info[3].lastIndexOf("'") == gate_info[3].length() - 1)
        && (gate_info[4].equals("+"))
        && (gate_info[5].lastIndexOf("'") == gate_info[5].length() - 1))
        gate_category = "aoi12_comb";
    else if ((gate_info[2].lastIndexOf("'") == gate_info[2].length() - 1)
        && (gate_info[3].lastIndexOf("'") == gate_info[3].length() - 1)
        && (gate_info[4].equals("+"))
        && (gate_info[5].lastIndexOf("'") == gate_info[5].length() - 1))

```

```

        gate_category = "oai12_comb";
    else
        gate_category = "and4_comb";
    break;
case 7: if ((gate_info[2].lastIndexOf("'") == gate_info[2].length() - 1)
        && (gate_info[3].equals("+"))
        && (gate_info[4].lastIndexOf("'") == gate_info[4].length() - 1)
        && (gate_info[5].equals("+"))
        && (gate_info[6].lastIndexOf("'") == gate_info[6].length() - 1))
        gate_category = "nand3_comb";
    else if ((gate_info[2].lastIndexOf("'") == gate_info[2].length() - 1)
        && (gate_info[3].lastIndexOf("'") == gate_info[3].length() - 1)
        && (gate_info[4].equals("+"))
        && (gate_info[5].lastIndexOf("'") == gate_info[5].length() - 1)
        && (gate_info[6].lastIndexOf("'") == gate_info[6].length() - 1))
        gate_category = "oai22_comb";
    else if ((gate_info[2].lastIndexOf("'") != gate_info[2].length() - 1)
        && (gate_info[3].equals("+"))
        && (gate_info[4].lastIndexOf("'") != gate_info[4].length() - 1)
        && (gate_info[5].equals("+"))
        && (gate_info[6].lastIndexOf("'") != gate_info[6].length() - 1))
        gate_category = "or3_comb";
    else if ((gate_info[2].lastIndexOf("'") != gate_info[2].length() - 1)
        && (gate_info[3].lastIndexOf("'") != gate_info[3].length() - 1)
        && (gate_info[4].equals("+"))
        && (gate_info[5].lastIndexOf("'") != gate_info[5].length() - 1)
        && (gate_info[6].lastIndexOf("'") != gate_info[6].length() - 1))
        gate_category = "ao22_comb";
    else if (gate_info[5].equals("+"))
        gate_category = "ao33_3_comb";
    break;
case 8: if ((gate_info[2].lastIndexOf("'") == gate_info[2].length() - 1)
        && (gate_info[3].equals("+"))
        && (gate_info[4].lastIndexOf("'") == gate_info[4].length() - 1)
        && (gate_info[5].lastIndexOf("'") == gate_info[5].length() - 1)
        && (gate_info[6].equals("+"))
        && (gate_info[7].lastIndexOf("'") == gate_info[7].length() - 1))
        gate_category = "aoi22_comb";
    else if (gate_info[5].equals("+"))
        gate_category = "ao33_2_comb";
    else if (gate_info[4].equals("+"))

```

```

        gate_category = "ao222_3_comb";
        break;
    case 9: if ((gate_info[2].lastIndexOf("'") == gate_info[2].length() - 1)
        && (gate_info[3].equals("+"))
        && (gate_info[4].lastIndexOf("'") == gate_info[4].length() - 1)
        && (gate_info[5].equals("+"))
        && (gate_info[6].lastIndexOf("'") == gate_info[6].length() - 1)
        && (gate_info[7].equals("+"))
        && (gate_info[8].lastIndexOf("'") == gate_info[8].length() - 1))
        gate_category = "nand4_comb";
    else if ((gate_info[2].lastIndexOf("'") != gate_info[2].length() - 1)
        && (gate_info[3].equals("+"))
        && (gate_info[4].lastIndexOf("'") != gate_info[4].length() - 1)
        && (gate_info[5].equals("+"))
        && (gate_info[6].lastIndexOf("'") != gate_info[6].length() - 1)
        && (gate_info[7].equals("+"))
        && (gate_info[8].lastIndexOf("'") != gate_info[8].length() - 1))
        gate_category = "or4_comb";
    else if (gate_info[5].equals("+"))
        gate_category = "ao33_comb";
    else if (gate_info[4].equals("+"))
        gate_category = "ao222_2_comb";
        break;
    case 10: gate_category = "ao222_comb";
        break;
    case 13: gate_category = "ao2222_comb";
        break;
    default: gate_category = "--unknown--";
        break;
    }
    return gate_category;
}

public void test1(){
    String test = "abc";
    System.out.println(test.lastIndexOf("'"));
    System.out.println(test.length());
}

public void test2(String [] gate_info){
    int gate_info_length = Array.getLength(gate_info);

```



```

        System.out.println(gate_info_length);
        for(int i = 0; i < gate_info_length; i++){
            System.out.print(gate_info[i] + " ");
        }
        System.out.println();
    }
}

```

```

//Node.java
package com.Greedy;
import java.util.*;
import java.awt.Dimension;
import java.awt.Graphics2D;
import java.awt.Color;
import java.awt.BasicStroke;
import java.awt.geom.*;

class Node{
    private List fanin_node_list;
    private List fanout_node_list;
    private String nodename;
    private String category;
    private String [] gate_info;
    private double weight = 1;
    private double area = 0;
    private double delay_coefficient = 1;
    private double power_coefficient = 1;
    private double switching = 0;

    private boolean in_graph = true;
    private double delay;
    private double arrival_time;
    private double required_time;
    private double slack;
    private double delta_delay = 0;
    private int max_local_PS = 1; //1: self; 2: fanin; 3: fanout;

    private double power;
    private double power_slack;
    private double delta_power = 0;

```

```

private int max_local_power_PS = 1; //1: self; 2: fanin; 3: fanout;

private double releaseable_slack;
private double releaseable_power_slack;
private int max_releaseable_local_PS = 1;
private int max_releaseable_local_power_PS = 1;
private int releaseable_lib_level = 0;
private int lib_level = 1; //1: full_size; 2: half_size; 3: quarter_size;

//draw
private int column = 0;
private int row = 0;
private boolean selected = false;

public void set_column(int column){
    this.column = column;
}

public int get_column(){
    return column;
}

public void set_row(int row){
    this.row = row;
}

public int get_row(){
    return row;
}

public void set_selected(boolean selected){
    this.selected = selected;
}

public boolean get_selected(){
    return selected;
}

public void set_gate_info(String [] gate_info){
    this.gate_info = gate_info;
}

```

```

public String [] get_gate_info(){
    return gate_info;
}

public void set_category(String category){
    this.category = category;
}

public String get_category(){
    return category;
}

public void set_weight(double weight){
    this.weight = weight;
}

public double get_weight(){
    return weight;
}

public void set_area(double area){
    this.area = area;
}

public double get_area(){
    return area;
}

public void set_switching(double switching){
    this.switching = switching;
}

public double get_switching(){
    return switching;
}

public void set_delay_coefficient(double delay_coefficient){
    this.delay_coefficient = delay_coefficient;
}

```

```

public double get_delay_coefficient(){
    return delay_coefficient;
}

public void set_power_coefficient(double power_coefficient){
    this.power_coefficient = power_coefficient;
}

public double get_power_coefficient(){
    return power_coefficient;
}

public void set_delay(double delay){
    this.delay = delay;
}

public double get_delay(){
    return delay;
}

public void set_power(double power){
    this.power = power;
}

public double get_power(){
    return power;
}

public void set_delta_delay(double delta_delay){
    this.delta_delay = delta_delay;
}

public double get_delta_delay(){
    return delta_delay;
}

public void set_arrival_time(double arrival_time){
    this.arrival_time = arrival_time;
}

public double get_arrival_time(){

```

```

        return arrival_time;
    }

    public void set_required_time(double required_time){
        this.required_time = required_time;
    }

    public double get_required_time(){
        return required_time;
    }

    public void set_slack(double slack){
        this.slack = slack;
    }

    public double get_slack(){
        return slack;
    }

    public void set_power_slack(double power_slack){
        this.power_slack = power_slack;
    }

    public double get_power_slack(){
        return power_slack;
    }

    public void set_delta_power(double delta_power){
        this.delta_power = delta_power;
    }

    public double get_delta_power(){
        return delta_power;
    }

    public List get_fanin_node_list(){
        return fanin_node_list;
    }

    public void new_fanin_node_list(){
        fanin_node_list = new ArrayList();
    }

```

```

}

public List get_fanout_node_list(){
    return fanout_node_list;
}

public void new_fanout_node_list(){
    fanout_node_list = new ArrayList();
}

public void set_nodename(String nodename){
    this.nodename = nodename;
}

public String get_nodename(){
    return this.nodename;
}

public void set_in_graph(boolean in_graph){
    this.in_graph = in_graph;
}

public boolean get_in_graph(){
    return in_graph;
}

public int get_max_local_PS(){
    return max_local_PS;
}

public int get_max_local_power_PS(){
    return max_local_power_PS;
}

public int get_max_releaseable_local_PS(){
    return max_releaseable_local_PS;
}

public int get_max_releaseable_local_power_PS(){
    return max_releaseable_local_power_PS;
}

```

```

public void set_lib_level(int lib_level){
    this.lib_level = lib_level;
}

public int get_lib_level(){
    return lib_level;
}

public void set_releaseable_lib_level(int releaseable_lib_level){
    this.releaseable_lib_level = releaseable_lib_level;
}

public int get_releaseable_lib_level(){
    return releaseable_lib_level;
}

public void set_releaseable_slack(double releaseable_slack){
    this.releaseable_slack = releaseable_slack;
}

public double get_releaseable_slack(){
    return releaseable_slack;
}

public void set_releaseable_power_slack(double releaseable_power_slack){
    this.releaseable_power_slack = releaseable_power_slack;
}

public double get_releaseable_power_slack(){
    return releaseable_power_slack;
}

public boolean check_reconvergence(Node check_node, List node_list){
    Node node = new Node();
    if (check_node.get_fanout_node_list() != null){
        for(int index = 0; index < check_node.get_fanout_node_list().size(); index++){
            node = (Node)check_node.get_fanout_node_list().get(index);
            if (node_list.contains(node)){
                if(check_node.get_slack() >= node.get_slack())
                    return true;
            }
        }
    }
}

```

```

        else
            return false;
    }
}
}
if (check_node.get_fanin_node_list() != null){
    for(int index = 0; index < check_node.get_fanin_node_list().size(); index++){
        node = (Node)check_node.get_fanin_node_list().get(index);
        if (node_list.contains(node)){
            if(check_node.get_slack() > node.get_slack())
                return true;
            else
                return false;
        }
    }
}
return true;
}

public boolean check_power_reconvergence(Node check_node, List node_list){
    Node node = new Node();
    if (check_node.get_fanout_node_list() != null){
        for(int index = 0; index < check_node.get_fanout_node_list().size(); index++){
            node = (Node)check_node.get_fanout_node_list().get(index);
            if (node_list.contains(node)){
                if(check_node.get_power_slack() >= node.get_power_slack())
                    return true;
                else
                    return false;
            }
        }
    }
    if (check_node.get_fanin_node_list() != null){
        for(int index = 0; index < check_node.get_fanin_node_list().size(); index++){
            node = (Node)check_node.get_fanin_node_list().get(index);
            if (node_list.contains(node)){
                if(check_node.get_power_slack() > node.get_power_slack())
                    return true;
                else
                    return false;
            }
        }
    }
}

```



```

    }
}
return true;
}

public boolean check_releaseable_reconvergence(Node check_node, List node_list){
    Node node = new Node();
    if (check_node.get_fanout_node_list() != null){
        for(int index = 0; index < check_node.get_fanout_node_list().size(); index++){
            node = (Node)check_node.get_fanout_node_list().get(index);
            if (node_list.contains(node)){
                if(check_node.get_releaseable_slack() >=
node.get_releaseable_slack())
                    return true;
                else
                    return false;
            }
        }
    }
    if (check_node.get_fanin_node_list() != null){
        for(int index = 0; index < check_node.get_fanin_node_list().size(); index++){
            node = (Node)check_node.get_fanin_node_list().get(index);
            if (node_list.contains(node)){
                if(check_node.get_releaseable_slack() > node.get_releaseable_slack())
                    return true;
                else
                    return false;
            }
        }
    }
    return true;
}

public boolean check_releaseable_power_reconvergence(Node check_node, List node_list){
    Node node = new Node();
    if (check_node.get_fanout_node_list() != null){
        for(int index = 0; index < check_node.get_fanout_node_list().size(); index++){
            node = (Node)check_node.get_fanout_node_list().get(index);
            if (node_list.contains(node)){
                if(check_node.get_releaseable_power_slack() >=
node.get_releaseable_power_slack())

```

```

        return true;
    else
        return false;
    }
}
}
if (check_node.get_fanin_node_list() != null){
    for(int index = 0; index < check_node.get_fanin_node_list().size(); index++){
        node = (Node)check_node.get_fanin_node_list().get(index);
        if (node_list.contains(node)){
            if(check_node.get_releaseable_power_slack() >
node.get_releaseable_power_slack())
                return true;
            else
                return false;
        }
    }
}
return true;
}

public double slack_sum_fanin(){
    Node node;
    double slack_sum_fanin = 0;
    if (fanin_node_list != null){
        for(int index = 0; index < fanin_node_list.size(); index++){
            node = (Node)fanin_node_list.get(index);
            if(node.get_in_graph()){
                if (check_reconvergence(node, fanin_node_list))
                    slack_sum_fanin += node.get_slack();
            }
        }
    }
    return slack_sum_fanin;
}

public double slack_sum_fanout(){
    Node node;
    double slack_sum_fanout = 0;
    if (fanout_node_list != null){
        for(int index = 0; index < fanout_node_list.size(); index++){

```

```

        node = (Node)fanout_node_list.get(index);
        if(node.get_in_graph())
            if (check_reconvergence(node, fanout_node_list))
                slack_sum_fanout += node.get_slack();
    }
}
return slack_sum_fanout;
}

public double local_PS(){
    double local_PS = this.get_slack();
    if (slack_sum_fanin() > local_PS){
        local_PS = slack_sum_fanin();
        max_local_PS = 2;
    }
    if (slack_sum_fanout() > local_PS){
        local_PS = slack_sum_fanout();
        max_local_PS = 3;
    }
    return local_PS;
}

public void arrange_slack(){
    Node node = new Node();
    switch(max_local_PS){
        case 1: this.set_delta_delay(this.get_slack());
                break;
        case 2: for(int index = 0; index < fanin_node_list.size(); index++){
                    node = (Node)fanin_node_list.get(index);
                    if(node.get_in_graph())
                        if (check_reconvergence(node, fanin_node_list))
                            node.set_delta_delay(node.get_slack());
                }
                break;
        case 3: for(int index = 0; index < fanout_node_list.size(); index++){
                    node = (Node)fanout_node_list.get(index);
                    if(node.get_in_graph())
                        if (check_reconvergence(node, fanout_node_list))
                            node.set_delta_delay(node.get_slack());
                }
                break;
    }
}

```

```

        default: break;
    }
}

public double releaseable_slack_sum_fanin(){
    Node node;
    double releaseable_slack_sum_fanin = 0;
    if (fanin_node_list != null){
        for(int index = 0; index < fanin_node_list.size(); index++){
            node = (Node)fanin_node_list.get(index);
            if(node.get_in_graph()){
                if (check_releaseable_reconvergence(node, fanin_node_list))
                    releaseable_slack_sum_fanin += node.get_releaseable_slack();
            }
        }
    }
    return releaseable_slack_sum_fanin;
}

public double releaseable_slack_sum_fanout(){
    Node node;
    double releaseable_slack_sum_fanout = 0;
    if (fanout_node_list != null){
        for(int index = 0; index < fanout_node_list.size(); index++){
            node = (Node)fanout_node_list.get(index);
            if(node.get_in_graph())
                if (check_releaseable_reconvergence(node, fanout_node_list))
                    releaseable_slack_sum_fanout += node.get_releaseable_slack();
        }
    }
    return releaseable_slack_sum_fanout;
}

public double releaseable_local_PS(){
    double releaseable_local_PS = this.get_releaseable_slack();
    max_releaseable_local_PS = 1;
    if (releaseable_slack_sum_fanin() > releaseable_local_PS){
        releaseable_local_PS = releaseable_slack_sum_fanin();
        max_releaseable_local_PS = 2;
    }
    if (releaseable_slack_sum_fanout() > releaseable_local_PS){

```

```

        releaseable_local_PS = releaseable_slack_sum_fanout();
        max_releaseable_local_PS = 3;
    }
    return releaseable_local_PS;
}

public void arrange_releaseable_slack(){
    Node node = new Node();
    switch(max_releaseable_local_PS){
        case 1: this.set_delta_power(this.get_releaseable_power_slack());
                this.set_delta_delay(this.get_releaseable_slack());
                this.set_lib_level(this.get_releaseable_lib_level());
                break;
        case 2: for(int index = 0; index < fanin_node_list.size(); index++){
                    node = (Node)fanin_node_list.get(index);
                    if(node.get_in_graph())
                        if (check_releaseable_reconvergence(node, fanin_node_list)){
                            node.set_lib_level(node.get_releaseable_lib_level());
                            node.set_delta_delay(node.get_releaseable_slack());
                        }
                }
                break;
        case 3: for(int index = 0; index < fanout_node_list.size(); index++){
                    node = (Node)fanout_node_list.get(index);
                    if(node.get_in_graph())
                        if (check_releaseable_reconvergence(node, fanout_node_list)){
                            node.set_lib_level(node.get_releaseable_lib_level());
                            node.set_delta_delay(node.get_releaseable_slack());
                        }
                }
                break;
        default: break;
    }
}

public double power_slack_sum_fanin(){
    Node node;

```

```

double power_slack_sum_fanin = 0;
if (fanin_node_list != null){
    for(int index = 0; index < fanin_node_list.size(); index++){
        node = (Node)fanin_node_list.get(index);
        if(node.get_in_graph())
            if (check_power_reconvergence(node, fanin_node_list))
                power_slack_sum_fanin += node.get_power_slack();
    }
}
return power_slack_sum_fanin;
}

public double power_slack_sum_fanout(){
    Node node;
    double power_slack_sum_fanout = 0;
    if (fanout_node_list != null){
        for(int index = 0; index < fanout_node_list.size(); index++){
            node = (Node)fanout_node_list.get(index);
            if(node.get_in_graph())
                if (check_power_reconvergence(node, fanout_node_list))
                    power_slack_sum_fanout += node.get_power_slack();
        }
    }
    return power_slack_sum_fanout;
}

public double local_power_PS(){
    double local_power_PS = this.get_power_slack();
    if (power_slack_sum_fanin() > local_power_PS){
        local_power_PS = power_slack_sum_fanin();
        max_local_power_PS = 2;
    }
    if (power_slack_sum_fanout() > local_power_PS){
        local_power_PS = power_slack_sum_fanout();
        max_local_power_PS = 3;
    }
    return local_power_PS;
}

public void arrange_power_slack(){
    Node node = new Node();

```

```

switch(max_local_power_PS){
    case 1: this.set_delta_power(this.get_power_slack());
           this.set_delta_delay(this.get_slack());
           break;
    case 2: for(int index = 0; index < fanin_node_list.size(); index++){
           node = (Node)fanin_node_list.get(index);
           if(node.get_in_graph()){
               if (check_power_reconvergence(node, fanin_node_list)){
                   node.set_delta_power(node.get_power_slack());
                   node.set_delta_delay(node.get_slack());
               }
           }
           break;
    case 3: for(int index = 0; index < fanout_node_list.size(); index++){
           node = (Node)fanout_node_list.get(index);
           if(node.get_in_graph()){
               if (check_power_reconvergence(node, fanout_node_list)){
                   node.set_delta_power(node.get_power_slack());
                   node.set_delta_delay(node.get_slack());
               }
           }
           break;
    default: break;
}

}

public double releaseable_power_slack_sum_fanin(){
    Node node;
    double releaseable_power_slack_sum_fanin = 0;
    if (fanin_node_list != null){
        for(int index = 0; index < fanin_node_list.size(); index++){
            node = (Node)fanin_node_list.get(index);
            if(node.get_in_graph())
                if (check_releaseable_power_reconvergence(node, fanin_node_list))
                    releaseable_power_slack_sum_fanin +=
node.get_releaseable_power_slack();
        }
    }
    return releaseable_power_slack_sum_fanin;
}

```

```

}

public double releaseable_power_slack_sum_fanout(){
    Node node;
    double releaseable_power_slack_sum_fanout = 0;
    if (fanout_node_list != null){
        for(int index = 0; index < fanout_node_list.size(); index++){
            node = (Node)fanout_node_list.get(index);
            if(node.get_in_graph())
                if (check_releaseable_reconvergence(node, fanout_node_list))
                    releaseable_power_slack_sum_fanout +=
node.get_releaseable_power_slack();
        }
    }
    return releaseable_power_slack_sum_fanout;
}

public double releaseable_delay_oriented_power_slack_sum_fanin(){
    Node node;
    double releaseable_power_slack_sum_fanin = 0;
    if (fanin_node_list != null){
        for(int index = 0; index < fanin_node_list.size(); index++){
            node = (Node)fanin_node_list.get(index);
            if(node.get_in_graph())
                if (check_releaseable_reconvergence(node, fanin_node_list))
                    releaseable_power_slack_sum_fanin +=
node.get_releaseable_power_slack();
        }
    }
    return releaseable_power_slack_sum_fanin;
}

public double releaseable_delay_oriented_power_slack_sum_fanout(){
    Node node;
    double releaseable_power_slack_sum_fanout = 0;
    if (fanout_node_list != null){
        for(int index = 0; index < fanout_node_list.size(); index++){
            node = (Node)fanout_node_list.get(index);
            if(node.get_in_graph())
                if (check_releaseable_reconvergence(node, fanout_node_list))
                    releaseable_power_slack_sum_fanout +=

```



```

node.get_releaseable_power_slack();
    }
}
return releaseable_power_slack_sum_fanout;
}

public double releaseable_local_power_PS(){
double releaseable_local_power_PS = this.get_releaseable_power_slack();
    max_releaseable_local_power_PS = 1;
if (releaseable_power_slack_sum_fanin() > releaseable_local_power_PS){
    releaseable_local_power_PS = releaseable_power_slack_sum_fanin();
    max_releaseable_local_power_PS = 2;
}
if (releaseable_power_slack_sum_fanout() > releaseable_local_power_PS){
    releaseable_local_power_PS = releaseable_power_slack_sum_fanout();
    max_releaseable_local_power_PS = 3;
}
return releaseable_local_power_PS;
}

public void arrange_releaseable_power_slack(){
    Node node = new Node();
    switch(max_releaseable_local_power_PS){
        case 1: this.set_delta_power(this.get_releaseable_power_slack());
                this.set_delta_delay(this.get_releaseable_slack());
                this.set_lib_level(this.get_releaseable_lib_level());
                break;
        case 2: for(int index = 0; index < fanin_node_list.size(); index++){
                    node = (Node)fanin_node_list.get(index);
                    if(node.get_in_graph()){
                        if
fanin_node_list))){
                            (check_releaseable_power_reconvergence(node,
                                node.set_delta_power(node.get_releaseable_power_slack());
                                    node.set_delta_delay(node.get_releaseable_slack());
                                        node.set_lib_level(node.get_releaseable_lib_level());
                                            }
                                                }
                                                    }
                                                        break;
        case 3: for(int index = 0; index < fanout_node_list.size(); index++){

```

```

        node = (Node) fanout_node_list.get(index);
        if(node.get_in_graph()){
            if (check_power_reconvergence(node, fanout_node_list)){

node.set_delta_power(node.get_releaseable_power_slack());
                node.set_delta_delay(node.get_releaseable_slack());
                node.set_lib_level(node.get_releaseable_lib_level());
            }
        }
    }
    break;
    default: break;
}
}

public void delete_transitive_fanins(){
    Node node;
    if (fanin_node_list != null){
        for(int index = 0; index < fanin_node_list.size(); index++){
            node = (Node) fanin_node_list.get(index);
            node.set_in_graph(false);
            node.delete_transitive_fanins();
        }
    }
}

public void delete_transitive_fanouts(){
    Node node;
    if (fanout_node_list != null){
        for(int index = 0; index < fanout_node_list.size(); index++){
            node = (Node) fanout_node_list.get(index);
            node.set_in_graph(false);
            node.delete_transitive_fanouts();
        }
    }
}

public void delete_sub_graph(){
    delete_transitive_fanins();
    delete_transitive_fanouts();
    set_in_graph(false);
}

```

```

}

public void delete_candidate_sub_graph(){
    Node node = new Node();
    switch(max_local_PS){
        case 1: this.delete_sub_graph();
                break;
        case 2: for(int index = 0; index < fanin_node_list.size(); index++){
                    node = (Node)fanin_node_list.get(index);
                    node.delete_sub_graph();
                }
                break;
        case 3: for(int index = 0; index < fanout_node_list.size(); index++){
                    node = (Node)fanout_node_list.get(index);
                    node.delete_sub_graph();
                }
                break;
        default: break;
    }
}

public void update_self(){
    arrival_time = arrival_time + delta_delay;
    area = (delay / (delay + delta_delay)) * area;
    delay = delay + delta_delay;
    if (delta_delay > 0)
        lib_level = releaseable_lib_level;
    calculate_slack();
    calculate_power();
    calculate_power_slack();
    calculate_releaseable_slack();
    calculate_releaseable_power_slack();
}

public void update_transitive_fanins(){
    Node node;
    if (fanin_node_list != null){
        for(int index = 0; index < fanin_node_list.size(); index++){
            node = (Node)fanin_node_list.get(index);
            if(node.get_required_time() > required_time - delay){

```

```

        node.set_required_time(required_time - delay);
        node.calculate_slack();
        node.calculate_power();
        node.calculate_power_slack();
        node.calculate_releaseable_slack();
        node.calculate_releaseable_power_slack();
    }
    node.update_transitive_fanins();
}
}
}

public void update_transitive_fanouts(){
    Node node;
    if (fanout_node_list != null){
        for(int index = 0; index < fanout_node_list.size(); index++){
            node = (Node)fanout_node_list.get(index);
            if(node.get_arrival_time() < arrival_time + node.get_delay()){
                node.set_arrival_time(arrival_time + node.get_delay());
                node.calculate_slack();
                node.calculate_power();
                node.calculate_power_slack();
                node.calculate_releaseable_slack();
                node.calculate_releaseable_power_slack();
            }
            node.update_transitive_fanouts();
        }
    }
}

public void update_sub_graph(){
    update_self();
    update_transitive_fanins();
    update_transitive_fanouts();
}

public void update_candidate_sub_graph(){
    Node node = new Node();
    switch(max_releaseable_local_PS){
        case 1: this.update_sub_graph();
            break;
    }
}

```

```

        case 2: for(int index = 0; index < fanin_node_list.size(); index++){
                node = (Node) fanin_node_list.get(index);
                node.update_sub_graph();
            }
            break;
        case 3: for(int index = 0; index < fanout_node_list.size(); index++){
                node = (Node) fanout_node_list.get(index);
                node.update_sub_graph();
            }
            break;
        default: break;
    }
}

public void update_power_candidate_sub_graph(){
    Node node = new Node();
    switch(max_releaseable_local_power_PS){
        case 1: this.update_sub_graph();
            break;
        case 2: for(int index = 0; index < fanin_node_list.size(); index++){
                node = (Node) fanin_node_list.get(index);
                node.update_sub_graph();
            }
            break;
        case 3: for(int index = 0; index < fanout_node_list.size(); index++){
                node = (Node) fanout_node_list.get(index);
                node.update_sub_graph();
            }
            break;
        default: break;
    }
}

public double calculate_delay(){
    int fanout = 0;
    delay = 0;
    if (fanout_node_list != null)
        fanout = fanout_node_list.size();
    delay = delay_coefficient * fanout / area;
    return delay;
}

```

```

public double calculate_power(){
    power = 0;
    if (fanout_node_list != null)
        power = power_coefficient * area * switching;
    return power;
}

public double calculate_releaseable_slack(){
    releaseable_slack = 0;
    releaseable_lib_level = lib_level;
    switch(lib_level){
        case 1: if (delay * 3 <= slack){
                releaseable_lib_level = 3;
                releaseable_slack = delay * 3;
            }
            else if (delay <= slack){
                releaseable_lib_level = 2;
                releaseable_slack = delay;
            }
            break;
        case 2: if (delay <= slack){
                releaseable_lib_level = 3;
                releaseable_slack = delay;
            }
            break;
        case 3: break;
        default: break;
    }
    return releaseable_slack;
}

public double calculate_releaseable_power_slack(){
    if (delay == 0)
        releaseable_power_slack = 0;
    else
        releaseable_power_slack = power * delay * (1/delay - 1/(delay +
releaseable_slack));
    return releaseable_power_slack;
}

```

```

public int calculate_column(){
    Node node = new Node();
    column = 1;

    if (fanin_node_list == null)
        return column;
    for (int index = 0; index < fanin_node_list.size(); index++){
        node = (Node)fanin_node_list.get(index);
        if (column <= node.calculate_column())
            column = node.get_column() + 1;
    }
    return column;
}

public double calculate_arrival_time(){
    Node node = new Node();
    double fanin_arrival_time = 0;

    arrival_time = get_delay();
    if (fanin_node_list == null)
        return arrival_time;
    for (int index = 0; index < fanin_node_list.size(); index++){
        node = (Node)fanin_node_list.get(index);
        if (fanin_arrival_time < node.calculate_arrival_time())
            fanin_arrival_time = node.get_arrival_time();
    }
    arrival_time += fanin_arrival_time;
    return arrival_time;
}

public double calculate_required_time(double max_arrival_time){
    Node node = new Node();
    required_time = max_arrival_time;
    if (fanout_node_list == null){
        if (fanin_node_list == null){
            required_time = 0;
        }
        return required_time;
    }
    for (int index = 0; index < fanout_node_list.size(); index++){
        node = (Node)fanout_node_list.get(index);

```

```

        if (required_time > node.calculate_required_time(max_arrival_time) -
node.get_delay()){
            required_time = node.get_required_time() - node.get_delay();
        }
    }
    return required_time;
}

public double calculate_slack(){
    slack = 0;
    if (fanout_node_list != null)
        slack = required_time - arrival_time;
    return slack;
}

public double calculate_power_slack(){
    double coefficient = 0;
    if (fanout_node_list != null)
        coefficient = power_coefficient * delay_coefficient * switching *
fanout_node_list.size();
    if (delay != 0){
        power_slack = coefficient * (1 / delay - 1 / (delay + slack));
    }
    else{
        power_slack = 0;
    }
    return power_slack;
}

public void show_myself(){
    System.out.println(nodename);
    if (fanin_node_list != null){
        System.out.println("fanin nodes:");
        for(int index = 0; index < fanin_node_list.size(); index++){
            Node node = (Node) fanin_node_list.get(index);
            System.out.println(node.get_nodename());
        }
    }
    if (fanout_node_list != null){
        System.out.println("fanout nodes:");
        for(int index = 0; index < fanout_node_list.size(); index++){

```



```

        Node node = (Node)fanout_node_list.get(index);
        System.out.println(node.get_nodename());
    }
}
System.out.println(">>-----");
}

public String show_myself_tostring(){
    String output_string = new String("");
    output_string += "Node Name: " + nodename + "\n";
    if (fanin_node_list != null){
        output_string += "Fanin Node(s): ";
        for(int index = 0; index < fanin_node_list.size(); index++){
            Node node = (Node)fanin_node_list.get(index);
            output_string += node.get_nodename() + ", ";
        }
        output_string += "\n";
    }
    if (fanout_node_list != null){
        output_string += "Fanout Node(s): ";
        for(int index = 0; index < fanout_node_list.size(); index++){
            Node node = (Node)fanout_node_list.get(index);
            output_string += node.get_nodename() + ", ";
        }
        output_string += "\n";
    }
    output_string += "Delay: " + (float)get_delay() + " time units" + "      ";
    output_string += "Power: " + (float)get_power() + " power units" + "      ";
    output_string += "Area: " + (float)get_area() + " area units";
    return output_string;
}

public void draw(Graphics2D g2, double zoom, double radius, double dimension_coefficient,
Color node_color){
    g2.setPaint(node_color);
    // A solid stroke
    BasicStroke stroke = new BasicStroke(2);
    g2.setStroke(stroke);

    g2.drawOval(new Double((column + 1 - radius) * zoom * dimension_coefficient).intValue(),
        new Double((row - radius) * zoom * dimension_coefficient).intValue(),

```

```

        new Double(2 * radius * zoom * dimension_coefficient).intValue(),
        new Double(2 * radius * zoom * dimension_coefficient).intValue());
stroke = new BasicStroke(1);
g2.setStroke(stroke);
}

public void draw_fanin_lines(Graphics2D g2, double zoom, double radius, double
dimension_coefficient, Color line_color){
    g2.setPaint(line_color);
    if (fanin_node_list != null){
        for(int index = 0; index < fanin_node_list.size(); index++){
            Node node = (Node)fanin_node_list.get(index);
            g2.draw(new Line2D.Double((node.get_column() + 1 + radius) * zoom *
dimension_coefficient,
                                     node.get_row() * zoom * dimension_coefficient,
                                     (column + 1 - radius) * zoom * dimension_coefficient,
                                     row * zoom * dimension_coefficient));
        }
    }
}

public void draw_fanout_lines(Graphics2D g2, double zoom, double radius, double
dimension_coefficient, Color line_color){
    g2.setPaint(line_color);
    if (fanout_node_list != null){
        for(int index = 0; index < fanout_node_list.size(); index++){
            Node node = (Node)fanout_node_list.get(index);
            g2.draw(new Line2D.Double((column + 1 + radius) * zoom * dimension_coefficient,
                                     row * zoom * dimension_coefficient,
                                     (node.get_column() + 1 - radius) * zoom *
dimension_coefficient,
                                     node.get_row() * zoom * dimension_coefficient));
        }
    }
}

public boolean clickInside(int x, int y, double zoom, double dimension_coefficient, double
radius){
    boolean inside = false;

    if ((x >= new Double((column + 1 - radius) * zoom * dimension_coefficient).intValue())

```

```
        && ( y >= new Double((row - radius) * zoom * dimension_coefficient).intValue())
        && ( x <= new Double((column + 1 + radius) * zoom * dimension_coefficient).intValue())
        && ( y <= new Double((row + radius) * zoom * dimension_coefficient).intValue())){
            inside = true;
        }
    return inside;
}
}
```

Appendix B

Program 2 – SET Threshold Gates Parameter Selection and Optimization (Chapter 3, 4)

```

/*****
*Functions Profile:
This is a command line program. With a threshold expression as input, this tool determines the
structure of a SET threshold gate, and then gives all capacitance parameters. Also, this tool
has the ability to analyze the reliability of a SET threshold gate, with a uniform or normal
distribution background charge on nodes of the threshold logic structure.

*Usage:
java Threshold <threshold expression>;
/*example: 2-input AND */ java Threshold sgn{a+b-1.5};

*Package Structure:
Package com.ThresholdGate
    |--- Class Threshold                // Command line application launcher
    |           |--- Class Expressionreader // Threshold Expression reader
    |           |--- Class Inputport      // Inputport holder
*****/

```

```

//Threshold.java
package com.ThresholdGate;
import java.util.*;
import java.io.*;

public class Threshold extends Expressionreader{
    private boolean always_open = false;
    private boolean always_close = false;
    private double open_Cb;
    private double close_Cb;
    private double optimal_Cb;
    private double optimal_prob;
    private double sum_Cp;

    public void set_always_open(boolean always_open){
        this.always_open = always_open;
    }

    public boolean get_always_open(){
        return always_open;
    }

    public void set_always_close(boolean always_close){
        this.always_close = always_close;
    }

    public boolean get_always_close(){
        return always_close;
    }

    public double get_open_Cb(){
        return open_Cb;
    }

    public void set_open_Cb(double open_Cb){
        this.open_Cb = open_Cb;
    }

    public double get_close_Cb(){
        return close_Cb;
    }
}

```

```

public void set_close_Cb(double close_Cb){
    this.close_Cb = close_Cb;
}

public double get_optimal_Cb(){
    return optimal_Cb;
}

public void set_optimal_Cb(double optimal_Cb){
    this.optimal_Cb = optimal_Cb;
}

public double get_optimal_prob(){
    return optimal_prob;
}

public void set_optimal_prob(double optimal_prob){
    this.optimal_prob = optimal_prob;
}

public double get_sum_Cp(){
    return sum_Cp;
}

public void set_sum_Cp(double sum_Cp){
    this.sum_Cp = sum_Cp;
}

public static void main(String[] args){
    Threshold threshold = new Threshold();
    threshold.new_inputport_list();
    threshold.translator(args[0]);
    threshold.locate_open_signal();
    threshold.clear_signal();
    threshold.locate_close_signal();
    threshold.clear_signal();
    threshold.get_threshold();
    //--threshold.test_open_close_signal();
    //--threshold.test_inputport_list();
    //--threshold.voltage_range_even_distribution(13);
}

```

```

        //--threshold.pass_probability(13);
        //--threshold.allowable_delta_voltage(13, 1);
        //--threshold.allowable_delta_voltage(13, 0);
        threshold.calculate_optimal_Cb(50);
    }

    public void clear_signal(){
        List inputport_list = get_inputport_list();
        for (int index = 0; index < inputport_list.size(); index++){
            Inputport inputport = (Inputport)inputport_list.get(index);
            inputport.set_signal(0);
        }
    }

    public void locate_open_signal(){
        List inputport_list = get_inputport_list();
        double signal_sum = get_threshold_value();
        int addup_signal = 0;
        int all_high_signal = 0;
        while ((signal_sum >= 0) || (signal_sum <= -1)){
            signal_sum = get_threshold_value();
            for (int index = 0; index < inputport_list.size(); index++){
                Inputport inputport = (Inputport)inputport_list.get(index);
                if (index == 0){
                    inputport.reverse_signal();
                    if (inputport.get_signal() == 0)
                        addup_signal = 1;
                    all_high_signal = inputport.get_signal();
                }
                else {
                    inputport.set_signal(inputport.get_signal() + addup_signal);
                    if (inputport.get_signal() == 2){
                        inputport.set_signal(0);
                        addup_signal = 1;
                    }
                    else{
                        addup_signal = 0;
                    }
                    all_high_signal += inputport.get_signal();
                }
            }
            if (inputport.get_pole() == 'p')

```

```

        signal_sum = signal_sum - inputport.get_open_signal() *
inputport.get_weight();
        else if (inputport.get_pole() == 'n')
            signal_sum = signal_sum + inputport.get_open_signal() *
inputport.get_weight();
    }
    //System.out.println("open signal sum: " + signal_sum);
    if ((signal_sum > -1) && (signal_sum < 0))
        break;
    for (int index = 0; index < inputport_list.size(); index++){
        Inputport inputport = (Inputport)inputport_list.get(index);
        inputport.set_open_signal(inputport.get_signal());
    }
    if (all_high_signal == 0){
        if (signal_sum <= -1)
            set_always_open(true);
        else if (signal_sum >= 1)
            set_always_close(true);
        break;
    }
}

}

public void locate_close_signal(){
    List inputport_list = get_inputport_list();
    double signal_sum = get_threshold_value();
    int addup_signal = 0;
    int all_high_signal = 0;
    while ((signal_sum >= 1) || (signal_sum <= 0)){
        signal_sum = get_threshold_value();
        for (int index = 0; index < inputport_list.size(); index++){
            Inputport inputport = (Inputport)inputport_list.get(index);

            if (index == 0){
                inputport.reverse_signal();
                if (inputport.get_signal() == 0)
                    addup_signal = 1;
                all_high_signal = inputport.get_signal();
            }
            else {
                inputport.set_signal(inputport.get_signal() + addup_signal);
            }
        }
    }
}

```



```

        if (inputport.get_signal() == 2){
            inputport.set_signal(0);
            addup_signal = 1;
        }
        else{
            addup_signal = 0;
        }
        all_high_signal += inputport.get_signal();
    }
    if (inputport.get_pole() == 'p')
        signal_sum = signal_sum - inputport.get_close_signal() *
inputport.get_weight();
    else if (inputport.get_pole() == 'n')
        signal_sum = signal_sum + inputport.get_close_signal() *
inputport.get_weight();
    }
    //System.out.println("close signal sum: " + signal_sum);
    if ((signal_sum > 0) && (signal_sum < 1))
        break;
    for (int index = 0; index < inputport_list.size(); index++){
        Inputport inputport = (Inputport)inputport_list.get(index);
        inputport.set_close_signal(inputport.get_signal());
    }
    if (all_high_signal == 0){
        if (signal_sum <= -1)
            set_always_open(true);
        else if (signal_sum >= 1)
            set_always_close(true);
        break;
    }
}
}

public void test_open_close_signal(){
    if (get_always_close()){
        System.out.println("Tunnel always close! Threshold meaningless...");
        test_inputport_list();
        return;
    }
    else if (get_always_open()){
        System.out.println("Tunnel always open! Threshold meaningless...");
    }
}

```

```

        test_inputport_list();
        return;
    }
    List inputport_list = get_inputport_list();
    if (inputport_list.isEmpty()){
        Expression_illegal_help();
        return;
    }
    System.out.println("threshold value:" + get_threshold_value());
    for (int index = 0; index < inputport_list.size(); index++){
        Inputport inputport = (Inputport)inputport_list.get(index);
        System.out.println((index + 1) + ":");
        System.out.println("pole: " + inputport.get_pole());
        System.out.println("portname: " + inputport.get_portname());
        System.out.println("weight: " + inputport.get_weight());
        System.out.println("open signal: " + inputport.get_open_signal());
        System.out.println("close signal: " + inputport.get_close_signal());
        System.out.println("signal: " + inputport.get_signal());
    }
}

public double calculate_Co(){
    double Co = 9;
    List inputport_list = get_inputport_list();
    for (int index = 0; index < inputport_list.size(); index++){
        Inputport inputport = (Inputport)inputport_list.get(index);
        if (inputport.get_pole() == 'n')
            Co -= 0.5 * inputport.get_weight();
    }
    return Co;
}

public double calculate_open_Cb(){
    List inputport_list = get_inputport_list();
    for (int index = 0; index < inputport_list.size(); index++){
        Inputport inputport = (Inputport)inputport_list.get(index);
        inputport.set_signal(inputport.get_open_signal());
    }
    set_open_Cb(calculate_Cb());
    return get_open_Cb();
}

```

```

}

public double calculate_close_Cb(){
    List inputport_list = get_inputport_list();
    for (int index = 0; index < inputport_list.size(); index++){
        Inputport inputport = (Inputport)inputport_list.get(index);
        inputport.set_signal(inputport.get_close_signal());
    }
    set_close_Cb(calculate_Cb());
    return get_close_Cb();
}

public double calculate_Cb(){
    double Cb = 0.1;
    double Cb_temp_low = 0.1;
    double Cb_temp_high = 10000;
    double sum_Cp = 0;
    double sum_Cp_1 = 0;
    double sum_Cn_1 = 0;
    double Vj = 0; //voltage across junction
    double Vc = 0; //critical voltage
    double Cj = 0.1; //junction capacitor
    double Cl = 10; //total loading capacitor, original 10
    double Vbuffer = 1.17; //original 1.17
    List inputport_list = get_inputport_list();
    for (int index = 0; index < inputport_list.size(); index++){
        Inputport inputport = (Inputport)inputport_list.get(index);
        if (inputport.get_pole() == 'p'){
            sum_Cp += 0.5 * inputport.get_weight();
            if (inputport.get_signal() == 1)
                sum_Cp_1 += 0.5 * inputport.get_weight();
        }
        if ((inputport.get_pole() == 'n') && (inputport.get_signal() == 1))
            sum_Cn_1 += 0.5 * inputport.get_weight();
    }
    //System.out.println("sum_Cp: " + sum_Cp);
    set_sum_Cp(sum_Cp);
    //System.out.println("sum_Cp_1: " + sum_Cp_1);
    //System.out.println("sum_Cn_1: " + sum_Cn_1);
    for (Cb = Cb_temp_low; Cb < Cb_temp_high; Cb += 0.05){
        Vj = 16 * (Cb/(Cb + sum_Cp) + sum_Cp_1/(sum_Cp + Cb) - sum_Cn_1/Cl) - Vbuffer;
    }
}

```

```

        Vc = 80/(Cj + Cl*(Cb + sum_Cp)/(Cb + sum_Cp + Cl));
        if (((Vj - Vc)/Vc < 0.001) && ((Vj - Vc)/Vc > -0.001)){
            //System.out.println("Vj: " + Vj);
            //System.out.println("Vc: " + Vc);
            break;
        }
    }
    return Cb;
}

public void get_threshold(){
    if (get_always_close()){
        System.out.println("Tunnel always close! Threshold meaningless...");
        test_inputport_list();
        return;
    }
    else if (get_always_open()){
        System.out.println("Tunnel always open! Threshold meaningless...");
        test_inputport_list();
        return;
    }
    List inputport_list = get_inputport_list();
    if (inputport_list.isEmpty()){
        Expression_illegal_help();
        return;
    }
    System.out.println("threshold value:" + get_threshold_value());
    System.out.println("Co: " + calculate_Co());
    System.out.println("Open Cb: " + calculate_open_Cb());
    System.out.println("Close Cb: " + calculate_close_Cb());
    for (int index = 0; index < inputport_list.size(); index++){
        Inputport inputport = (Inputport)inputport_list.get(index);
        System.out.println((index + 1) + ":");
        System.out.println("pole: " + inputport.get_pole());
        System.out.println("portname: " + inputport.get_portname());
        System.out.println("weight: " + inputport.get_weight());
        System.out.println("open signal: " + inputport.get_open_signal());
        System.out.println("close signal: " + inputport.get_close_signal());
        System.out.println("signal: " + inputport.get_signal());
    }
}
}

```

```

public double voltage_range_even_distribution(double Cb){
    double C1 = 10; //total loading

    //-- (-0.5e, 0.5e) even distribution
    //double voltage_range = 80 * (Cb + get_sum_Cp() + C1) / ((Cb + get_sum_Cp()) * C1);

    //-- (-0.4e, 0.4e) even distribution
    //double voltage_range = 160 * 0.4 * (Cb + get_sum_Cp() + C1) / ((Cb + get_sum_Cp())
* C1);

    //-- (-0.3e, 0.3e) even distribution
    //double voltage_range = 160 * 0.3 * (Cb + get_sum_Cp() + C1) / ((Cb + get_sum_Cp())
* C1);

    //-- (-0.2e, 0.2e) even distribution
    //double voltage_range = 160 * 0.2 * (Cb + get_sum_Cp() + C1) / ((Cb + get_sum_Cp())
* C1);

    //-- (-0.1e, 0.1e) even distribution
    //double voltage_range = 160 * 0.1 * (Cb + get_sum_Cp() + C1) / ((Cb + get_sum_Cp())
* C1);

    //-- (-0.05e, 0.05e) even distribution
    //double voltage_range = 160 * 0.05 * (Cb + get_sum_Cp() + C1) / ((Cb + get_sum_Cp())
* C1);

    //-- (-0.03e, 0.03e) even distribution
    //double voltage_range = 160 * 0.03 * (Cb + get_sum_Cp() + C1) / ((Cb + get_sum_Cp())
* C1);

    //-- (-0.02e, 0.02e) even distribution
    double voltage_range = 160 * 0.02 * (Cb + get_sum_Cp() + C1) / ((Cb + get_sum_Cp())
* C1);

    //-- (-0.01e, 0.01e) even distribution
    //double voltage_range = 160 * 0.01 * (Cb + get_sum_Cp() + C1) / ((Cb + get_sum_Cp())
* C1);

    //System.out.println("voltage_range: " + voltage_range);
    return voltage_range;
}

```

```

}

public double allowable_delta_voltage(double Cb, int open_close){
    double sum_Cp = 0;
    double sum_Cp_1 = 0;
    double sum_Cn_1 = 0;
    double Vj = 0; //voltage across junction
    double Vc = 0; //critical voltage
    double Cj = 0.1; //junction capacitor
    double Cl = 10; //total loading capacitor
    double Vbuffer = 1.17;
    List inputport_list = get_inputport_list();
    for (int index = 0; index < inputport_list.size(); index++){
        Inputport inputport = (Inputport)inputport_list.get(index);
        if (open_close == 1) //open
            inputport.set_signal(inputport.get_open_signal());
        else if (open_close == 0) //close
            inputport.set_signal(inputport.get_close_signal());
        else
            return 0;
        if ((inputport.get_pole() == 'n') && (inputport.get_signal() == 1))
            sum_Cn_1 += 0.5 * inputport.get_weight();
        else if ((inputport.get_pole() == 'p') && (inputport.get_signal() == 1))
            sum_Cp_1 += 0.5 * inputport.get_weight();
    }
    sum_Cp = get_sum_Cp();
    Vj = 16 * (Cb/(Cb + sum_Cp) + sum_Cp_1/(sum_Cp + Cb) - sum_Cn_1/Cl) - Vbuffer;
    Vc = 80/(Cj + Cl*(Cb + sum_Cp)/(Cb + sum_Cp + Cl));
    //System.out.println("delta_voltage: " + (Vc - Vj));
    return (Vc - Vj);
}

public double pass_probability(double Cb){
    double voltage_range = voltage_range_even_distribution(Cb);
    double close_delta_voltage = allowable_delta_voltage(Cb, 0);
    double open_delta_voltage = allowable_delta_voltage(Cb, 1);
    double close_part = 0;
    double open_part = 0;
    if (voltage_range - close_delta_voltage > 0)
        close_part = 0.5 * Math.pow((voltage_range - close_delta_voltage)/voltage_range,
2);

```

```

        if (voltage_range + open_delta_voltage > 0)
            open_part = 0.5 * Math.pow((voltage_range + open_delta_voltage) / voltage_range, 2);

        //double pass_probability = 1 - 0.5 * Math.pow((voltage_range -
close_delta_voltage) / voltage_range, 2)
        //
        // - 0.5 * Math.pow((voltage_range +
open_delta_voltage) / voltage_range, 2);
        double pass_probability = 1 - close_part - open_part;
        //System.out.println("pass probability: " + pass_probability);
        return pass_probability;
    }

    public double pass_normal_probability(double Cb, double sigma){
        double C1 = 10;
        double sum_Cp = get_sum_Cp();
        double voltage_sigma = sigma / Math.pow(10, -18) * Math.sqrt(Math.pow(Cb + sum_Cp,
2) + Math.pow(C1, 2)) / (Cb + sum_Cp) / C1;
        double close_delta_voltage = allowable_delta_voltage(Cb, 0) / 1000;
        double open_delta_voltage = allowable_delta_voltage(Cb, 1) / 1000;
        //System.out.println("close voltage: " + close_delta_voltage);
        //System.out.println("open voltage: " + open_delta_voltage);
        double steps = 200;
        double step_size = (close_delta_voltage - open_delta_voltage) / steps;
        double pass_normal_probability = 0;
        for (double voltage = open_delta_voltage; voltage <= close_delta_voltage; voltage +=
step_size){
            pass_normal_probability += (1 / (Math.sqrt(2 * Math.PI) * voltage_sigma) *
Math.exp(- Math.pow(voltage / voltage_sigma, 2) / 2)) * step_size;
        }
        return pass_normal_probability;
    }

    public double calculate_optimal_Cb(int steps){
        double current_Cb = get_open_Cb();
        double optimal_Cb = get_open_Cb();
        double lowest_Cb = get_open_Cb();
        double step_size = (get_close_Cb() - get_open_Cb()) / steps;
        double current_probability = 0;
        double optimal_probability = 0;
        double lowest_probability = 1;

```

```

        try{
            BufferedWriter out = new BufferedWriter(new FileWriter("recorder"));

            for (current_Cb = get_open_Cb(); current_Cb <= get_close_Cb(); current_Cb +=
step_size){
                //current_probability = pass_probability(current_Cb);
                current_probability = pass_normal_probability(current_Cb, 0.1 * 1.6 *
Math.pow(10, -19));
                System.out.println("current Cb: " + current_Cb);
                System.out.println("current prob: " + current_probability);
                out.write(current_Cb + "          " + current_probability + "\n");

                if (current_probability > optimal_probability){
                    optimal_Cb = current_Cb;
                    optimal_probability = current_probability;
                }
                if (current_probability < lowest_probability){
                    lowest_Cb = current_Cb;
                    lowest_probability = current_probability;
                }
            }
            out.close();
        } catch (IOException e) {
        }

        set_optimal_Cb(optimal_Cb);
        set_optimal_prob(optimal_probability);
        System.out.println("optimal Cb: " + optimal_Cb + " prob: " + optimal_probability);
        System.out.println("lowest Cb: " + lowest_Cb + " prob: " + lowest_probability);

        return optimal_Cb;
    }
}

// Expressionreader.java
package com.ThresholdGate;
import java.util.*;
import java.lang.reflect.Array;

class Expressionreader{

```



```

private List inputport_list;
private double threshold_value;

public List get_inputport_list(){
    return inputport_list;
}

public void set_inputport_list(List inputport_list){
    this.inputport_list = inputport_list;
}

public void new_inputport_list(){
    inputport_list = new ArrayList();
}

public double get_threshold_value(){
    return threshold_value;
}

public void set_threshold_value(double threshold_value){
    this.threshold_value = threshold_value;
}

public static void main(String[] args){
    Expressionreader reader = new Expressionreader();
    reader.new_inputport_list();
    reader.translator(args[0]);
    reader.test_inputport_list();
}

public void translator(String threshold_expression){
    String expression = threshold_expression.trim();
    if ((!expression.substring(0,4).toLowerCase().equals(new String("sgn{"))
        ||(!expression.substring(expression.length()-1).equals(new String(")")))){
        Expression_illegal_help();
        return;
    }
    expression = expression.substring(4);
    expression = expression.substring(0,expression.length()-1);
    if (!expression.startsWith("-"))
        expression = new String("+").concat(expression);
}

```

```

//System.out.println(expression);
String [] minus_split = expression.split("-");
int minus_split_length = Array.getLength(minus_split);
for (int i = 0; i < minus_split_length; i++ ){
    if (minus_split[i].equals(new String("")))
        continue;
    else
        minus_split[i] = minus_split[i].trim();
//System.out.println(minus_split[i]);
String [] plus_minus_split = minus_split[i].split("\\+");
int plus_minus_split_length = Array.getLength(plus_minus_split);
//System.out.println(plus_minus_split_length);
for (int j = 0; j < plus_minus_split_length; j++){
    if (plus_minus_split[j].equals(new String("")))
        continue;
    else
        plus_minus_split[j] = plus_minus_split[j].trim();
//System.out.println(plus_minus_split[j]);
if (plus_minus_split[j].lastIndexOf(".") != -1){
    if (j == 0)
        set_threshold_value(0 - Double.parseDouble(plus_minus_split[j]));
    else
        set_threshold_value(Double.parseDouble(plus_minus_split[j]));
}
else{
    Inputport input_port = new Inputport();
    if (j == 0)
        input_port.set_pole('p');
    else
        input_port.set_pole('n');
String [] multiple_split = plus_minus_split[j].split("\\*");
if (Array.getLength(multiple_split) == 1){
    input_port.set_portname(multiple_split[0].trim());
}
else if (Array.getLength(multiple_split) == 2){

input_port.set_weight(Double.parseDouble(multiple_split[0].trim()));
        input_port.set_portname(multiple_split[1].trim());
    }
    else {
        Expression_illegal_help();
    }
}
}

```

```

        return;
    }
    inputport_list.add(input_port);
}
}
}

public void Expression_illegal_help(){
    System.out.println("illegal threshold expression!");
    System.out.println("-----EXPRESSION EXAMPLE-----");
    System.out.println("    sgn(a+2*b-2.5)");
    System.out.println("-----");
}

public void test_inputport_list(){
    if (inputport_list.isEmpty()){
        Expression_illegal_help();
        return;
    }
    System.out.println("threshold value:" + threshold_value);
    for (int index = 0; index < inputport_list.size(); index++){
        Inputport inputport = (Inputport)inputport_list.get(index);
        System.out.println((index + 1) + ":");
        System.out.println("pole: " + inputport.get_pole());
        System.out.println("portname: " + inputport.get_portname());
        System.out.println("weight: " + inputport.get_weight());
    }
}
}

```

```

//Inputport.java
package com.ThresholdGate;

class Inputport{
    private char pole = 'n'; // 'n' or 'p'
    private double weight = 1; //
    private String portname;
    private int signal = 0; // 0 or 1
    private int open_signal = 0; // 0 or 1
}

```

```

private int close_signal = 0; // 0 or 1

public void set_portname(String portname){
    this.portname = portname;
}

public String get_portname(){
    return portname;
}

public void set_pole(char pole){
    this.pole = pole;
}

public char get_pole(){
    return pole;
}

public void set_weight(double weight){
    this.weight = weight;
}

public double get_weight(){
    return weight;
}

public void set_signal(int signal){
    this.signal = signal;
}

public int get_signal(){
    return signal;
}

public int reverse_signal(){
    if (signal == 1){
        signal =0;
    }
    else {
        signal = 1;
    }
}

```

```

        return signal;
    }

    public void set_open_signal(int open_signal){
        this.open_signal = open_signal;
    }

    public int get_open_signal(){
        return open_signal;
    }

    public int reverse_open_signal(){
        if (open_signal == 1){
            open_signal =0;
        }
        else {
            open_signal = 1;
        }
        return open_signal;
    }

    public void set_close_signal(int close_signal){
        this.close_signal = close_signal;
    }

    public int get_close_signal(){
        return close_signal;
    }

    public int reverse_close_signal(){
        if (close_signal == 1){
            close_signal =0;
        }
        else {
            close_signal = 1;
        }
        return close_signal;
    }
}

```

References

- [1] Gary K. Yeap, "Practical low power digital VLSI design", Kluwer Academic Publishers, 1998
- [2] C. Chen, X. Yang, and M. Sarrafzadeh, "Predicting Potential Performance for Digital Circuits", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 21, no. 3, March 2002, pp. 253-262.
- [3] M. Vujkovic and C. Sechen, "Optimized Power-Delay Curve Generation for Standard Cell ICs," in Proc. of International Conference on Computer-Aided Design, November 2002, pp. 387-394.
- [4] C. C Kuo and A. C. Wu, "Delay Budgeting for a Timing-Closure-Driven Design Method", in Proc. of International Conference on Computer-Aided Design, November 2000, pp.202-207.
- [5] M. Sarrafzadeh, D. Knol, and G. Tellez, "A Delay Budgeting Algorithm Ensuring Maximum Flexibility in Placement," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 16, no. 11, Nov 1997, pp. 1332-1341.
- [6] Elaheh Bozorgzadeh, et al, "Optimal Integer Delay Budgeting on Directed Acyclic Graphs," in Proc. of the 40th Design Automation Conference, June 2003, pp. 920-925.

- [7] R. Nair, C. L. Berman, P. S. Hauge, and E. J. Yoffa, "Generation of Performance Constraints for Layout," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 8, August 1989, pp. 860-874.
- [8] C. Chen, X. Yang, and M. Sarrafzadeh, "Potential Slack: An Effective Metric of Combinational Circuit Performance," in *Proc. of International Conference on Computer-Aided Design*, November 2000, pp. 198-201.
- [9] E. M. Sentovich et al., "SIS: A System for Sequential Circuit Synthesis," Technical Report UCB/Erl M92/41, Univ. of California, Berkeley, May 1992.
- [10] T. Gao, P. M. Vaidya, and C. L. Liu, "A new performance driven placement algorithm," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1991, pp. 44-47.
- [11] H. Youssef and E. Shragowitz, "Timing constraints for correct performance," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1990, pp. 24-27.
- [12] H. R. Lin and T. Hwang, "Power reduction by gate sizing with path-oriented slack calculation," in *Proc. Asia-South Pacific Design Automation Conf.*, June 1995, pp. 7-12.
- [13] W. Swartz and C. Sechen, "Timing driven placement for large standard cell circuits," in *Proc. Design Automation Conf.*, June 1995, pp. 211-215.
- [14] J. Cong, L. He, K. Khoo, C. Koh, and Z. Pan, "Interconnection design for deep submicron IC's," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1997, pp. 478-485.
- [15] R. H. Moehring, *Graphs and Orders: The Role of Graphs in the Theory of Ordered Sets and its Applications*, I. Rival, Ed. New York: D. Reidel, 1984, pp. 41-101.
- [16] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.
- [17] A. Iranli, P. Rezvani, and M. Pedram, "Low Power Synthesis of Finite State Machines with Mixed D and T Flip-Flops," in *Proceedings of 2003 Asia and South Pacific Design Automation Conference (ASPDAC)*, January 2003, pp. 803-808.
- [18] S. Chattopadhyay and P. N. Reddy, "Finite State Machine State Assignment

- Targeting Low Power Consumption,” IEE Proceedings – Computers and Digital Techniques, vol. 151, no. 1, January 2004, pp. 61-70.
- [19] L. Benini, G. D. Micheli, and F. Vermeulen, “Finite State Machine Partitioning for Low Power,” in Proceedings of the 1998 IEEE International Symposium on Circuits and Systems (ISCAS), vol. 2, June 1998, pp. 5-8.
- [20] K. K. Likharev, “Single-Electron Devices and Their Applications,” Proceedings of IEEE, vol. 87, no. 4, April 1999, pp. 606-632.
- [21] A. M. Ionescu, M. Declercq, S. Mahapatra, K. Banerjee, J. Gautier, “Few Electron Devices: Towards Hybrid CMOS-SET Integrated Circuits,” in Proceedings of ACM/IEEE Design Automation Conference (DAC), June 2002, pp. 88-93.
- [22] C. Lageweg, S. Cotofana, and S. Vassiliadis, “Single Electron Encoded Latches and Flip-Flops,” in Proceedings of the 4th IEEE Conference on Nanotechnology, August 2004, pp. 327-330.
- [23] K. Yano, et al, “Single-Electron Memory for Giga-to-Tera Bit Storage,” Proceedings of the IEEE, vol. 87, no. 4, April 1999, pp. 633-651.
- [24] C. Wasshuber, Computational Single-Electronics, Springer-Verlag Wien New York, 2001.
- [25] K. Finkenzer, RFID Handbook, Jhon Wiley & Sons, 2003.
- [26] F. Zhou, C. Chen, et al, “Evaluating and Optimizing Power Consumption of Anti-Collision Protocols for Applications in RFID system,” in Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED), August 2004, pp. 357-362.
- [27] C. Wasshuber, H. Kosina, and S. Selberherr, “SIMON - A Simulator for Single-Electron Tunnel Devices and Circuits,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 16, no. 9, September 1997, pp. 937-944.
- [28] R. H. Klunder and J. Hoekstra, “Programmable Logic Using a SET Electron Box,” in

- Proceedings of the 8th IEEE International Conference on Electronics, Circuits and Systems (ICECS), vol. 1, September 2001, pp. 185-188.
- [29] C. Lageweg, S. Cotofana, and S. Vassiliadis, "Binary Addition Based on Single Electron Tunneling Devices," in Proceedings of the 4th IEEE Conference on Nanotechnology, August 2004, pp. 327-330.
- [30] C. Lageweg, S. Cotofana, and S. Vasiliadis, "A Linear Threshold Gate Implementation in Single Electron Technology," in Proceedings of IEEE Computer Society Workshop on VLSI, April 2001, pp. 93-98.
- [31] R. H. Chen, A. N. Korotkov, and K. K. Likharev, "Single-Electron Transistor Logic," *Appl. Phys. Lett.*, vol. 68, 1996, pp. 1954-1956.
- [32] A. M. Ionescu, M. Declercq, S. Mahapatra, K. Banerjee, J. Gautier, "Few Electron Devices: Towards Hybrid CMOS-SET Integrated Circuits," in Proceedings of ACM/IEEE Design Automation Conference (DAC), June 2002, pp. 88-93.
- [33] S. M. Goodnick and J. Bird, "Quantum-Effect and Single-Electron Devices," *IEEE Transactions on Nanotechnology*, vol. 2, no. 4, December 2003, pp. 368-385.
- [34] C. Wasshuber, H. Kosina, and S. Selberherr, "SIMON - A Simulator for Single-Electron Tunnel Devices and Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 9, September 1997, pp. 937-944.
- [35] Y. Taur, D.A. Buchanan, W. Chen, D. Frank, K. Ismail, S. Lo, G. Sai-Halasz, R. Viswanathan, H. Wann, S. Wind, and H. Wong, "CMOS scaling into the nanometer regime," *Proc. IEEE*, vol. 85, Apr. 1997, pp. 486-504.
- [36] C. Lageweg, S. Cotofana, and S. Vassiliadis, "A linear threshold gate implementation in single electron technology," in IEEE Computer Society VLSI Workshop, Apr. 2001, pp. 93-98.
- [37] T. Oya, T. Asai, T. Fukui, and Y. Amemiya, "A majority-logic device using and irreversible single-electron box," *IEEE Trans. Nanotechnol.*, vol. 2, Mar. 2003,

pp. 15–22.

- [38] S. Banerjee, S. Huang, and S. Oda, “Operation of nanocrystalline-silicon- based few-electron memory devices in the light of electron storage, ejection and lifetime characteristics,” *IEEE Trans. Nanotechnol.*, vol. 2, June 2003, pp. 88–92.
- [39] K. Yano, T. Ishii, T. Sano, T. Mine, F. Murai, T. Hashimoto, T. Kobayashi, T. Kure, and K. Seki, “Single-electron memory for giga-to-tera bit storage,” *Proc. IEEE*, vol. 87, Apr. 1999, pp. 633–651.
- [40] I. Karafyllidis, “Design and simulation of a single-electron random-access memory array,” *IEEE Trans. Circuits Syst. I*, vol. 49, Sept. 2002, pp. 1370–1375.
- [41] <http://java.sun.com>
- [42] J. Mi and C. Chen, “Power-oriented delay budgeting for combinational circuits”, to appear in *Proceedings of 2006 ISCAS*.
- [43] J. Mi and C.Chen, “Finite state machine implementation with single-electron tunneling technology”, to appear in *Proceedings of 2006 ISVLSI*.

VITA AUCTORIS

Jialin Mi was born in 1977 in Shanghai, China. He received his Bachelor's Degree in Communication from the Electronics Engineering Department of Shanghai JiaoTong University in 2000. He is currently a candidate for a Master of Applied Science Degree in the Department of Electrical and Computer Engineering at University of Windsor and expects to graduate in winter 2006.