I, Qiang Han, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Engineering.

It is entitled:
An Error-Tolerant Dynamic Voltage Scaling Method for Low-Power Pipeline Circuit Design

Student's name: Qiang Han

This work and its defense approved by:

Committee chair: Wen Ben Jone, PhD

Committee member: Carla Purdy, C, PhD

Committee member: Ranganadha Vemuri, PhD
An Error-Tolerant Dynamic Voltage Scaling Method
for Low-Power Pipeline Circuit Design

A thesis submitted to the
Division of Research and Advanced Studies
of the
University of Cincinnati

in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in the Department of
Electrical and Computer Engineering
of the College of Engineering
October 28, 2011

by

Qiang Han
B.S. (Telecommunication Engineering), Tianjin University, June 2007

Thesis Advisor and Committee Chair: Dr. Wen-Ben Jone
Abstract

In recent years, many innovative researches have been conducted on dynamic voltage scaling (DVS), such as Razor [1]. This thesis presents an error-tolerant DVS design that can enhance the reliability and reduce the power consumption of a pipeline circuit simultaneously. Based on delay distributions of all pipeline stages, an efficient voltage island partitioning method is developed to cluster all pipeline stages into several voltage islands. By assigning the best voltages to stages, the DVS design can enable the pipeline stages to work at an optimal energy consumption with least performance penalty. Experimental results obtained by HSPICE and Matlab simulations demonstrate the feasibility of this method.
To my Dearest Parents
Acknowledgement

I would like to extend my sincere thanks to my advisor Dr. Wen-Ben Jone, who has mentored me along all the way of my research. He is always ready to help me overcome the obstacles on this work and inspire me towards a brilliant direction. He has worked with me constantly giving me advises even during vacation and in the late hours of the night.

I would like to thank my committee members, Dr. Ranga Vemuri and Dr. Carla Purdy, for taking the time out of their busy schedules to review this work. Special thanks goes to Hao Xu for his help and suggestion to my research, and thank Julie Muechen for her work on my master graduation.

I would like to say thank you to my dearest parents, for their endless support and love since I was born. Finally, I express sincere gratitude to my girlfriend Mengxia, for her constant support and love.

Thank you all.
Contents

1 Introduction 1

2 Background 6
   2.1 Circuit Power and Energy 6
   2.2 Strategies to address power consumption 9
   2.3 RAZOR 13
      2.3.1 Dynamic Voltage Scaling 13
      2.3.2 RAZOR 15

3 Fine-Grained Razor Design 20
   3.1 Variations 20
      3.1.1 Process Variation 20
      3.1.2 Input Vector 22
      3.1.3 Uneven Pipeline Stage 23
   3.2 Razor’s Potential 24
   3.3 Fine-Grained Razor Design 26
      3.3.1 Statistical Analysis on Propagation Delay 26
3.3.2 Normal Mode in Pipeline ................................... 28
3.3.3 Razor Mode in One Stage ................................. 30
3.3.4 Razor Mode on Pipelined Processor ................. 38
3.3.5 Fine-Grained Razor Design .............................. 41
3.3.6 Design Review ........................................... 48
3.4 Gain and Overhead .......................................... 51
  3.4.1 Gain .................................................. 51
  3.4.2 Overhead ............................................. 52

4 Experiment Setup and Results .............................. 56
  4.1 Experiment Setup ......................................... 56
    4.1.1 Experiment Setup .................................... 56
    4.1.2 EDA Tools ........................................... 64
    4.1.3 Experiment Procedure .............................. 64
  4.2 Experiment Results ....................................... 65
    4.2.1 Stage Delay Distribution and Parameter Extraction 65
    4.2.2 Stage Grouping ..................................... 70
    4.2.3 Verification of Proposed Method by HSPICE ...... 71

5 Conclusion and Future Work ............................... 75
List of Figures

2.1 The enable signal controls the circuit working mode.  
2.2 The relations of $I_D$ current versus $V_{GS}$ with different threshold 
  voltage.  
2.3 Power Gating Overview[40].  
2.4 Reverse body bias.  
2.5 Qualitative relationship between supply voltage, energy, and 
  IPC of Razor scheme[1].  
2.6 Gate level view of Razor flip-flop[1].  
2.7 Five-stage pipeline processor with centralized pipeline recovery 
  mechanism[46].  
2.8 Five-stage pipeline processor with distributed pipeline recovery 
  mechanism[1].  
3.1 Simplified(lumped) model.  
3.2 Possible propagation delays of a five-stage pipeline in one clock 
  cycle.  
3.3 The proposed error-tolerant DVS architecture.
4.11 Delay distribution curve fitting of stage 4 . . . . . . . . . . . . . . 67
4.12 Delay distribution curve fitting of stage 5 . . . . . . . . . . . . . . 68
4.13 Five curve fittings in one coordinate . . . . . . . . . . . . . . . . 69
4.14 Minimum energy consumption ($\times 10^{-9} J$) on each partition . . 71
## List of Tables

3.1 HSPICE and Matlab simulation on operating voltage and delay 31
3.2 True distribution, generated distribution and transformed distribution 35
3.3 Derivation process 37
3.4 Experiment on circuit without level shifter 55

4.1 Instruction set 58
4.2 Control Signals 59
4.3 Controls signals generation 59
4.4 EDA tools 64
4.5 Statistics of stage delay distribution 68
4.6 Stage Energy Consumption in the Normal Mode 70
4.7 Simulation result on processor circuit version 1 72
4.8 Simulation result on processor circuit version 2 73
4.9 Simulation result on processor circuit version 3 73
4.10 Simulation result on benchmark circuits 74
4.11 Experiment on benchmark circuits 74
Chapter 1

Introduction

Low power design has been researched for decades; and nowadays it becomes even more significant as a design metric in the portable device era. Dynamic voltage scaling (DVS) is one of the most effective and widely used methods for power-aware computing. DVS has been implemented in many circuits and systems [2, 3, 4, 5, 6, 7], and has also been applied to advanced commercial chips [8, 9]. The basic idea behind DVS is to reduce the supply voltage and thus power consumption of a circuit without affecting the desired performance (e.g., using parallel architecture). Another major application of DVS is to find a trade-off between power and performance. When a circuit block is detected too hot, the supply voltage can be reduced to cool down the chip with degraded performance [10, 11]. It is also possible to trade-off power consumption and reliability using DVS. For example, SRAM supply scaling down can reduce the leakage power, but the data error probability
will be increased. Thus, the design issue is to recover back the circuit relia-

bility using system level techniques like error-correction codes [12]. In fact, DVS is an effective method to trade-off power, performance and reliability for a circuit design. Recently, DVS has been pushed to the limit by scaling down the supply voltage such that chips can work in subthreshold region with extremely low power consumption [13, 14]. The major application of subthreshold voltage operating is on fabricating extremely low power chips such as medical monitoring circuits.

In digital logic circuits, there are several kinds of variations (e.g., PVT - process, voltage and temperature) that can influence circuit performance (e.g., delay, power) [15, 16, 17]. PVT variations have been identified as a major bottleneck in deep sub-micron technologies. In order to deal with PVT variations, conservative design by keeping sufficient margins has to be employed. That is, besides critical voltage determined by critical circuit paths, voltages for tolerating process margin, ambient (e.g., temperature) margin and noise (e.g., $di/dt$) margin must be added to guarantee the worst-case voltage requirement [18]. Unfortunately, the worse-case conditions rarely happen and this unnecessarily wastes power consumption with circuits working too fast mostly. To solve this problem, modern circuit design methods use adaptive control to compensate PVT variations. That is, if a circuit is working too fast (slow) due to PVT variations, the circuit itself can detect the performance deviation and increase (decrease) the circuit delay by different methods such as DVS.
There are many process-variation aware (or even PVT-aware) methods developed recently to calibrate each chip after fabrication [19, 20, 21, 22, 23]. The basic idea is to fine-tune the supply voltage or body bias, on-line or off-line, to compensate the effect of PVT variations. Resilient circuit design methods have also been proposed to push the idea of circuit adaptation to the limit in that errors are allowed to occur during normal operations [1, 24, 25, 26, 27, 28]. The working principle of resilient circuit design is to allow errors to occur but the circuit is equipped with error detectors to detect and correct them using re-execution. By allowing a small number of errors to occur, instead of working in the worst case voltage, supply voltage of the circuit can be reduced to save power consumption. High supply voltage is used only when errors are detected and the instruction is re-executed. Since the number of circuit re-executions is small, the circuit throughput is not affected significantly. The idea of resilient computing has been successfully implemented to several microprocessor chips [29, 30].

Pipelining is a technique used in the design of computers and other digital electronic devices to increase their instruction throughput [31]. The goal of pipeline design is to balance the length of each pipeline stage. However, generally, the stages will not be perfectly balanced due to pipeline partitioning by functions. For example, instruction decoding (ID) and execution (EXE) are identified as critical stages (i.e., longest delays) in Razor [1] which implemented a 64-bit error-tolerant low-power Alpha processor. Hence, uneven partitioning brings variations on propagation delay between stages. The
length of a processor cycle is determined by the delay of the slowest pipeline stage, since all stages are working at the same clock domain in most microprocessors [31]. Most resilient microprocessors, such as Razor, use single clock domain pipeline design due to the complexity of adaptive voltage or clock circuits. This motivates our idea of fine-grained adaptive voltage or clock circuit design to further take advantage of different delays for pipeline stages.

To avoid the necessity of implementing a voltage or clock regulator for each clock domain, we use DVS with a small number of voltage levels (which can be generated inside or outside the chip) as in ReVIVal [21]. To push the power-saving to the limit, we also use error detectors as in Razor [1] to allow errors to occur. Thus, our method is a compromise between Razor and ReVIVal and tries to take advantage of both. The major application is to design low-power microprocessors that cannot afford the cost of voltage or clock regulators. The major design issue is how to find the number of voltage islands and determine the supply voltage for each voltage island, such that the power consumption and the number of re-executions can be both minimized. The key idea to achieve this purpose is to partition the pipeline stages into several voltage islands based on the delay probability density function (pdf) of each pipeline stage. Experimental results obtained by circuit simulation demonstrate the feasibility of this method.

Chapter 2 reviews the background on power and energy estimation, low power schemes, and Razor.
Chapter 3 presents an optimal voltage level determination method for partitioning all pipeline stages into a single voltage island and then extends it to the dual voltage islands.

Chapter 4 describes how to set up experiments to verify the proposed method and presents the results.

Chapter 5 gives conclusions and future research.
Chapter 2

Background

In this chapter, we discusses the background knowledge related to our thesis project, which involves circuit power and energy, strategies to address power consumption, and Razor [46, 1, 24] which our proposal is based on.

2.1 Circuit Power and Energy

Low power design has been a critical theme in nowadays’ electronic industry for years. The need for low power design has caused a major paradigm shift where power dissipation has become as important a consideration as performance and area[32]. With the remarkable success and growth of the market of battery-based portable devices, micro-processors, and high performance desktops and servers, low power is triggered to be a principal factor in the IC design. Another important reason to reduce power consumption is heat
dissipation. As IC area shrinks and integration density grows, heat dissipation increases and therefore damages the performance of circuit. Low power design is also aimed to compensate the influence of heat dissipation.

Power consumption of IC is mainly divided into two parts, one is dynamic power consumption and the other is static power consumption. Dynamic power consumption is due to charging and discharging capacitances and due to short-circuit currents in switching activity, while static power consumption is from leakage current[16].

\[ P_{tot} = P_{dyn} + P_{sc} + P_{stat} = (C_L \cdot V_{DD}^2 + V_{DD} \cdot I_{sc} \cdot t_s) \cdot f_{sw} + V_{DD} \cdot I_{leak} \] (2.1)

The power consumption of charging and discharging capacitance is derived by the view of energy consumption. Every time the capacitor \( C_L \) is charged through pull-up network, its voltage rises from 0 to \( V_{DD} \), therefore approximately half of this energy is dissipated in pull-up network while the other half is stored in \( C_L \). When \( C_L \) is discharged, the stored energy is dissipated in the pull-down network[16]. While in normal operation of CMOS circuit, the ideal zero rising or falling times of input do not exist. The short circuit arises during the very short interval, when the gate voltage is switched. And for static power consumption, leakage power consumption becomes a major challenge as IC scales down. A leakage current can flow through source/drain junction, gate direct tunnel, or becomes sub-threshold leakage through the channel of an OFF transistor[33]. To put them all to-
gether, the total power consumption of CMOS circuit is expressed as the above equation.

For a portable device, such as cell phone, extending the battery usage time and to minimize the energy consumption is critical. The power-delay product is introduced as a quality measure for a logic circuit.

\[ PDP = P_{\text{tot}} \times t_p \] (2.2)

It presents a parameter of energy with the unit of Joule. Assuming that the gate is switched at its maximum possible rate of \( f_{\text{max}} = 1/t_p \), and ignoring the contributions of the static and short current to the power consumption, we see that

\[ PDP = C_L \cdot V_{DD}^2 \cdot f_{\text{max}} \cdot t_p = C_L \cdot V_{DD}^2 \] (2.3)

So PDP stands for the average energy consumed per clock cycle[16]. However, the PDP cannot represent the energy efficiency in some cases. The most significant example is a microprocessor with a replay scheme, which is designed for speculative execution, such as branch prediction. Razor also uses the replay function to flush the pipeline due to the timing error. In this case, only lowering the power will not archive a good performance, because it may increase the execution time for a certain benchmark, since the count of replays is increased. Therefore, the energy consumption can be a good metric for a speculative execution processor for finishing a certain benchmark in the
execution time, as in Equation 2.4.

\[ E = P_{tot} \times t \]  \hspace{1cm} (2.4)

2.2 Strategies to address power consumption

To deal with the challenge to reduce power consumption, IC designers have adopted multiple ways to solve it focusing on both dynamic and static power.

1. Dynamic Power Reduction

Based on the equation of \( P_{dyn} = (C_L \cdot V_{DD}^2 + V_{DD} \cdot I_s \cdot t_s) \cdot f_{sw} \), to attack dynamic power consumption, the aim is to reduce the factors of \( C_L \), \( V_{DD} \), and \( f_{sw} \).

- Clock Gating

It adds additional gates to a circuit to prune the clock tree, thus disabling parts of the circuitry in order to reduce the switching component of the power dissipation in Figure 2.1. This optimization technique is largely based on the advantage of dynamic power management methods.

- Logic Design Optimization

It includes a variety of methods, whose aims are mainly to reduce the switching activities. Good circuit design style is used to achieve symmetrical shape, and path balancing. Dynamic CMOS structure can be used instead of static CMOS to reduce the area and thus swing activities.
Sizing of Transistor and Wire

Transistor sizing at circuit level can select the component with optimum sizing for power reduction. And it can employ a large load through a source with a low driving capacitance. Wire sizing can also achieve a low load capacitance as well as good propagation.

2. Static Power Reduction

With the continuous technology scaling of IC, leakage power is playing an important role in circuitry power consumption. The aim of static power minimization is to reduce the leak current.

• Dual Threshold Voltage

In a CMOS circuit, threshold voltage determines the degree of leakage. It can be represented in Figure 2.2. Low threshold voltage CMOS gives good transition performance with short propagation delay, but it has larger leakage current compared to the high threshold voltage CMOS, so dual threshold voltage
technology employs both low threshold CMOS and high threshold CMOS in one circuitry. Low threshold CMOS is used in critical path to achieve low propagation delay, while non-critical path can adopt high threshold CMOS.

Figure 2.2: The relations of $I_D$ current versus $V_{GS}$ with different threshold voltage

- **Power Gating**

In standby mode, CMOS circuit continues to consume leakage power. To reduce leakage power consumption in standby mode, power gating is designed to shut off the power supply to the standby sections of logic [34]. The whole circuit should be partitioned into several isolations to power down the corresponding parts by power gating management.

- **Reverse Body Bias**
For NMOS (PMOS), slightly negative (positive) $V_{BS}$ can give a further reduction of leakage current on off state. Reverse body bias is usually used in standby mode. The same as power gating, it requires a power management scheme to control when negative (positive) voltage can be applied at which parts of the circuitry.
2.3 RAZOR

2.3.1 Dynamic Voltage Scaling

Recently, significant research and development have been made on Dynamic Voltage Scaling (DVS) [2, 3, 4, 5, 6, 7]. DVS is a technique of power management in computer architecture, where the voltage used in the components increases or decreases, depending on the environments[41]. DVS is a widely used technique to reduce the overall energy consumption of a processor.

Time analysis of the critical path determines the margins that need to be added in order to ensure that the replica delay path is guaranteed to fail before the core does even in the presence of a worst case combination of process variations[1]. To ensure correct operations on all possible variations,
a conservative supply voltage is chose at design-time using corner analysis. Therefore, margins are added to the critical voltage to compensate the worst-case combination of variations. Moreover, the variations would be sharper and more uncertain with the technology scaling down[46].

The basic goal of DVS is to adjust the processor’s working voltage at run time for the minimum level of performance and thus to bring down the safe margin of supply voltage according to the wide variations. Constantly adapting to the application through the different performance requirements maximizes energy conservation[35]. In a DVS system, the supply voltage is dynamically adjusted depending on environment demands to achieve the minimum power consumption. Dynamic power consumption is the dominant component of the total power consumed in current CMOS circuit. Based on the equation \( P_{\text{dyn}} = C_L \cdot V_{DD}^2 \cdot f_{sw} \), dynamic power is quadratically reduced by lowering the supply voltage[1]. Meanwhile, with the technology scaling, leakage power becomes more important to the whole consumption as well. DVS can also leverage leakage power.

On a portable device such as cell phone, to lengthen battery usage time is much critical. Energy-aware DVS is trading the performance for the energy consumption[36]. It can extend the usage time of the battery-powered equipment, as well as help to save energy on the server systems. The key point of Energy-aware DVS is to address the tradeoff between performance and battery life. Because the lower voltage gives a long propagation delay, for deadline-required tasks, the scaled voltage can not be very low, while the
other tasks can be applied with lower voltage\cite{37}. Energy-aware DVS usually applies a power management algorithm to minimize task power consumption, to provide real-time guarantees, and to consider deadlines and periodicity of real-time tasks.

To implement DVS, DC-DC converter and voltage regulator are required to tightly control voltage supply variation. Voltage regulator directs DC-DC converter to regulate and assign operating voltage to the IC when a clock frequency is requested. The procedure of regulating working voltage is monitored and operated by either hardware circuit or operating system.

\section*{2.3.2 RAZOR}

Razor, proposed by University of Michigan, is based on dynamic detection and correction of speed path failures in digital designs. The key idea of Razor is to tune the supply voltage by monitoring the error rate\cite{46}. Razor uses a delay-error tolerant flip-flop on critical paths to scale the supply voltage to the point of first failure for a given frequency\cite{1}.

It has been observed that circuit propagation is strongly data dependent, and only exhibits its worst-case delay for very specific instruction and data sequences\cite{38}. Moreover with the technology continuously scaling down, process variations of both inter- and intra-die largely exacerbate. Based on this, Razor applies moderately sub-critical supply voltages to circuit to let only a few critical instructions fail in the worst variations, while a majority of instructions will continue to operate correctly\cite{46}. 
In [1], they use a delay-error tolerant flip-flop on critical paths to scale the supply voltage to the point of first failure for a given frequency. In the figure 2.5, both energy with Razor support and energy without razor support decrease as the supply voltage scales down, while Razor support consumes more by adding razor flip-flop than non Razor support. After the point of first failure the pipeline recovery energy occurs and increases immediately. Therefore the total energy yields a curve and it has an optimal point of the minimum energy. In Razor, a delay fault is not a disaster, but a tradeoff between the quadratic energy savings and the overhead to recover from the error.

The gate level Razor flip-flop design is in figure 2.6. The outputs from
logic stage L1 are going to main flip-flop as well as a shadow latch, which is used to detecting delay fault and reserve the guaranteed correct value for recovery. A meta-detector is implemented to flag the occurrence of metastability at the output of flip-flop. The output of OR gate from shadow latch and meta-detector generates an error signal. If shadow latch finds a timing error or metastability occurs, error signal will trigger the main flip-flop to reload the correct value from scratch. This process will take one clock cycles penalty to finish.

There are two ways to implement recovery mechanism. One is centralized pipeline recovery mechanism in Figure 2.7, which uses the restore signal as a global clock-gating signal to stall the pipeline and reload the correct value. The global clock-gating signal is generated by a big OR gate from the error signal of all Razor flip-flops. It is obvious that it adds a severe
Figure 2.7: Five-stage pipeline processor with centralized pipeline recovery mechanism[46]

Figure 2.8: Five-stage pipeline processor with distributed pipeline recovery mechanism[1]

timing constraint for the pipeline. But the recovery time for it is small. It only takes one clock cycle to recover from error. In contrast, the distributed pipeline recovery mechanism in Figure 2.8 has a good timing performance. It is based on a counter-flow pipeline architecture [39]. If an error happens, it has a bubble signal to tell the next stage and the following stages to stall. Meanwhile the error signal and another flushID signal with the error instruction information will be generated and be passed in the backward direction through flip-flops to the flush controller to let errant instruction re-operate. This process normally takes more than one clock cycle. Both two mechanisms implement a dummy flip-flop in front of Write Back stage
to avoid errant date to be written back to the registers.

In [24], they simplify the Razor flip-flop for only error detection without holding the true value. Razor II uses replay scheme which is very common in modern processor for out-of-order operation to recover from error. The Razor II flip-flop increases fault toleration by introducing dynamic time-borrowing. When a potential error would occur, this stage can borrow time, from the next stage if that does not trigger a critical path at that time. Moreover, Razor II has good capability to detect logic and register Soft Error Rate (SER).
Chapter 3

Fine-Grained Razor Design

3.1 Variations

In a pipelined logic circuit, there are a few kinds of variations. Those variations have influence up on the circuits’ performance. In this section, we focus on variations’ effects on the circuit propagation delay. We take three key variations into consideration: process variation, input vector dependency, and uneven stage partition.

3.1.1 Process Variation

In reality, the parameters, including resistance capacitance and inductance, of a transistor and interconnection vary from wafer to wafer, or even from die to die[16]. Process variations are due to uncertainty in the fabricated device and interconnect characteristics, such as variations in the effective
gate length, doping concentrations, gate oxide thickness, interlayer dielectric thickness, etc. They can be broadly classified into two categories:

1. Intra-die Variations: These variations are caused due to deviations within a die. These variations are ignored in inter-die variation estimations. But, they play an important role especially in analog circuit applications.

2. Inter-die Variations: Inter-die variations are characterized by lot-to-lot, wafer-to-wafer, or die-to-die fluctuations during the manufacturing process. Inter-die variations affect all transistors in a given circuit equally and are usually much larger than intra-die variations.

For example, it shows a simplified (lumped) model of inverter in Figure 3.1.

![Figure 3.1: Simplified(lumped) model.](image)

In equation 3.1[16], $t_p$ is the propagation time of inverter, $i$ is the charging/discharging current, $v$ is the voltage over the capacitor, $v_1$ and $v_2$ are the initial and final voltages respectively, and $C_L$ is the output capacitance of the gate which is composed of the drain diffusion capacitances of the NMOS and PMOS transistors, the capacitance of the connecting wires, and the input capacitance of the fan-out gates[16]. From the expression of inverter, its
propagation would be determined by $C_L$, $i$ and $v_1 \& v_2$. Due to the process variation, circuitries’ parameter $C_L$ could be different in different dies, even in the same die, and thus their propagation time varies.

$$t_p = \int_{v_1}^{v_2} \frac{C_L(v)}{i(v)} dv$$  \hspace{1cm} (3.1)

### 3.1.2 Input Vector

Every digital circuit gate (And, Or, Inverter ...) has its different propagation delay. For each single gate this delay can be very short, maybe somewhere around nano seconds or shorter. But as the circuit grows larger and more gates are placed one after another, the delay of a pipeline stage becomes complicated[42]. Traditional timing verification techniques such as Static Timing Analysis (STA) are usually conservative or sometimes optimistic. This inaccuracy may lead to an unnecessary procrastination of time to market or even silicon failure. It is mainly due to the inability to detect false paths and handle multiple-input-transitioning effects in the timing analysis process[43]. However, in a DVS pipeline circuit, the margin from clock cycle on longest delay could be avoided more or less by lowering the operating voltage.

It has been observed that circuit delay is strongly data dependent, and it only exhibits its worst-case delay for very specific instruction and data sequences[44]. The delay time between two consecutive input vectors makes the STA estimation inaccurate. From the view of delay fault model, two-
pattern sequence is used to detect one propagation delay, which is the time slot between the input transition and the output transition in one stage of a pipeline. Two continuous input vectors determine the following two output values. Thus the time between those two is the stage propagation delay in that cycle.

Based on the classic static timing analysis, the worst case corner is considered. In a large circuit, the timing issue would be much more complicated due to different combinations of input vectors. The longest path or very long paths are triggered occasionally. It is straightforward that in a worst case scheme, most of the propagation delay times are less than the clock cycle. Therefore, it is very critical that the delay distribution is a significant parameter of a circuit’s characteristics.

### 3.1.3 Uneven Pipeline Stage

A pipelining is a technique used in the design of computers and other digital electronic devices to increase their instruction throughput[45]. Pipelining yields a reduction in the average execution time per instruction. The pipeline designer’s goal is to balance the length of each pipeline stage. If the stages are perfectly balanced, then the time per instruction on the pipelined processor in the ideal conditions is equal to

\[
\frac{\text{Time per instruction on unpipelined machine}}{\text{Number of pipe stages}}
\]
Under these conditions, the speedup from pipelining equals the number of pipe stages. However, the stages will not usually be perfectly balanced. Hence, the uneven partitioning brings variations on propagation delay between stages. And the length of a processor cycle is only determined by the time of the slowest pipeline stage, since all stages are working at the same clock domain[31].

In [46], they implemented Razor error detection and correction approach in a 64-bit Alpha processor. The processor was using a simple in-order pipeline structure consisted of instruction fetch, instruction decode, execute, memory, and writeback. But two stages, ID and EXE, are critical at the worst-case voltage and frequency setting. So the propagation delays on the other three stages are almost lower than the clock cycle.

In conclusion, all the three types of variations have large effect on stage propagation delay. The characteristics of propagation delays from different stages are various.

### 3.2 Razor’s Potential

As we have talked Razor in the last chapter, its basic idea is to apply a self-adaptive power supply to all the pipelined circuit to achieve the lower working voltage and thus the lower power consumption. The drawback of this scheme is treating all the stages with the same runtime supply voltage. But based on the observation of variations in the above section, not all stages
have the similar delay time in the clock cycle in the pipelined circuit. Some stages’ latency might be short, while the others’ might be long.

So the stages in a pipelined processor would present largely different delays in one clock cycle, for example in Figure 3.2, but Razor determines power supply voltage by the longest delay time of stages in pipeline. For those stages with a short delay time, supply voltage could be lower than that applied on stages with long propagation delay. If the lower supply voltage could be applied on the shorter stages, a further power consumption reduction would be achieved. Therefore, the basic idea of our design is that we apply lower operating voltage on the stages with the shorter delay time to achieve power reduction.

![Figure 3.2: Possible propagation delays of a five-stage pipeline in one clock cycle.](image)

3.3 Fine-Grained Razor Design

A fine-grained Razor system is proposed in this research to further reduce power consumption. Our method is to partition the pipeline stages into two groups, one for stages with the long propagation delays and the other for stages with the short delays. Each group is powered by its individual DVS power supply grid as shown in Figure 3.3. Hence, two groups of the processor pipeline can be assigned with different supply voltage values at the same time. This design results in further energy saving because different stages have different performance requirements, especially different stages can operate at their optimal energy-delay points simultaneously [47]. As shown in Figure 3.3, the proposed low-power architecture also includes timing error detectors (e.g., Razor FFs) to allow errors to occur and corrected.

Our proposed method mainly includes two parts. The first one is to get the delay distribution for each stage, and the second is to partition the pipeline stages into two groups according to their delay distributions and to apply different supply voltage to each group. Though this discussion is confined to two voltage domains, it can be generalized to any number of voltage domains as long as the hardware overhead can be justified.

3.3.1 Statistical Analysis on Propagation Delay

Delay analysis is used in integrated circuit design for the propagation delay calculation of a single logic gate and the wires attached to it. In contrast,
static timing analysis computes the delays of all the propagation paths from primary inputs to primary outputs, using delay analysis discussed above to determine the delay of each gate and wire.

There are many methods used in delay analysis for a combinational circuit, and the choice depends primarily on the simulation speed and accuracy required. The following two methods are widely used:

- Circuit simulators such as SPICE may be used. This is the most accurate, but slowest, method.

- Two-dimensional tables are commonly used in applications such as logic synthesis, placement and routing. These tables take one gate’s input slope and output load as inputs, and generate delay time and output slope for the gate and its associated wires[48].

Figure 3.3: The proposed error-tolerant DVS architecture.
In our experiments, we used HSPICE to collect propagation delays, since it is more accurate than the table look-up method, as well as it has an acceptable running time. We extracted the parameter from the layout designed in Cadence Encounter, and ran post-layout simulation for timing in HSPICE. The purpose of timing analysis is to determine the distributions of stage delays, which describe the timing variations, especially the input vector variation. To obtain stage delay distributions, we used a Monte-Carlo sampling technique, which relies on repeated random sampling to get the results. After a large number of HSPICE simulations have been run, the statistical data is obtained. Curve fitting in Matlab is then used to determine the probability density function (PDF) on circuit propagation delay for each stage. Then we will get that for the delays of N pipeline stages the PDFs are following the Gaussian distributions: \( D_1\{\mu_1, \sigma_1^2\}, D_2\{\mu_2, \sigma_2^2\}, \ldots, D_N\{\mu_N, \sigma_N^2\} \). For example, delay distributions of a pipeline processor with five stages are illustrated in Figure 3.4.

3.3.2 Normal Mode in Pipeline

First, we discuss the case that a processor working at the normal operating voltage. Suppose an N stages pipeline processor is working in normal mode, so the processor is working with normal voltage \( V_{dd} \) and normal clock period \( T_{clk} \). Generally \( T_{clk} \) is the clock cycle derived from conventional worst-case timing analysis to satisfy the constraint on the longest propagation delay of all stages. As discussed before, we assume that delays of N Stages follow
the Gaussian distributions \( D_1\{\mu_1, \sigma_1^2\}, D_2\{\mu_2, \sigma_2^2\}, \ldots, D_N\{\mu_N, \sigma_N^2\} \) without loss of generality.

Based on the 3-sigma rule, about 99.7% of values drawn from a normal distribution are within three standard deviations from the mean \( \mu \) [49]. Assume \( u_{max} \) is \( \max\{\mu_1, \mu_2, \ldots, \mu_N\} \), and \( \sigma_{max} \) is the corresponding standard deviation of the \( u_{max} \). Usually \( T_{clk} \) is determined larger than \( u_{max} + 3\sigma_{max} \) as shown in Figure 3.5 [50].

The energy consumption at normal operating voltage can be represented by Equation 3.2.

\[
E_{\text{normal}} = \alpha \cdot f \cdot C_{\text{tot}} \cdot V_{dd}^2
\]

(3.2)

where \( \alpha \) is the switching activity factor, \( f \) is the clock frequency, \( V_{dd} \) is the operating voltage, and \( C_{\text{tot}} \) is the normalized equivalent capacitance for the whole circuit.

Figure 3.4: Distributions of a five-stage pipeline.
3.3.3 Razor Mode in One Stage

Here we analyze the Razor mode at only one stage. The dependency of energy and delay on $V$ is showing in the following Equations 3.3 and 3.4, where $V$ is the operating voltage, and $C$ is the normalized equivalent capacitance of a combinational circuit. Energy is proportional to the square of the operating voltage, so as the voltage scales down, the energy consumption is significantly decreased. But the delay time is increased accordingly in the meantime. We assume all NMOS and PMOS transistors have the same absolute value of $V_{th}$. Thus there is a trade-off between power and delay time for an energy-aware
We implemented an experiment to verify Equation 3.4 in HSPICE and Matlab. We ran HSPICE simulation on benchmark circuit C880 with the operating voltage from 1.3v to 0.6v by scaling of 0.1v under the same input vector for deriving the delay times. We implemented equation 3.4 in Matlab to generate the delay times for different operating voltages with the delay time at the operating voltage of 1.3v from HSPICE as a reference. We list the data from HSPICE and Matlab in Table 3.1 and Figure 3.6.

Table 3.1: HSPICE and Matlab simulation on operating voltage and delay

<table>
<thead>
<tr>
<th>Supply Voltage(v)</th>
<th>HSPICE Results(s)</th>
<th>Model Results(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3</td>
<td>1.76e-10</td>
<td>1.76e-10</td>
</tr>
<tr>
<td>1.2</td>
<td>2.00e-10</td>
<td>1.92e-10</td>
</tr>
<tr>
<td>1.1</td>
<td>2.30e-10</td>
<td>2.13e-10</td>
</tr>
<tr>
<td>1</td>
<td>2.71e-10</td>
<td>2.43e-10</td>
</tr>
<tr>
<td>0.9</td>
<td>3.33e-10</td>
<td>2.87e-10</td>
</tr>
<tr>
<td>0.8</td>
<td>4.38e-10</td>
<td>3.60e-10</td>
</tr>
<tr>
<td>0.7</td>
<td>6.36e-10</td>
<td>5.05e-10</td>
</tr>
<tr>
<td>0.6</td>
<td>1.11e-09</td>
<td>9.12e-10</td>
</tr>
</tbody>
</table>

- Relation between $T_k$ and $k$.

In Razor design, a delay fault comes into the sight when the operating voltage decreases too much. Let us use $k$ represent the circuit correct rate. Since we have already had the delay distribution $D_{normal}\{\mu_{normal}, \sigma^2_{normal}\}$
Figure 3.6: HSPICE and Matlab simulation results for different operating voltage and delay

at the normal operating voltage for one particular circuit, by applying the cumulative distribution function of a normal distribution, we can achieve Equation 3.5 to reflect the relation between k and $T_{k}$, where k is in direct proportion to $T_{k}$. Using Equation 3.5, given a circuit correct rate k, we can get the corresponding $T_{k}$.

$$k = \int_{-\infty}^{T_{k}} \frac{1}{\sqrt{2\pi}\sigma_{normal}} e^{-\frac{(t-\mu_{normal})^2}{2\sigma_{normal}^2}} dt$$  \hspace{1cm} (3.5)

- Relation of Propagation Voltage Scaling and Delay Scaling.

In this research, we would like to explore the tradeoff between lowering
the supply voltage, and allowing a small number of errors to occur and immedi-
ately corrected, with the clock rate kept constant. The energy consumption
can be greatly reduced by supply voltage reduction, while the performance
of the circuit is not degraded if the number of errors is controlled small. The
problem is how to reduce the supply voltage to a specific value such that the
error rate can be controlled to 1-k. This problem can be solved by finding a
hypothetical clock cycle time $T_k$ that will yield 1-k of error rate under nor-
mal supply voltage $V_{dd}$. Then, we can derive reduced voltage supply $V_{dd-l}$
that yields 1-k of error rate under $V_{dd-l}$ by finding the relationship between
$(V_{dd}, T_k)$ and $(V_{dd-l}, T_{clk})$ indirectly as shown in Figure 3.8.

As the operating voltage scales down, the delay distribution curve is ex-
tended along the x axies, so the whole shape of delay distribution becomes
lower and wider. Although the shape changes when the operating voltage is
decreased, we focus on the correct rate k, which actually is the cumulative
density function (CDF) of the delay time. In Figure 3.7, the shaded area
represents the error rate (1-k). From Equation 3.4, we have

$$T_k \propto \frac{C \cdot V_{dd}}{(V_{dd} - V_{th})^{1.3}} \tag{3.6}$$

$$T_{clk} \propto \frac{C \cdot V_{dd-l}}{(V_{dd-l} - V_{th})^{1.3}} \tag{3.7}$$

where $V_{dd-l}$ donates the scaled lower voltage. Since Equations 3.6 and 3.7 are
based on the same circuit, they should have the same coefficient and the same
circuit capacitance C. Therefore we can put these two equations together to
eliminate C yielding Equation 3.8. Equation 3.8 reflects the relation between propagation delay scaling (from $T_k$ to $T_{clk}$) and voltage scaling (from $V_{dd}$ to $V_{dd-l}$). Given any particular propagation delay scaling $T_k/T_{clk}$, in Equation 3.8 we can determine $V_{dd-l}$ by giving the operating voltage $V_{dd}$ in normal mode.

\[
\frac{T_k}{T_{clk}} = \frac{V_{dd}}{(V_{dd-V_{th}})^{1.3}} = \frac{V_{dd-l}}{V_{dd}} \cdot \left(\frac{V_{dd-l} - V_{th}}{V_{dd} - V_{th}}\right)^{1.3} \tag{3.8}
\]

Here we review the above derivation that the correct rate remains the same after the delay is increased as the operating voltage scales down. We give an example. Suppose one combinational logic circuit’s delay distribution follows normal distribution $\{1.5 \times 10^{-10}, (0.5 \times 10^{-10})^2\}$ at the operating voltage of 1v. Using Matlab, we use this normal distribution to generate 100 delays, which follows the normal distribution with $\{1.44 \times 10^{-10}, (0.53 \times 10^{-10})^2\}$. However, there is a difference between the true distribution and the observed results. We apply Equation 3.8 to the 100 observed samples by assigning $V_{dd-l}=0.7v$ to get the transformed distribution following $\{2.99 \times 10^{-10}, (1.11 \times 10^{-10})^2\}$ in Table 3.2. The histogram of the original observed delay distribution is shown as delay data and the fitted curve is shown as fit 1 in Figure 3.8, while the histogram of the transformed delay distribution is shown as delay2 data and the fitted curve is shown as fit 2 in Figure 3.8. It can be observed that as the operating voltage scales down, the shape of the distribution is extended and shifted to the right.
Figure 3.7: Delay distribution shift and relationship between $T_{clk}$ and $T_k$.

Table 3.2: True distribution, generated distribution and transformed distribution

<table>
<thead>
<tr>
<th></th>
<th>Original Distribution</th>
<th>Generated Distribution</th>
<th>Transformed Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu$</td>
<td>1.5e-10</td>
<td>1.44e-10</td>
<td>2.99e-10</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>0.5e-10</td>
<td>0.53e-10</td>
<td>1.11e-10</td>
</tr>
</tbody>
</table>
Figure 3.8: Correct rate keeps as distribution is extended

Suppose we need the circuit to work at the correct rate of 0.95. So in the generated distribution, the delay $D_1$ with value equal to $2.31e-10$ ensures that area1 marked by red slash lines in Figure 3.8 is equal to 0.95. Then we get the delay $D_2$ with value equal to $4.81e-10$ on the extended distribution by applying Equation 3.8 with the delay $D_1$. So it can be verified that $D_2$ can make the area2 in Figure 3.8 equal to 0.95, which is equivalent to area1. The above derivation steps are shown in Table 3.3.

- **Energy of only one stage in Razor mode.**

In this step, we derive the energy expression of one stage circuit. As we know, to further reduce power, the supply voltage should be scaled down as
low as possible, but, lowering the supply voltage causes error rate to increase, which triggers instructions replay frequently. As a result, the gain of power reduction will be compensated. What is more, the execution time is increased too. So determining the scaled voltage is a tradeoff to ensure the pipeline work in the low power and with small number of instructions replay. In a stage with the Razor scheme, the energy consumption includes two parts. One is consumed in performing standard operations with the scaled voltage, while the other is consumed in the recovery scheme as represented in Equation 3.9.

\[ E_{\text{razor}1} = E_{\text{scaled}1} + E_{\text{rec}1} \]  

(3.9)

To simplify our model, we just use two different voltage values for the dynamic voltage scaling method, where the circuit can remain at the lower voltage \((V_{dd-l})\), or it can raise the voltage to the normal value \((V_{dd})\) after encountering a delay fault detected by Razor. The recovery scheme is operated in the following clock cycle, and the working voltage falls down to the lower one again after the recovery is done. Actually, there can be many other algorithms to implement the ideas of the voltage scaling and the pipeline recovery.

\[ D_1 = 2.31 \times 10^{-9} \text{ s} \rightarrow \text{by Equation 3.8} \]

\[ D_2 = 4.81 \times 10^{-9} \text{ s} \]
We can represent the power consumption of each process by Equations 3.10 and 3.11.

\[ E_{\text{razor1}} = \alpha \cdot f \cdot C_1 \cdot V_{dd-l}^2 \]  
\[ E_{\text{rec1}} = (1 - k) \cdot \alpha \cdot f \cdot C_1 \cdot V_{dd}^2 \]  

Taking Equations 3.10 and 3.11 into 3.16, we get

\[ E_{\text{razor1}} = \alpha \cdot f \cdot C_1 \cdot V_{dd-l}^2 + (1 - k) \cdot \alpha \cdot f \cdot C_1 \cdot V_{dd}^2 \]  

Equation 3.12 can be used to estimate the total energy consumption of a pipeline stage implemented by Razor. It has only two unknown parameters: the correct rate k and the scaled (lower) operating voltage \( V_{dd-l} \). Given a value k, we can use Equations 3.5 and 3.8 to derive the value of \( V_{dd-l} \). Thus given an ideal range of k, the corresponding range of energy consumptions can be calculated. The lowest energy consumption can be found with the corresponding k. Taking this k into Equations 3.5 and 3.8, the best \( V_{dd-l} \) for minimum energy consumption is derived.

### 3.3.4 Razor Mode on Pipelined Processor

So far we have applied Razor to only one stage, and we extend our analysis to a pipelined processor with many stages. The key point of Razor on one stage is that for each expected correct rate k, we can find a scaled operating voltage and estimate the value of energy consumption. After going through
a reasonable range of correct rate \( k \), the minimum energy consumption can be determined.

The difference of Razor on a pipelined processor from the one-stage case is that several stages lie in one pipeline. As the operating voltage scales down, the propagation delays of some stages might be larger than the fixed clock cycle, which means more than one stage could probably come up with errors in one clock cycle. Errors occurring in a three-stage pipeline processor could be illustrated in Figure 3.9. Since the events of occurrence of errors in different stages are independent, the correct rate for a pipelined processor is equal to the multiplication of the correct rate in every stage as shown in Equation 3.13.

\[
k_{\text{pipeline}} = k_1 \times k_2 \times \ldots \times k_n \quad (3.13)
\]

We bring Equation 3.5 into Equation 3.13 to achieve Equation 3.14. Within Equation 3.14, the same as Equation 3.5 of one Razor stage’s correct rate, there are only two parameters \( k_{\text{pipeline}} \) and \( T_{k-\text{pipeline}} \). Given a value of correct rate \( k_{\text{pipeline}} \), we can get the corresponding \( T_{k-\text{pipeline}} \) value. Then, we revise Equation 3.8 for the entire pipeline Razor scheme in Equation 3.15.

\[
k_{\text{pipeline}} = \int_{-\infty}^{T_{k-\text{pipeline}}} \frac{1}{\sqrt{2\pi}\sigma_1} e^{-\frac{(t-\mu_1)^2}{2\sigma_1^2}} dt \times \int_{-\infty}^{T_{k-\text{pipeline}}} \frac{1}{\sqrt{2\pi}\sigma_2} e^{-\frac{(t-\mu_2)^2}{2\sigma_2^2}} dt \times \\
\ldots \times \int_{-\infty}^{T_{k-\text{pipeline}}} \frac{1}{\sqrt{2\pi}\sigma_n} e^{-\frac{(t-\mu_n)^2}{2\sigma_n^2}} dt 
\]

\[
(3.14)
\]
Figure 3.9: Error occurrences in more than one stages

\[
\frac{T_{k\text{-pipeline}}}{T_{clk}} = \frac{V_{dd}}{V_{dd-l}} \cdot \frac{V_{dd-l} - V_{th}}{V_{dd} - V_{th}}^{1.3}
\]  

(3.15)

In a Razor system [1], the total energy consumption of a pipelined processor (\(E_{razor}\)) is composed of the energy (\(E_{scaled}\)) required to perform standard operations at the scaled voltage, and the energy (\(E_{recovery}\)) required in recovery from timing errors as shown in Equation 3.16.

\[
E_{razor} = E_{scaled} + E_{rec}
\]  

(3.16)

As we analyzed before, Razor uses one single voltage on all the stages in a pipelined processor, so \(V_{dd-l}\) is determined by the stage with the largest delay.
mean $D\{\mu_{\text{max}}, \sigma^2\}$. We expand Equation 3.16 to achieve Equation 3.17.

$$E_{\text{razor}} = \alpha \cdot f \cdot C_{\text{tot}} \cdot V_{dd-l}^2 + (1 - k_{\text{pipeline}}) \cdot \alpha \cdot f \cdot C_{\text{tot}} \cdot V_{dd}^2$$  \hfill (3.17)

where $\alpha$ is the switching activity factor, $f$ is the clock frequency, $V_{dd}$ is the operating voltage in normal mode, $V_{dd-l}$ is the scaled lower operating voltage, and $C_{\text{tot}}$ is the normalized equivalent capacitance for the whole pipeline. Similar to the method to find the minimum energy consumption for Razor with one stage, we can determine the minimum energy consumption using Equations 3.17, 3.14, and 3.15. That is, given a range of $k$, we can find the corresponding range of energy consumptions, and thus we find the minimum energy consumption.

### 3.3.5 Fine-Grained Razor Design

Theoretically, we can apply the above analysis to every stage of a pipeline, and find the perfect working voltage for each stage. But, if multiple supply voltages are implemented in a real processor, it will cost too much overhead on adding voltage regulators. So, a fine-grained solution with toleration circuit overhead is needed. In our implementation, we group the stages into two voltage islands based on the delay distributions of stages.

The characteristics of delay distributions are used to classify those logic stages into two groups powered by two supply voltages respectively. The stages with small propagation delays will yield distribution with smaller
means, whose curves are closed to the Y coordinate. These stages are more
tolerant for lower power supply. To get lower power consumption, the lower
voltage can be assigned to these stages whose delay times will become larger.
For those stages with smaller propagation delays than others, even though
their delays become longer after voltage scaling, the most of signals going
through these stages can still propagate within one clock cycle. For those
whose delays are larger than clock cycle, Razor Flip-Flops can detect them,
and Razor can reload and operate them using the normal voltage. So, the
energy consumption for a fine-grained Razor design with two power supply
grid can be represented by Equation 3.18.

\[
E_{\text{razor}} = E_{G1-\text{scaled}} + E_{G2-\text{scaled}} + E_{\text{rec}}
\]  

(3.18)

Assume there are N stages in a pipelined processor and delay distrib-
utions of these N stages approximately follow the Gaussian distributions:
\(D_1\{\mu_1, \sigma^2_1\}, D_2\{\mu_2, \sigma^2_2\}, \ldots, D_N\{\mu_N, \sigma^2_N\}\). Without loss of generality, we as-
sume \(u_1\) is the largest mean among all stages, while \(\mu_N\) is the smallest mean
with \(u_1 > u_2 > \ldots > u_N\). We divide these stages into two groups. Group
one contains the stages \(D_1\{\mu_1, \sigma^2_1\}, \ldots, D_i\{\mu_i, \sigma^2_i\}\) while group two contains
stages \(D_{i+1}\{\mu_{i+1}, \sigma^2_{i+1}\}, \ldots, D_N\{\mu_N, \sigma^2_N\}\). When building a dual power grid
system, the working voltage of each can be determined by the stage with
the longest delay in each group. We use \(V_{G1-l}\) to denote group one’s scaled
voltage, while we use \(V_{G2-l}\) to denote group two’s scaled voltage. Since the
stage delays in group one is larger than those in group two, \( V_{G1-t} \) is larger than \( V_{G2-t} \). Here we use the normal mode operating voltage \( V_{dd} \) as the recovery voltage for both group one and group two. The reason we do not specialize the other recovery voltage value which should be lower than \( V_{dd} \) is to avoid introducing more voltage value which makes much overhead on voltage regulator. Finally, the energy consumption represented in Equation 3.18 can be further refined to Equation 3.19.

To find out the best partition point, intuitively, all the possible partitions need to be considered and all their power consumptions must be calculated. Theoretically, there are \( 2^N \) group configurations corresponding to all the possible voltage combinations for \( N \) stages given two power supply grids. However, the number of meaningful partition points is only \( N-1 \), because the partition can be done only based on the delay mean values of all stages. The partition points only exist between two adjacent stages of \( u_1, u_2, \ldots, u_N \) in sequence. So it is never possible that \( \text{stage}_1 \) and \( \text{stage}_N \) are in one group, because their delay distributions are too different to share one operating voltage for the energy reduction.

Similar as the one stage Razor mode Equation 3.12, the unknown parameters include \( V_{G1-t}, V_{G2-t}, k_{G1}, \) and \( k_{G2} \). While Equation 3.20 and 3.21 yield the relation between \( V_{G1-t} \) and \( k_{G1} \), Equation 3.22 and 3.23 yield the relation between \( V_{G2-t} \) and \( k_{G2} \). For simplification, we make \( k_{G1} \) equal to \( k_{G2} \). So now Equations 3.19 to 3.24 provide a generic modeling for the two-group fine-grained Razor energy calculation, we will give an instance for the usage.
of these equations.

\[ E_{FG-razor} = E_{G1-scaled} + E_{G2-scaled} + E_{rec} \]

\[ = \alpha \cdot f \cdot (C_1 + \ldots + C_i) \cdot V_{G1-l}^2 + \alpha \cdot f \cdot (C_{i+1} + \ldots + C_n) \cdot V_{G2-l}^2 \]

\[ + (1 - k_{FG-razor}) \cdot \alpha \cdot f \cdot (C_1 + \ldots + C_n) \cdot V_{dd}^2 \]  

(3.19)

\[ k_{G1} = k_1 \times \ldots \times k_i \]

\[ = \int_{-\infty}^{T_{k-G1}} \frac{1}{\sqrt{2\pi\sigma_i}} e^{-\frac{(t-\mu_i)^2}{2\sigma_i^2}} dt \times \]

\[ \ldots \times \int_{-\infty}^{T_{k-G1}} \frac{1}{\sqrt{2\pi\sigma_i}} e^{-\frac{(t-\mu_i)^2}{2\sigma_i^2}} dt \]  

(3.20)

\[ \frac{T_{k-G1}}{T_{clk}} = \frac{V_{dd}}{V_{G1-l}} \cdot \left( \frac{V_{G1-l} - V_{th}}{V_{dd} - V_{th}} \right)^{1.3} \]  

(3.21)

\[ k_{G2} = k_{(i+1)} \times \ldots \times k_n \]

\[ = \int_{-\infty}^{T_{k-G2}} \frac{1}{\sqrt{2\pi\sigma_{(i+1)}}} e^{-\frac{(t-\mu_{(i+1)})^2}{2\sigma_{(i+1)}^2}} dt \times \]

\[ \ldots \times \int_{-\infty}^{T_{k-G2}} \frac{1}{\sqrt{2\pi\sigma_n}} e^{-\frac{(t-\mu_n)^2}{2\sigma_n^2}} dt \]  

(3.22)

\[ \frac{T_{k-G2}}{T_{clk}} = \frac{V_{dd}}{V_{G2-l}} \cdot \left( \frac{V_{G2-l} - V_{th}}{V_{dd} - V_{th}} \right)^{1.3} \]  

(3.23)

\[ k_{FG-razor} = k_{G1} \cdot k_{G2} \]  

(3.24)

For example, stage_1 is in one group, while stage_2 to stage_N are in the
second group. So the partition just exists between \( \text{stage}_1 \) and \( \text{stage}_2 \). Again, we assume the delay means of \( \text{stage}_1, \text{stage}_2, \ldots, \text{stage}_N \) are in the decreasing order without loss of generality. We further represent the energy consumption in Equation 3.25 with the above assumption.

\[
E_{FG-\text{razor}(1-2)} = E_{G1-scaled(1-2)} + E_{G2-scaled(1-2)} + E_{\text{rec}} \\
= \alpha \cdot f \cdot C_1 \cdot V_{G1-l(1-2)}^2 + \alpha \cdot f \cdot (C_2 + \ldots + C_N) \cdot V_{G2-l(1-2)}^2 \\
+ (1 - k_{FG-\text{razor}}) \cdot \alpha \cdot f \cdot (C_1 + \ldots + C_N) \cdot V_{dd}^2 \tag{3.25}
\]

where the subscript \((1-2)\) denotes that the partition is between \( \text{stage}_1 \) and \( \text{stage}_2 \).

Here we explain the recovery scheme in Fine-Grained Razor. There are two DVS power grids for group one and group two separatedly. Group one’s propagation delay is larger than group two’s propagation delay. To get approximately equal delays between group one and group two, we use \( V_{G1-l(1-2)} \) as the scaled voltage on group one, while we use \( V_{G2-l(1-2)} \) as the scaled voltage on group two. Therefore \( V_{G1-l(1-2)} \) should be larger than \( V_{G2-l(1-2)} \). The recovery voltage for both group one and group two is \( V_{dd} \) for simplification.

When error occurs in group one and no error occurs in group two, in the next clock cycle only group one will increase the operating voltage to \( V_{dd} \). If group one has no error and group two has error, group two works at \( V_{dd} \) in the next cycle. If both groups have error, both group will increase operating voltage accordingly in the next cycle, and the computation is retried.

45
In the first group, the longest stage is only \( \text{stage}_1 \), so \( D_1\{\mu_1, \sigma_1^2\} \) is used for \( k_{G1} \) in Equation 3.26.

\[
k_{G1} = \int_{-\infty}^{T_{k-G1}} \frac{1}{\sqrt{2\pi}\sigma_1} e^{-\frac{(t-\mu_1)^2}{2\sigma_1^2}} \, dt
\]  

(3.26)

For \( k_{G2} \), since group two has more than one stage, whose mechanism is similar to the correct rate calculation in pipelined Razor mode. Since the delays of stages are independent, group two’s correct rate is the multiplication of all stages’ correct rates. Thus, the correct rate of group two can be derived in Equation 3.27. The distributions of a five stage pipeline processor are illustrated in Figure 3.10. Here, the Razor stages are partitioned between stage one and two.

\[
k_{G2} = k_2 \times k_3 \times \cdots \times k_N
\]

\[
= \int_{-\infty}^{T_{k-G2}} \frac{1}{\sqrt{2\pi}\sigma_2} e^{-\frac{(t-\mu_2)^2}{2\sigma_2^2}} \, dt \times \int_{-\infty}^{T_{k-G2}} \frac{1}{\sqrt{2\pi}\sigma_3} e^{-\frac{(t-\mu_3)^2}{2\sigma_3^2}} \, dt \times \\
\cdots \times \int_{-\infty}^{T_{k-G2}} \frac{1}{\sqrt{2\pi}\sigma_N} e^{-\frac{(t-\mu_N)^2}{2\sigma_N^2}} \, dt
\]  

(3.27)

In group two, the stages can be extended to a certain degree where the clock cycle \( T_{clk} \) can just make group two working at the expected correct rate. We make \( k_{G1} \) and \( k_{G2} \) the same, and the expected correct rate \( k_{\text{group}} \) is equal to the multiplication of \( k_{G1} \) and \( k_{G2} \). While this \( k_{\text{group}} \) only represents the correct rate of one group, for a pipelined processor with two groups of its
Figure 3.10: Distributions of a five stage pipeline processor.

stages, the correct rate of the entire pipeline processor with the fine-grained Razor is represented by Equation 3.24. Further, we revise Equation 3.8 for both groups to Equations 3.28 and 3.29.

\[
\frac{T_{k-G1}}{T_{clk}} = \frac{V_{dd}}{V_{G1-l(1-2)}} \cdot \left( \frac{V_{G1-l(1-2)} - V_{th}}{V_{dd} - V_{th}} \right)^{1.3}
\]

(3.28)

\[
\frac{T_{k-G2}}{T_{clk}} = \frac{V_{dd}}{V_{G2-l(1-2)}} \cdot \left( \frac{V_{G2-l(1-2)} - V_{th}}{V_{dd} - V_{th}} \right)^{1.3}
\]

(3.29)

Taking a certain value of \( k_{\text{group}} \) into Equations 3.26 and 3.27, we can get \( T_{k-G1} \) and \( T_{k-G2} \), and then we take them into Equations 3.28 and 3.29 to get \( V_{G1-l(1-2)} \) and \( V_{G2-l(1-2)} \). So far we have all the parameters needed for Equation 3.25. Thus we have achieved the derivation of \( E_{(1-2)} \) (in fact \( E_{FG-Razor(1-2)} \) with the parameter \( k_{\text{group}} \)). Given a reasonable range for \( k_{\text{group}} \), we can find the corresponding values of \( E_{(1-2)} \). The \( k_{\text{group}} \) value
which provides the minimum energy consumption $E_{(1-2)\text{min}}$ will be selected as the best working point for that particular partition. After going through all the possible N-1 partitions, we get all the lowest energy consumptions $E_{(1-2)\text{min}}, E_{(2-3)\text{min}}, \ldots, E_{((N-1)-N)\text{min}}$ for each partition in Figure 3.11. The minimum value from $E_{(1-2)\text{min}}, E_{(2-3)\text{min}}, \ldots, E_{((N-1)-N)\text{min}}$ is the best operating point for this processor. The corresponding $k$, $V_{G1-l}$ and $V_{G2-l}$ are determined as well.

Figure 3.11: Relation between partition position and energy consumption

### 3.3.6 Design Review

The cornerstone of this research is to first develop a unique fine-grained DVS voltage island partitioning method to further reduce the power consumption of an error-tolerant pipelined circuit, maintain the circuit performance, and
Input: Netlist of N stages in pipeline($NL_1$ to $NL_N$); Normal Voltage $V_{dd}$; Tech file; Library

Output: Partition; Scaled voltage $V_{G1-l}$ and $V_{G2-l}$; Clock cycle $T_{clk}$

—Calculate delay distribution for each stage—

foreach $NL_i$ do
    Synthesize and place & route $NL_i$ with lib;
    Extract RC parameters for HSPICE simulation;
    for $j = 1$ to $n$ do
        Simulate $NL_i$ delay with tech file and random input vectors;
    end
    Find $T_{clk}$;
    Calculate delay Dist($NL_i$);
end

—–Find partition—–

foreach $Partition_i$ do
    for $k_{FG-razor} = 0.8$ to $1.0$ by $0.01$ do
        Find $k_{G1}$ and $k_{G2}$ by Equation 3.24;
        Group 1: Find $T_{G1-l}$ by Equation 3.20;
        Find $V_{G1-l}$ by Equation 3.21;
        Group 2: Find $T_{G2-l}$ by Equation 3.22;
        Find $V_{G2-l}$ by Equation 3.23;
        Find $E_i(k_{FG-razor})$ by Equation 3.19;
    end
    $E_i \leftarrow \min(E_i(k_{FG-razor}))$;
end

$E_{FG-razor} \leftarrow \min(E_i)$;
Find the best partition and the scaled voltage $V_{G1-l}$ & $V_{G2-l}$, which gives $E_{FG-razor}$.

Algorithm 1: Fine-grained Razor algorithm

increase the circuit reliability and manufacturing yield. Intuitively, it is not quite appropriate to apply the same supply voltage to all pipeline stages due to their discrepancy in circuit delay distribution. Applying high supply volt-
age to a pipeline stage with small critical path delay distribution is a waste in power consumption. It is a natural thinking to at least apply multiple supply voltages to the pipeline stages. The problem is how many voltage islands should be used? If the granularity of voltage island partitioning is too high, the hardware cost (e.g., power grid distribution) will be too high to be justified; however, if it is too low, the benefit of fine-grained DVS cannot be accomplished. Once the number (N) of voltage islands is determined, the next question is how to partition the M pipeline stages into N voltage islands; e.g., which stages should be included in a voltage island and what is its supply voltage?

To simplify the analysis, we answer the above questions by assuming $N = 2$ and can extend the results to the general case. The first step is to find the probability density function (pdf) of circuit delay for each pipeline stage under normal voltage $V_{dd}$, and this can be achieved by circuit delay simulation. Probability density function $pdf_i$ of a pipeline stage circuit i can be approximated by a normal distribution. The second step is to group stages and assign the corresponding voltages. In the stage grouping, given the normal operating voltage and working clock cycle, stage grouping and correct rate for minimum power consumption are exclusively determined. Thought the above discussions concentrate on a special case where only two groups are allowed and group one contains only one stage, the idea can be applied to the general case where any reasonable number of groups can be formed and each group can have any number of pipeline stages.
3.4 Gain and Overhead

3.4.1 Gain

The main idea of fine-grained Razor design is implementing two DVS in one processor. One DVS is applied on the stages with long propagation delay, and the other is applied those with short delay. So the fine grain supply voltage regulation provides suitable voltage for stages to achieve further energy reduction. Therefore the ideal condition for good power consumption is that stages in one processor are divided into two groups, and the distributions of stages in each group have the similar curves, or have the similar $\{\mu, \sigma^2\}$.

Compared with the traditional Razor method, the further power reduction in fine-grained Razor mainly comes from the group of stages with the lower delays, because the traditional Razor’s operating voltage is determined by the only one stage with the longest delay. In the fine-grained Razor this longest stage is in one group, so this group’s voltage is determined by this longest stage. Therefore we will see that one of the two groups in fine-grained Razor is using the same DVS method in the traditional Razor, while the other group is using a lower DVS for its stages with the shorter delay. Experimentally small circuitries give small delays; therefore the ideal energy saving model of a pipeline processor would contain the less stages with the large delay and the more stages with the small delay. The more parts of small
delays exist in pipeline, the further power consumption will be achieved.

$$E_{\text{razor}} = \alpha \cdot f \cdot C_{\text{tot}} \cdot V_{G1-l}^2 + (1-k) \cdot \alpha \cdot f \cdot C_{\text{tot}} \cdot V_{dd}^2$$  \hspace{1cm} (3.30)

$$E_{\text{FG-razor}} = \alpha \cdot f \cdot (C_1 + \ldots + C_i) \cdot V_{G1-l}^2 + \alpha \cdot f \cdot (C_i+1 + \ldots + C_N) \cdot V_{G2-l}^2 + (1-k) \cdot \alpha \cdot f \cdot C_{\text{tot}} \cdot V_{dd}^2$$  \hspace{1cm} (3.31)

$$E_{\text{reduced}} = E_{\text{razor}} - E_{\text{FG-razor}}$$

$$E_{\text{reduced}} = \alpha \cdot f \cdot (C_{i+1} + C_{i+2} + \ldots + C_N) \cdot (V_{G1-l}^2 - V_{G2-l}^2)$$  \hspace{1cm} (3.32)

### 3.4.2 Overhead

One major overhead imposed by fine-grained Razor design is dual voltage supply grids. The power network is one of the most challenging aspects of microprocessor design; the multiply power supply grids design would cost overhead and consideration in area, placement and grid routing[21]. There are a few papers working on the dual or multiply voltage supplies, such as [21, 51, 47] and so on. Generally, a DVS system needs the supply voltage regulator, for example in [1] the error signals from Razor flip-flop are monitored to adjusts the supply voltage through two external voltage regulators. However, the good side of our design does not need a voltage regulator, because the scaled the voltages are pre-calculated and preset for the circuit.
Another design issue is level shifter. If a gate powered by \( V_l \) is directly connected to a gate powered by \( V_h \), the voltage of the passing signal cannot have a value as high as \( V_h \). The PMOS device at the \( V_{ddh} \) gate will be weakly ON, conducting static current from supply to ground through PMOS and NMOS. This becomes a serious problem in multiple power domain CMOS circuit. A typical way of avoiding the static current at the receiver side is to insert a level shifter in the Figure 3.12 [51].

![Level shifter diagram](image)

**Figure 3.12: Level shifter [53]**

While in [21], they proved the level shifter can be avoided in their design. In our implementation, the difference between \( V_h \) and \( V_l \) is small and thus it will not cause any trouble without level shifter. The difference only brings longer propagation delay but not a value failure giving a reasonable \( V_l \). The non-zero threshold voltage of transistors avoids level shifter.

In Figure 3.13, we implemented an experiment in an inverter chain. The first two are powered by \( V_L \), and the last two are powered by \( V_H=1v \). We put
Figure 3.13: Inverter chain includes four inverters.

$V_H$ and input signal $\text{In}$ together with the different values from 1v to 0.4v with the step 0.05v. We got the results in Table 3.4. The output signal is all right at low level. When $V_L$ is greater than 0.5, Out signal has good performance at high level. But when $V_L$ is smaller than 0.5, Out signal yields very bad performance at high level. In our experiment setup, the lowest scaled voltage is 0.5673v, which is proper value avoiding level shifter.
Table 3.4: Experiment on circuit without level shifter

<table>
<thead>
<tr>
<th>$V_L$=In</th>
<th>Out(High level)</th>
<th>Out(Low level)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000E+00</td>
<td>9.7971E-06</td>
</tr>
<tr>
<td>0.95</td>
<td>1.0000E+00</td>
<td>1.9162E-05</td>
</tr>
<tr>
<td>0.9</td>
<td>9.9999E-01</td>
<td>2.7589E-05</td>
</tr>
<tr>
<td>0.85</td>
<td>9.9999E-01</td>
<td>1.3199E-04</td>
</tr>
<tr>
<td>0.8</td>
<td>9.9999E-01</td>
<td>-2.7079E-06</td>
</tr>
<tr>
<td>0.75</td>
<td>9.9998E-01</td>
<td>-1.0111E-04</td>
</tr>
<tr>
<td>0.7</td>
<td>9.9999E-01</td>
<td>9.9309E-05</td>
</tr>
<tr>
<td>0.65</td>
<td>9.9997E-01</td>
<td>1.6046E-04</td>
</tr>
<tr>
<td>0.6</td>
<td>9.9993E-01</td>
<td>2.3408E-05</td>
</tr>
<tr>
<td>0.55</td>
<td>9.9998E-01</td>
<td>5.4784E-06</td>
</tr>
<tr>
<td>0.5</td>
<td>9.9641E-01</td>
<td>8.2956E-07</td>
</tr>
<tr>
<td>0.45</td>
<td>5.7113E-03</td>
<td>-7.4795E-05</td>
</tr>
<tr>
<td>0.4</td>
<td>2.2728E-05</td>
<td>8.9207E-06</td>
</tr>
</tbody>
</table>

55
Chapter 4

Experiment Setup and Results

4.1 Experiment Setup

To verify our proposed design, a gate level circuit of 16-bit 5-stage pipelined RISC (Reduced instruction set computing) processor is implemented and analyzed. The RISC processor is implemented using the NCSU FreePDK45 standard cell library. The capacitance and resistance for transistors and interconnection are extracted into HSPICE netlist. HSPICE simulation is given in an automatic Perl environment.

4.1.1 Experiment Setup

RISC, short for Reduced instruction set computing, is a CPU design strategy based on the insight that simplified instructions can provide higher performance if this simplicity enables much faster execution of each instruction [54].
MIPS, for Microprocessor without Interlocked Pipeline Stages, is a fashion of reduced instruction set computer (RISC) instruction set architecture in the Figure 4.1 [55]. The MIPS processor we designed is in classic pipeline structure, which has five stages, Instruction Fetch (IF), Instruction Decode (ID), Execution (EXE), Memory Access (MEM), and Write Back (WB). It has the simple instruction set in the Table 4.1 and thus enhances execution time for higher performance. Instructions are in 16 bit, similar to the LC-3’s instructions.

![Figure 4.1: Pipelined MIPS, showing the five stages](image)

**a) Control Unit**

The control unit of the processor shown as follows examines the instruction opcode bits and generates eleven control signals used by the other stages
Table 4.1: Instruction set

<table>
<thead>
<tr>
<th>Instruction</th>
<th>15..12</th>
<th>11..9</th>
<th>8..6</th>
<th>5</th>
<th>4..3</th>
<th>2..0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>0001</td>
<td>DR</td>
<td>SR1</td>
<td>0</td>
<td>00</td>
<td>SR2</td>
</tr>
<tr>
<td>AND</td>
<td>0101</td>
<td>DR</td>
<td>SR1</td>
<td>0</td>
<td>00</td>
<td>SR2</td>
</tr>
<tr>
<td>NOT</td>
<td>1001</td>
<td>DR</td>
<td>SR</td>
<td></td>
<td>111111</td>
<td></td>
</tr>
<tr>
<td>BR</td>
<td>00000</td>
<td>nzp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>1100</td>
<td>000</td>
<td>BaseR</td>
<td></td>
<td>00000</td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>0010</td>
<td>DR</td>
<td></td>
<td></td>
<td>Poffset9</td>
<td></td>
</tr>
<tr>
<td>LDR</td>
<td>0110</td>
<td>DR</td>
<td>BaseR</td>
<td></td>
<td>offset6</td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>0011</td>
<td>SR</td>
<td></td>
<td></td>
<td>Poffset9</td>
<td></td>
</tr>
<tr>
<td>STR</td>
<td>0111</td>
<td>SR</td>
<td>BaseR</td>
<td></td>
<td>offset6</td>
<td></td>
</tr>
</tbody>
</table>

of the processor.

The descriptions for control signals are in Table 4.2.

Here we explain these two signal JMP and AdderCTRL[1..0] as follows.

- For signal JMP, if the current instruction is JMP, or JSR, it generates a ONE signal directly. However, if the instruction is BR, system will consider the FLAG N, Z, & P from ALU and bits n, z, p from
Table 4.2: Control Signals

<table>
<thead>
<tr>
<th>Control Signal</th>
<th>Description</th>
<th>Where to go</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP</td>
<td>Tell PC whether to jump</td>
<td>IF</td>
</tr>
<tr>
<td>RegDest</td>
<td>Select source 2 Reg &amp; destination Reg</td>
<td>ID</td>
</tr>
<tr>
<td>RegWrite</td>
<td>Control whether to write value to a Reg</td>
<td>ID</td>
</tr>
<tr>
<td>MEMtoReg</td>
<td>Control whether to select value from Memory to Reg</td>
<td>ID</td>
</tr>
<tr>
<td>ALUop[1..0]</td>
<td>Tell ALU what computation to do</td>
<td>EXE</td>
</tr>
<tr>
<td>ALUsrc</td>
<td>Select one source of ALU</td>
<td>EXE</td>
</tr>
<tr>
<td>AdderCRTL[1..0]</td>
<td>Select sources for Adder</td>
<td>EXE</td>
</tr>
<tr>
<td>MEMWREN</td>
<td>Control whether to write value to a Memory</td>
<td>MEM</td>
</tr>
</tbody>
</table>

For signal AdderCRTL[1..0], there are situations which are taken care of by Adder.

Therefore all the control signals generated by control unit are in Table 4.3.

Table 4.3: Controls signals generation

<table>
<thead>
<tr>
<th>Inst</th>
<th>bit[15..12]</th>
<th>bit 11</th>
<th>bit 5</th>
<th>ALU op</th>
<th>Reg Dest</th>
<th>ALU src</th>
<th>Reg Write</th>
<th>MEM WREN</th>
<th>MEMto Reg</th>
<th>JMP</th>
<th>Adder CRTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>AND</td>
<td>0101</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>NOT</td>
<td>1001</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>BR</td>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>JMP</td>
<td>1100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>LD</td>
<td>0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>LDR</td>
<td>0110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>ST</td>
<td>0011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>STR</td>
<td>0111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>

b) IF Stage
The module diagram is in Figure 4.3.

In the normal mode, IF stage will fetch the following instruction and send it out. However, when control signal JMP = 1, it will fetch the value from Add result instead of the following instruction. In this case, a jump or branch is performing.

c) ID Stage

The block diagram is in Figure 4.4.

Based on the MIPS architecture in [56], we combine Stage ID and WB together as ID. So RegWrite tells ID whether a write to Reg is going to do here. MemtoReg tells ID which value is selected to store in the certain register, from ALUresult or Readdata. RegDst actually maps the Regs to the right outputs.

d) EXE Stage

The block diagram is in Figure 4.5.
In EXE stage, there are two main parts, Adder and ALU. As for Adder, AdderCTRL signal controls what will be done here. Refer AdderCTRL function in the Control section before. The output of Adder, AdderResult, will go to two places, IF stage as a jump destination and MEM stage as a memory address input. For ALU part, ALUop tells ALU what arithmetic operation will be done. ALUSrc decides what sources are taken as two inputs of ALU. And the N, Z, & P represent whether the result of ALU is negative, zero, or positive.

e) MEM Stage

The block diagram is as follows in Figure 4.6. The AdderResult sent
by Adder in EXE stage goes to memory’s address. The MEMWRen signal control when to perform a memory write, and the value is from ALUResult.
f) Forwarding Scheme

Forwarding scheme is adopted to deal with Data dependence. Data hazards happen in two cases: consecutive two instructions or two instructions with another intermediate. The first case can be detected by two adjacent instructions and can be handled by forwarding from output of EXE to input of EXE or from output of MEM to input of MEM. The second one can be detected by two instructions with another intermediate, and can be handled by forwarding from output of MEM to input of EXE.

For the instruction after LOAD hazards, forwarding scheme does not work; because the data needed to be written back to register will only be ready in MEM stage. A stall signal is generated in EXE stage, and goes back to IF stage and ID stage. If there is a stall signal to IF, PC in IF will
decrease 1, so it means to fetch the last instruction again. If there is a stall signal to ID, ID will be reset in this cycle.

### 4.1.2 EDA Tools

EDA tools used in this project include VCS, Synopsys Design Compiler, Cadence Encounter, Perl, HSPICE, and Matlab.

<table>
<thead>
<tr>
<th>Tools</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsys Verilog Compiler Simulator (VCS)</td>
<td>Functional Simulation and Debugging</td>
</tr>
<tr>
<td>Synopsys Design Compiler</td>
<td>Compile and Synthesis</td>
</tr>
</tbody>
</table>
| Cadence Encounter            | 1. Place and Route  
                                  | 2. Extract for DSPF                            |
| Perl                         | 1. Convert DSPF to HSPICE file  
                                  | 2. Create automatic environment for HSPICE simulation |
| HSPICE                       | Delay and Power Simulation                       |
| Matlab                       | Run proposed Algorithm                           |

### 4.1.3 Experiment Procedure

The experiment procedure is in these steps as follows.

1. Program pipelined RISC processor in Verilog, and simulated in VCS and Quartus.

2. Compile the Verilog codes in Synopsys Design Compiler by standard cells library of NCSU FreePDK 45nm.
3. Layout the compiled Verilog Netlist in Cadence Encounter using the same standard cells library.

4. Extract the DSPF file from Encounter.

5. Use script language Perl to generate HSPICE file from DSPF file.

6. Run random patterns to each pipeline stage to get Gaussian distribution.

7. Apply our algorithm in Matlab to determine the partition, operating voltage, and error rate.

8. Simulate the whole processor in HSPICE under the Perl scripting environment for power consumption information.

4.2 Experiment Results

4.2.1 Stage Delay Distribution and Parameter Extraction

We ran HSPICE simulation by assigning 1000 random input vectors to each stage at the normal operating voltage of 1v. After getting the delay distribution of each pipeline stage, we apply curve fitting on the distributions using Matlab as shown in Figure 4.8 4.9 4.10 4.11 and 4.12. Here we use the cumulative probability function (CPF) for curve fitting, because CPF can directly depict the correct rate of each stage.
Figure 4.8: Delay distribution curve fitting of stage 1

Figure 4.9: Delay distribution curve fitting of stage 2
Figure 4.10: Delay distribution curve fitting of stage 3

Figure 4.11: Delay distribution curve fitting of stage 4
Figure 4.12: Delay distribution curve fitting of stage 5

Table 4.5: Statistics of stage delay distribution

<table>
<thead>
<tr>
<th>Delay of Stage</th>
<th>$\mu$(s)</th>
<th>$\sigma$</th>
<th>max delay(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.41e-10</td>
<td>4.48e-11</td>
<td>4.26e-10</td>
</tr>
<tr>
<td>2</td>
<td>2.12e-10</td>
<td>1.37e-10</td>
<td>7.13e-10</td>
</tr>
<tr>
<td>3</td>
<td>8.99e-10</td>
<td>2.05e-10</td>
<td>2.15e-09</td>
</tr>
<tr>
<td>4</td>
<td>5.25e-11</td>
<td>3.79e-12</td>
<td>5.92e-11</td>
</tr>
<tr>
<td>5</td>
<td>5.37e-11</td>
<td>3.41e-12</td>
<td>6.05e-11</td>
</tr>
</tbody>
</table>

Generally, when a circuit is larger and contains more transistors, its delay distribution is better matched to a normal distribution. In our pipelined processor, stage 3 has the largest number of gates, so it has the largest delay mean and matched to a normal distribution better than others as shown in Figure 4.10. The fitted normal distribution of each stage is shown in Table 4.5, and the five stages’ fitted curves are plotted together in one coordinate in Figure 4.13.
Another important work in this phase is to extract the effective capacitance for each pipeline stage, such that energy consumption estimation (using Equation 3.19) for each partition in Matlab can be accomplished. After we get the energy consumption of each stage in normal mode from simulation, $\alpha \cdot f \cdot C_i$ (i.e., the effective capacitance at stage $i$) can be calculated by dividing the energy consumption of stage $i$ by the square of the operating voltage. Table 4.6 shows the energy consumption of each stage as well as that of all stages. In Table 4.5, we have also derived the maximum delay of every stage, so we can use the maximum delay of all five stages to determine the clock cycle of the entire pipeline circuit at normal mode. The maximum delay 2.15ns of stage 3 is the largest, so 2.35ns is determined as the clock cycle of
the entire pipeline, which is approximately 1.1 times of the maximum delay of stage 3, because a timing margin needs to be added to the clock cycle.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Energy Consumption($\times 10^{-9} J$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage1</td>
<td>0.02122</td>
</tr>
<tr>
<td>Stage2</td>
<td>0.03370</td>
</tr>
<tr>
<td>Stage3</td>
<td>0.4988</td>
</tr>
<tr>
<td>Stage4</td>
<td>0.004653</td>
</tr>
<tr>
<td>Stage5</td>
<td>0.004841</td>
</tr>
<tr>
<td>Normal Mode (total energy)</td>
<td>0.5632</td>
</tr>
</tbody>
</table>

### 4.2.2 Stage Grouping

In Figure 4.13, fit1, fit2, fit4, and fit5 are more crowded together, so we can expect that stage 1, 2, 4, and 5 can be clustered as one group, while stage 3 becomes the other group by itself. We verified this prediction by executing the partitioning part of Algorithm 1 using Matlab. We enumerated all different partitions with their energy consumptions (using Equation 3.19) estimated by Matlab as shown in Figure 4.14. As expected, the grouping method, which puts stage 3 in one group and stages 1, 2, 4, and 5 in the other group, yields the best energy saving. It should be noted that the energy consumption marked Razor was approximated using Equation 3.17. We emphasize that the data shown in Figure 4.14 is obtained using Matlab such that the partitioning process can be achieved efficiently. We can also expect discrepancy between data obtained by Matlab and data obtained by HSPICE. For example, the partition of (3) and (1, 2, 4, 5) shown in Figure
4.14 consumes about $0.3057 \times 10^{-9} J$ energy by Matlab, but the minimum energy consumed by this partition is $0.2294 \times 10^{-9} J$ as shown in Table 4.7 by HSPICE.

![Figure 4.14: Minimum energy consumption ($\times 10^{-9} J$) on each partition](image)

**4.2.3 Verification of Proposed Method by HSPICE**

Once the best partition has been obtained using Matlab analysis with scaled voltage values determined, it is interesting to compare the results with those obtained from HSPICE. As shown in Table 4.7, we applied two scaled voltage values $V_{lh}$ and $V_{ll}$ (two low-voltage domains) obtained from Matlab analysis, and repeated the partitioning part of Algorithm 1 with power analysis performed by HSPICE, instead of using Matlab (Equation 3.19). The column of $E_{FG-Total}$ shows the energy consumption for running the input patterns simulated by HSPICE. The energy saving presented in the last column is done by comparing $E_{FG-Total}$ and $E_{Razor}$ (divided by $E_{Razor}$). We have observed
that the best $k$ value from Matlab is 0.97 giving 4.3% power saving, which is different from the $k$ value 0.98 by HSPICE giving 4.1% power saving. However, the difference is minor and the $V_{th}$ ($V_{ll}$) values determined by Matlab and HSPICE are quite close.

Table 4.7: Simulation result on processor circuit version 1

<table>
<thead>
<tr>
<th>Correct Rate $k$</th>
<th>$V_{th}(V)$</th>
<th>$V_{ll}(V)$</th>
<th>$E_{FG-RZ}(\times10^{-9}J)$</th>
<th>Power Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.99</td>
<td>0.7529</td>
<td>0.5822</td>
<td>0.2307</td>
<td>3.6%</td>
</tr>
<tr>
<td>0.98</td>
<td>0.7406</td>
<td>0.5751</td>
<td>0.2294</td>
<td>4.1%</td>
</tr>
<tr>
<td>0.97</td>
<td>0.7329</td>
<td>0.5707</td>
<td>0.2316</td>
<td>3.2%</td>
</tr>
<tr>
<td>0.96</td>
<td>0.7271</td>
<td>0.5673</td>
<td>0.2477</td>
<td>-3.5%</td>
</tr>
<tr>
<td>0.95</td>
<td>0.7224</td>
<td>0.5646</td>
<td>0.2519</td>
<td>-5.3%</td>
</tr>
</tbody>
</table>

Another important observation from Table 4.7 is that the maximum energy reduction is only 4.1%, which is not good enough. However, it is a natural consequence of the circuit because stage 3 consumes the majority of energy as shown in Table 4.6 and dilutes the energy saved by the other stages which are clustered as another voltage island. In order to demonstrate the feasibility of this work, we tried to reduce the dominating effect of stage 3 by incrementally removing its size. First, the adder is removed from the ALU and the experiment is repeated. As shown in Table 4.8, the energy saving is increased to 7.7%. Further, the logic AND circuits are removed from the ALU and the energy saving is increased to 9.1% as shown in Table 4.9.

To further demonstrate the potential of our method, we build up another experiment using ISCAS85 benchmark circuit. We implement 5 stage pipeline circuit where the third stage is C880 and all the others are C499.
After going through all the steps above, we finally get the simulation result as good as the power saving of 41%. We can see the reason in Table 4.11. The stage3’s average delay is larger than the other stages, but the power it consumes takes about 1/3 of total power. Therefore when we lower the voltage on the other stages, the further power consumption is significant. Table 4.10 also demonstrates that Razor ($E_{RZ-Total}$) saves about 60% of energy consumption over the design without any power-saving ($E_{Normal}$), and our method further saves about 50% of energy consumption over Razor. Thus, our method can save about 80% of energy consumption over $E_{Normal}$ in this example. This experiment further demonstrates the feasibility and potential of the proposed error-tolerant DVS low-power design method for pipeline circuits.
### Table 4.10: Simulation result on benchmark circuits

<table>
<thead>
<tr>
<th>Correct Rate k</th>
<th>$V_{th}(V)$</th>
<th>$V_{ll}(V)$</th>
<th>$E_{\text{Normal}}$ ($\times 10^{-9}J$)</th>
<th>$E_{\text{RZ-Total}}$ ($\times 10^{-9}J$)</th>
<th>$E_{\text{FG-Total}}$ ($\times 10^{-9}J$)</th>
<th>Energy Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.99</td>
<td>0.7661</td>
<td>0.5854</td>
<td>5.3746</td>
<td>2.8391</td>
<td>1.3948</td>
<td>36.6%</td>
</tr>
<tr>
<td>0.98</td>
<td>0.7346</td>
<td>0.5788</td>
<td>5.3746</td>
<td>2.5168</td>
<td>1.2628</td>
<td>42.6%</td>
</tr>
<tr>
<td>0.97</td>
<td>0.7151</td>
<td>0.5746</td>
<td>5.3746</td>
<td>2.3309</td>
<td>1.1858</td>
<td>46.1%</td>
</tr>
<tr>
<td>0.96</td>
<td>0.7008</td>
<td>0.5716</td>
<td>5.3746</td>
<td>2.2011</td>
<td>1.1319</td>
<td>48.6%</td>
</tr>
<tr>
<td>0.95</td>
<td>0.6893</td>
<td>0.5691</td>
<td>5.3746</td>
<td>2.2077</td>
<td>1.1275</td>
<td>48.8%</td>
</tr>
</tbody>
</table>

### Table 4.11: Experiment on benchmark circuits

<table>
<thead>
<tr>
<th>Stage</th>
<th>Delay Avg.(s)</th>
<th>Delay Std. Dev.(s)</th>
<th>Energy in Normal ($\times 10^{-9}J$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage1: C499</td>
<td>5.834E-10</td>
<td>7.068E-11</td>
<td>0.8467</td>
</tr>
<tr>
<td>Stage2: C499</td>
<td>5.834E-10</td>
<td>7.068E-11</td>
<td>0.9574</td>
</tr>
<tr>
<td>Stage3: C880</td>
<td>1.0074E-9</td>
<td>2.4574E-9</td>
<td>1.7171</td>
</tr>
<tr>
<td>Stage4: C499</td>
<td>5.834E-10</td>
<td>7.068E-11</td>
<td>0.8884</td>
</tr>
<tr>
<td>Stage5: C499</td>
<td>5.834E-10</td>
<td>7.068E-11</td>
<td>0.9649</td>
</tr>
</tbody>
</table>
Chapter 5

Conclusion and Future Work

This thesis an error-tolerant DVS design method with the objectives of reducing power consumption and increasing circuit reliability without sacrificing circuit performance. The reliability is enhanced by allowing errors to occur and corrected, while the power saving is accomplished by error-tolerant design and DVS. Based on the pdf of each pipeline stage, the proposed method can efficiently partition the entire pipeline circuit into several voltage islands with the corresponding scaled down voltages identified. Experimental results obtained by HSPICE simulation demonstrate that the proposed method can save about 50% of energy (with only two low-power rails) over the traditional Razor design, when the pipeline clock cycle is dominated by pipeline stages that do not dominate the energy consumption. The limitation of this work is that the power saving might be small if the pipeline stages have been well balanced in delay distributions.
There are several major issues that must be further researched based on the results we have obtained so far. The first one is process variation issue. In our future research, we can improve the dynamic voltage scaling algorithm with small hardware overhead to deal with process variation on-line. For example, After voltage islands of the entire pipeline have been identified, some pipeline stages can change their assigned voltages to higher ones to make the circuits work faster, whenever the corresponding error detectors detect too many errors. This can be done by implementing counters to monitor the pipeline stages, and the counters can be turned off to save power when the machine has been well calibrated.

The other problem to be researched is DVS pipeline circuit testing, especially delay testing under different voltages for each pipeline stage. Though it is possible to perform delay testing for each pipeline stage by repeatedly applying the same test patterns under different voltage values, the test cost is too high to be used. Therefore, another more efficient method must be found to solve this problem. Delay testing is also very important for process variation and voltage calibration. If a circuit has too many delay faults due to process variation and these faults are not detected, the voltage calibration process discussed above might be extremely inefficient, or even cannot be done. Further, the delay fault test results under different voltage values can be fed-back to the voltage calibration process to find an optimal tuning method which can be fast and accurate.
Bibliography


[34] “http://www.edn.com/article/CA6301624.html”.


[40] Z. Zhang and Z. Guo, “Active Leakage Control with Sleep Transistors and Body Bias”, UCB internal, unpublished work.


82