


12-2012

Design and Analysis of an Adaptive Asynchronous System Architecture for Energy Efficiency

Brent Michael Hollosi
University of Arkansas, Fayetteville

Follow this and additional works at: <http://scholarworks.uark.edu/etd>

 Part of the [Electrical and Electronics Commons](#), [Systems and Communications Commons](#), and the [VLSI and Circuits, Embedded and Hardware Systems Commons](#)

Recommended Citation

Hollosi, Brent Michael, "Design and Analysis of an Adaptive Asynchronous System Architecture for Energy Efficiency" (2012). *Theses and Dissertations*. 654.
<http://scholarworks.uark.edu/etd/654>

This Dissertation is brought to you for free and open access by ScholarWorks@UARK. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu, ccmiddle@uark.edu.

**DESIGN AND ANALYSIS OF AN ADAPTIVE ASYNCHRONOUS SYSTEM
ARCHITECTURE FOR ENERGY EFFICIENCY**

**DESIGN AND ANALYSIS OF AN ADAPTIVE ASYNCHRONOUS SYSTEM
ARCHITECTURE FOR ENERGY EFFICIENCY**

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Engineering

By

Brent Hollosi
University of Arkansas
Bachelor of Science in Computer Engineering, 2006
University of Arkansas
Master of Science in Computer Engineering, 2008

December 2012
University of Arkansas

ABSTRACT

Power has become a critical design parameter for digital CMOS integrated circuits. With performance still garnering much concern, a central idea has emerged: minimizing power consumption while maintaining performance. The use of dynamic voltage scaling (DVS) with parallelism has shown to be an effective way of saving power while maintaining performance. However, the potency of DVS and parallelism in traditional, clocked synchronous systems is limited because of the strict timing requirements such systems must comply with. Delay-insensitive (DI) asynchronous systems have the potential to benefit more from these techniques due to their flexible timing requirements and high modularity. This dissertation presents the design and analysis of a real-time adaptive DVS architecture for paralleled Multi-Threshold NULL Convention Logic (MTNCL) systems. Results show that energy-efficient systems with low area overhead can be created using this approach.

This Dissertation is approved for recommendation to the Graduate Council.

Dissertation Director:

Dr. Jia Di

Dissertation Committee:

Dr. H. A. Mantooth

Dr. Scott C. Smith

Dr. James P. Parkerson

DISSERTATION DUPLICATION RELEASE

I hereby authorize the University of Arkansas Libraries to duplicate this Dissertation when needed for research and/or scholarship.

Agreed _____
Brent Hollosi

Refused _____
Brent Hollosi

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Jia Di, for the opportunity to work with this and other new and exciting technologies. In addition, I would like to thank him for his technical and editorial advice during this work's creation. I would like to thank all my friends and colleagues who provided encouragement and inspiration along the way. Lastly, I would like to thank my mom, Kathy Hollosi, whose tireless support has been a blessing beyond measure.

TABLE OF CONTENTS

1. Introduction.....	1
1.1 Problem.....	1
1.2 Thesis Statement	3
1.3 Approach.....	4
1.4 Organization of this Thesis	5
2. Background	6
2.1 Symbolically Incomplete Logic.....	6
2.2 NULL Convention Logic (NCL)	6
2.2.1 NCL Gates.....	8
2.2.2 NCL Registers.....	13
2.2.3 Timing Considerations	19
2.3 MTNCL	19
2.3.1 Timing Considerations	22
2.4 Dynamic Voltage Scaling (DVS).....	23
2.4.1 Literature Review.....	25
2.5 Parallelism	27
3. Architecture	29
3.1 High-Level Design.....	29
3.2 Core Design	29
3.3 Parallelism Design	31
3.4 Voltage Control Unit (VCU) Design.....	34

3.4.1 Vref Generator.....	34
3.4.2 Voltage Regulator.....	39
3.4.3 Voltage Selector	41
3.4.3.1 Design Style Analysis	41
3.4.3.2 High-Level Design.....	45
3.4.3.3 System Profiling.....	48
4. Methodology, Results and Analysis	57
4.1 Methodology	57
4.1.1 Testbench	59
4.1.2 Simulation Procedure.....	59
4.2 Energy Results and Analysis.....	60
4.3 Area Results and Analysis.....	67
5. Conclusions.....	70
5.1 Summary	70
5.2 Future Work	71
References	72

LIST OF TABLES

Table 1: Dual-Rail Encoding	7
Table 2: Quad-Rail Encoding	8
Table 3: 27 Fundamental NCL Gates	9
Table 4: Voltage Regulator Device Sizes	40
Table 5: System 1 Fullness	49
Table 6: System 2 Fullness	49
Table 7: System 3 Fullness	50
Table 8: System 4 Fullness	50
Table 9: System 5 Fullness	50

LIST OF FIGURES

Figure 1: System Approach	4
Figure 2: TH23	10
Figure 3: TH33w2	10
Figure 4: TH22d	11
Figure 5: TH12b	11
Figure 6: TH22 Transistor-Level Schematic	12
Figure 7: TH22d Transistor-Level Schematic	12
Figure 8: TH22n Transistor-Level Schematic	13
Figure 9: Single Bit Dual-Rail Register	14
Figure 10: 4-Bit Dual-Rail Register	15
Figure 11: COMP Block	16
Figure 12: NCL Pipeline Stage	16
Figure 13: 8-bit 3-Ring Register	18
Figure 14: Original Architecture of an MTNCL Pipeline with FEC	20
Figure 15: (a) Block Reduction from MTCMOS Power Gating, etc.	21
Figure 16: Current Architecture of an MTNCL Pipeline with EC	22
Figure 17: 4×4 Pipelined MTNCL Multiplier	30
Figure 18: Multiplier Chaining	31
Figure 19: Generic Sequencer	31
Figure 20: Generic Demultiplexer	32
Figure 21: Generic Multiplexer	33
Figure 22: VCU Block Diagram	34

Figure 23: Original V_{ref} Generator	35
Figure 24: V_{ref} Generator	36
Figure 25: V_{DD} Distribution Example	38
Figure 26: Processing Signal Computation	38
Figure 27: Voltage Regulator Schematic	39
Figure 28: V_{ref} Selector Enable	44
Figure 29: Ideal Modulate, QD1, QD2 Relationship	44
Figure 30: VerilogA Delay Block	45
Figure 31: V_{ref} Selector High-Level Design	47
Figure 32: V_{ref} Selector Combinational Circuit	48
Figure 33: Fullness Variance at 1.2V	52
Figure 34: Fullness Variance at 0.53V	52
Figure 35: Max Data Rate's Fullness Effect on Empty System	55
Figure 36: Min Data Rate's Fullness Effect on Full System	55
Figure 37: Modulate Equations	56
Figure 38: Design Flow	57
Figure 39: Simulation Testbench	60
Figure 40: Execution Time	61
Figure 41: Energy Usage by Voltage Domain	61
Figure 42: Energy per Computation	62
Figure 43: Control Domain Current at 1.2 V	64
Figure 44: Control Domain Current at 0.53 V	64
Figure 45: Energy Usage by Voltage Domain post Level-Converters	65

Figure 46: Energy per Computation post Level-Converters	66
Figure 47: Parallelism Area	67
Figure 48: Core Area	68
Figure 49: Control Area.....	68

1. Introduction

1.1 Problem

Power has become a primary design constraint for digital CMOS systems. This has emerged largely from two factors: transistor feature size reduction into deep sub-micron regions and the proliferation of energy-limited applications such as mobile multimedia devices, biomedical monitoring devices, and distributed sensor networks. As process feature size has reduced, dynamic power has shown only a linear increase, but leakage power and thermal output have shown exponential increases. Leakage power for some systems now comprises over 50% of the total power consumption as opposed to 1-2% seen in earlier decades [1]. As power distribution has not scaled with process size, at a cost level, circuits have become significantly more expensive to run and cool. The proliferation of mobile devices has helped make this increase in power and cost apparent and has created a need for devices that use energy as efficiently as possible to provide the most cost-effective solution for a given application. With semiconductor processes continuing to follow Moore's Law and mobile devices' growth increasing yearly, power will continue to be a critical concern in the near future.

In recent years, there are many power reduction techniques that have emerged as promising solutions to alleviate the power problem. One technique, the combination of parallelism and dynamic voltage scaling (DVS), has been effective in this area allowing for power savings while maintaining throughput at the cost of area. Parallelism is a technique where multiple instances of a computational core are set up to run concurrently; DVS is a power management technique where the supply voltage of a system is scaled in accordance with a certain power usage strategy. These techniques, in combination, have been well-studied and applied to the prevailing clocked, synchronous systems. However, asynchronous systems employing this technique show several disadvantages compared to delay-

insensitive, asynchronous systems like those designed using NULL Convention Logic (NCL) [2]:

- 1) For synchronous systems, the combination of V_{DD} scaling, clock frequency scaling, and process variation requires large timing margin and sophisticated timing analysis. NCL circuits, on the other hand, are correct-by-construction; they work correctly as long as the transistors switch properly. Therefore very little, if any, timing analysis is needed. This feature allows NCL circuits to employ DVS more aggressively, across a wider range of voltages, and with higher reliability than the synchronous counterparts;
- 2) The handshaking signals of NCL circuits naturally indicate the input data rate and current workload, which enables real-time adaptive DVS and substantially simplifies the V_{DD} control algorithm compared to synchronous counterparts;
- 3) The delay-insensitivity of NCL circuits automatically guarantees no input data will be lost during the rise of V_{DD} when a new burst of input arrives, where in synchronous circuits careful timing analysis is required to achieve this feature;
- 4) The robustness against process variations facilitates the implementation of parallelism in NCL circuits for further V_{DD} scaling and power reduction, where each processing unit can have its own individual V_{DD} scaling mechanism without the need to consider global timing;
- 5) The high modularity feature of NCL circuits allows for much easier turning on/off of parallel processing units for enhanced system scalability; and
- 6) Synchronous systems generally consume more active energy and leakage power than NCL counterparts, especially the most recent advance of this methodology – the Multi-Threshold NCL (MTNCL) [3].

Furthermore, it has recently been stated by Intel that current technology trends have prompted them to change their low power design methodology [4]. In the past, Intel has created designs specifically tailored for each application type: server, desktop, mobile, and medical implantable devices. Now they are looking into creating only 1-2 designs capable of operating across all applications. In order to do this, the following four features are highly desirable: 1) aggressive power management; 2) adaptive to workload, environment (temperature change), and aging; 3) resistant to errors during voltage/frequency changes; and 4) maximize(minimize) process advantages (disadvantages).

While the fourth feature, to a large degree, must still be handled by physical-level designers, the first three features are achievable by using delay-insensitive asynchronous logic with little to no timing analysis and overhead. Feature one and three are inherent features of NCL/MTNCL systems due to their timing insensitivity; NCL/MTNCL systems can have their supply voltages scaled across a large, continuous range in real-time without incurring computation errors. The second feature is achievable through the handshaking signals existent in every NCL/MTNCL system; changes in workload, environment, and aging are all detectable through the monitoring of these handshaking signals. Because of these reasons, NCL/MTNCL systems implementing parallelism and DVS can achieve the type of universal operation Intel introduced. Such systems can be easily customized for various applications in order to achieve the most energy efficient solution.

1.2 Dissertation Statement

The goal of this dissertation is to develop and explore an adaptive system architecture which uses MTNCL, parallelism, and DVS to create energy-efficient systems.

1.3 Approach

The approach is to build a design flow that for any MTNCL core with n dual-rail inputs, m dual-rail outputs, 1 K_i input, 1 $Sleep$ input, 1 $Reset$ input, and 1 K_o output, instances a user-specified number of cores and builds the surrounding circuitry to accommodate the parallelism and implement DVS. Figure 1 shows an example of such systems.

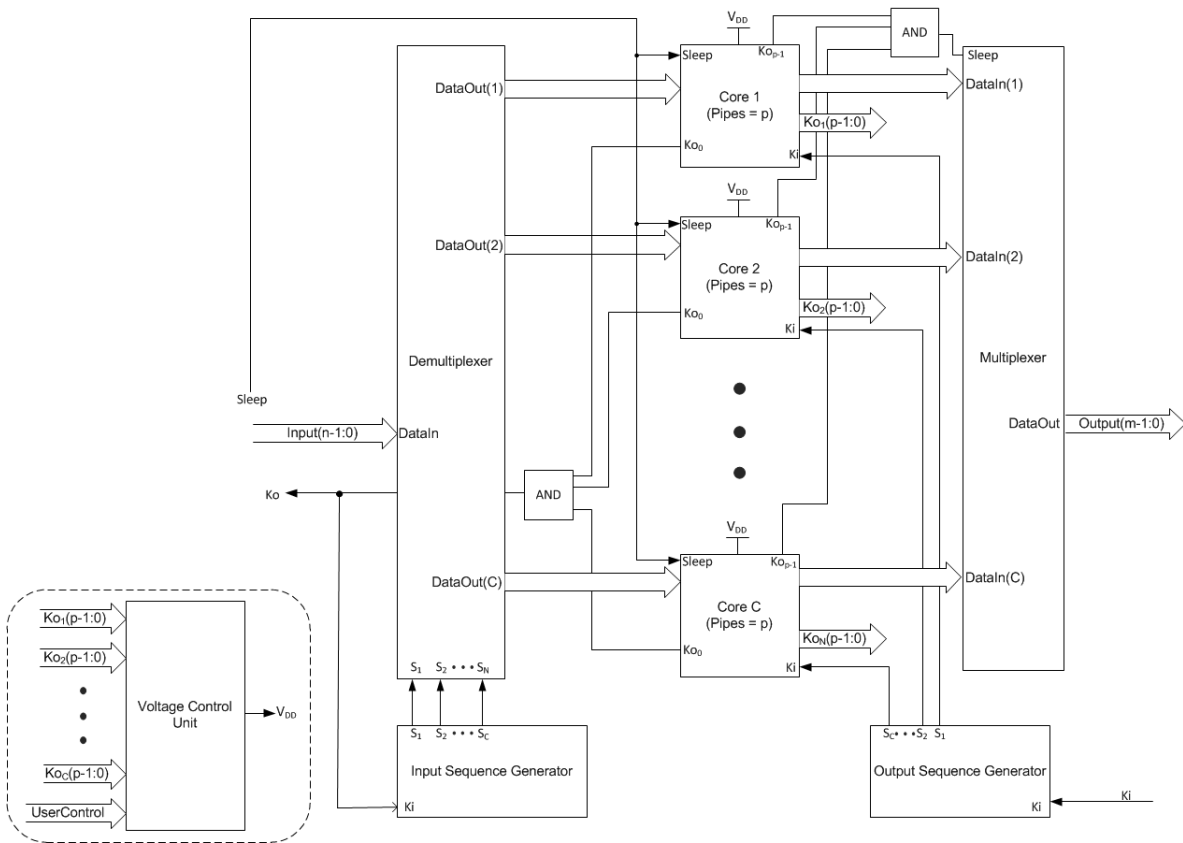


Figure 1. System Approach

To implement parallelism, the Demultiplexer and Input Sequence Generator dispatch incoming DATA to the proper Core while the Multiplexer and Output Sequence Generator guarantee the proper DATA exits the system. To implement DVS, a voltage control unit (VCU) is added. The VCU takes as inputs the handshaking signals (K_o 's) from each Core. These signals,

which are used to coordinate the MTNCL circuit's behavior by controlling the DATA/NULL wavefront alternation, also indicate a Core's fullness, i.e., how many DATA the core is processing. The VCU uses this information along with a *UserControl* input to regulate the Cores' supply voltage. The *UserControl* input is incorporated in order to allow the user to specify system constraints, i.e., how full the system should attempt to be at all times. If the user specifies that the system should try to be empty, the VCU tunes the Cores' V_{DD} to higher values facilitating higher throughput. On the other hand, if the user specifies that the system should try to be full, the VCU tunes the Cores' V_{DD} to lower values facilitating lower power. Furthermore, when the input data rate is low, e.g., the system is idle, the VCU aggressively scales the Cores' V_{DD} deep into the sub-threshold regime to reduce leakage power. As the Cores begin to fill with DATA, the VCU raises the Cores' V_{DD} in anticipation of the incoming workload.

Due to MTNCL's advantages, the value of V_{DD} can be scaled with very high precision in real-time, allowing for systems with near-optimal energy efficiency or near-optimal throughput as required by the application. Even higher levels of energy/performance precision could be obtained with features like giving each Core its own V_{DD} domain or implementing real-time disabling/enabling of Cores to further reduce leakage power or handle an increased workload, respectively.

1.4 Organization of this Dissertation

Chapter 2 contains background information on several key concepts of the system: NCL, MTNCL, DVS, and parallelism. Chapter 3 contains a detailed description of the system. Chapter 4 contains the simulation methodology and results and analysis of simulations for the system. Chapter 5 summarizes the main concepts learned, discusses the potential impact of this information, and explores future avenues for this work.

2. Background

2.1 Symbolically Incomplete Logic

Synchronous circuit designs can be separated into two units: datapath and control. The datapath unit's responsibility is to process data; the control unit's responsibility is to validate processed data. Control units use a time-dependent signal to affirm the validity of the datapath's data. This requires datapath elements' delays to be bounded by the time-dependent signal. This timing dependency between datapath and control exists because Boolean logic, which is used to implement the datapath elements, consists only of values representing data.

Boolean logic consists of two logic values: True and False. These values represent data, so processing sequences of these values yields only more data; there is no value in the logic that indicates the data's validity. Validity can only be obtained through the use of a time-dependent reference which uses the same values of the logic (True/False). As long as the data process and reference are synchronized, the reference's periodic oscillation between the two logic values can be used to indicate when data is valid and when it is not. A logic that does not contain explicit values that indicate data's validity is a main characteristic of a *symbolically incomplete* logic and the prime difference between Boolean logic and NCL [2].

2.2 NULL Convention Logic (NCL)

NCL is a symbolically complete logic which means that its logic set includes values for data processing and data validation. Data validation is made possible through the inclusion of a non-data (NULL) value along with the standard Boolean values. In this way, invalid data can be represented through the use of the non-data (NULL) value, and valid data can be represented through the use of the original Boolean data values.

A wire in traditional digital circuitry uses only two voltage levels to represent a logic value. These two voltage levels are typically represented as logic1 and logic0 and can be mapped directly to Boolean logic's two values: True = logic1; False = logic0. However, NCL requires three logic values: one non-data value (NULL) and the two data values. Thus, at least two wires are used to encode the three NCL logic values. The two-wire representation, denoted as *dual-rail* encoding, of the logic values is shown in the table below. The two wires cannot be logic 1 at the same time, which is considered as an invalid state.

	NULL	DATA 0	DATA 1	INVALID
Wire 0	0	1	0	1
Wire 1	0	0	1	1

Table 1. Dual-Rail Encoding

DATA 0 and DATA 1 can be used to represent any mutually exclusive two-value relationship, like Boolean logic's True/False relationship. However, NCL logic sets are not restricted to two data values. They can contain n data values, where $n \geq 1$, requiring n wires for encoding. For example, *quad-rail* encoding, shown in Table 2, uses four data values which can be used to represent a mutually exclusive four-value relationship like two paired Boolean signals.

Since mutual exclusivity is not inherent to an encoding scheme's n data values, by convention, the n data values of any encoding scheme are asserted in mutual exclusivity of each other. Assertion of multiple data values at the same time is, in nearly all cases, prohibited.

	NULL	DATA 0	DATA 1	DATA 2	DATA 3	INVALID
Wire 0	0	1	0	0	0	1
Wire 1	0	0	1	0	0	1
Wire 2	0	0	0	1	0	1
Wire 3	0	0	0	0	1	1

Table 2. Quad-Rail Encoding

An NCL signal consists of n wires which represent n data values; this type of signal is called an n -rail signal. Wire denotation is done in two ways: 1) name.railX, where name is the signal's name and X identifies the rail/wire number; and 2) name^X. For example, $A.rail0$ and A^0 both reference *Wire 0* or *rail 0* of signal A. Bus notation is done in the following two ways: $A(i).rail0$ or A_i^0 where i indexes a bus's signal.

2.2.1 NCL Gates

In NCL signals, a single wire can express only one data; the voltage level of a single wire indicates that it is representing data, but it does not indicate which data value is represented. As such a gate processing these signals cannot determine a signal's actual data value; it can only determine the quantity of data being presented to it. A gate that processes signals in this manner is called a threshold gate, the threshold being the number of inputs that need to be logic1 for the output of the gate to become logic1. NCL gates use hysteresis such that the output of the gate counts as a *threshold-1* number of inputs. Once the gate's threshold is met, the gate's asserted output will be sustained until all inputs to the gate are deasserted. A gate with these properties satisfies the process definition of NCL and provides a fundamental and scalable unit for processing logic values.

Threshold gates have two major naming conventions depending on whether or not the inputs to the gate are weighted. All threshold gates with non-weighted inputs use the form $THmn$ such that $1 \leq m \leq n$. n is the number of inputs to the gate, and m is the threshold of the gate. Threshold gates with weighted inputs, also called weighted threshold gates, use the form $THmnWx_1x_2\dots x_L$. L is as an integer value where $1 \leq L < n$, and the weight of input L is an integer value such that $1 < x_L \leq m$. These two types of gates, as well as a few others, comprise the 27 fundamental NCL gates. These 27 gates, shown in Table 3, are capable of expressing any Boolean function of four variables or less [2].

Combinations of these 27 gates can be used to design any combinational circuit; however, a few other gate types are needed for control and storage elements: resetting and inverting. Resetting gates are typically of the form $THnn$, and their reset state is denoted by either a d , signifying reset to logic1, or an n , signifying reset to logic0. The d or n is placed after the $THnn$. For example, the name TH22d would be used represent a TH22 gate capable of resetting to logic1. Inverting gates are typically of the form $THIn$ and are denoted by a b placed after the $THIn$.

NCL Gate	Boolean Function
TH12	$A + B$
TH22	AB
TH13	$A + B + C$
TH23	$AB + AC + BC$
TH33	ABC
TH23w2	$A + BC$
TH33w2	$AB + AC$
TH14	$A + B + C + D$
TH24	$AB + AC + AD + BC + BD + CD$
TH34	$ABC + ABD + ACD + BCD$

TH44	$ABCD$
TH24w2	$A + BC + BD + CD$
TH34w2	$AB + AC + AD + BCD$
TH44w2	$ABC + ABD + ACD$
TH34w3	$A + BCD$
TH44w3	$AB + AC + AD$
TH24w22	$A + B + CD$
TH34w22	$AB + AC + AD + BC + BD$
TH44w22	$AB + ACD + BCD$
TH54w22	$ABC + ABD$
TH34w32	$A + BC + BD$
TH54w32	$AB + ACD$
TH44w322	$AB + AC + AD + BC$
TH54w322	$AB + AC + BCD$
THxor0	$AB + CD$
THand0	$AB + BC + AD$
TH24comp	$AC + BC + AD + BD$

Table 3. 27 Fundamental NCL Gates

Figures 2-5 show symbols for examples of the gate types mentioned in the preceding section.

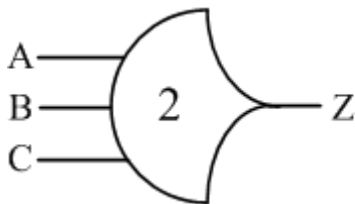


Figure 2. TH23

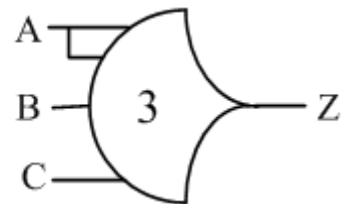


Figure 3. TH33w2

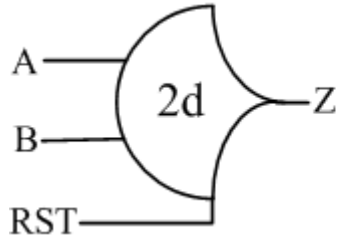


Figure 4. TH22d

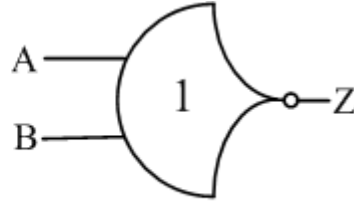


Figure 5. TH12b

Nearly all threshold gates contain four major blocks. The name of each block corresponds with its effect on the gate's output. From previous work [24], the four blocks are named the following: *Reset*, *Hold0*, *Set*, and *Hold1*. The *Set* and *Hold0* blocks are complements of each other, as are the *Reset* and *Hold1* blocks. The *Set/Reset* block is responsible for changing the gate's output from a logic0/logic1 to a logic1/logic0. The *Hold1/Hold0* block is responsible for holding the output logic1/logic0 once it has been changed to logic1/logic0 by the *Set/Reset* block. The transistors used for hysteresis are placed in series with the *Hold* blocks. An example of these blocks' arrangement in a common NCL gate is shown in Figure 6.

The *Set* block contains PMOS transistors ordered to represent its corresponding Boolean function. The *Hold1* block is a parallel combination of PMOS transistors driven by each gate input, respectively. The *Reset* block is a series combination of NMOS transistors driven by each gate input, respectively. The *Hold0* block is a complementary ordering of the *Set* block. These blocks output is inverted in order for the gate to function properly in a CMOS setting. All NCL gates can be created by following these block descriptions; however, some term combining between blocks can yield transistor savings. Resetting and inverting gates use the same block arrangement, but require slight modifications to achieve their function.

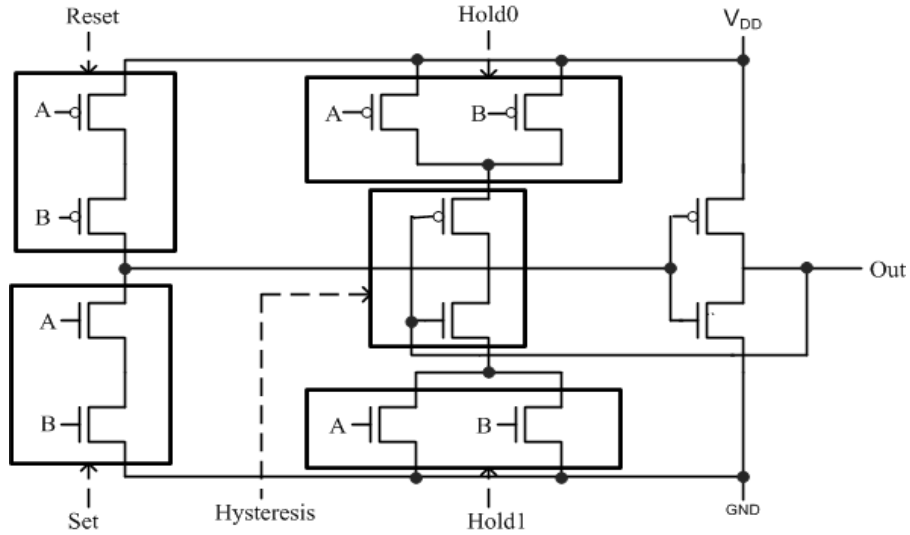


Figure 6. TH22 Transistor-Level Schematic

Inverting gates are n -input OR gates with names of the form $TH1n$. Removing a TH1n's output inverter achieves the function of a $TH1nb$ gate. Resetting gates require two or four extra transistors depending on whether the gate resets to a logic1 or logic0, respectively. Figure 7 and Figure 8 show the schematics for a $TH22d$ (reset to logic1) and $TH22n$ (reset to logic0) gate, respectively.

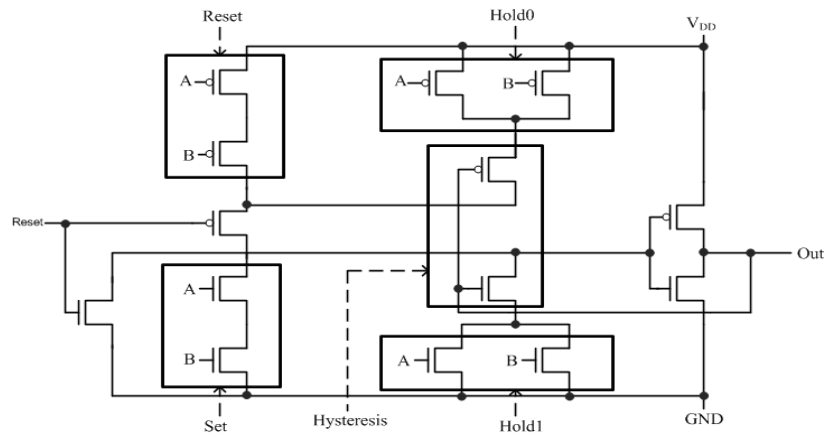


Figure 7. TH22d Transistor-Level Schematic

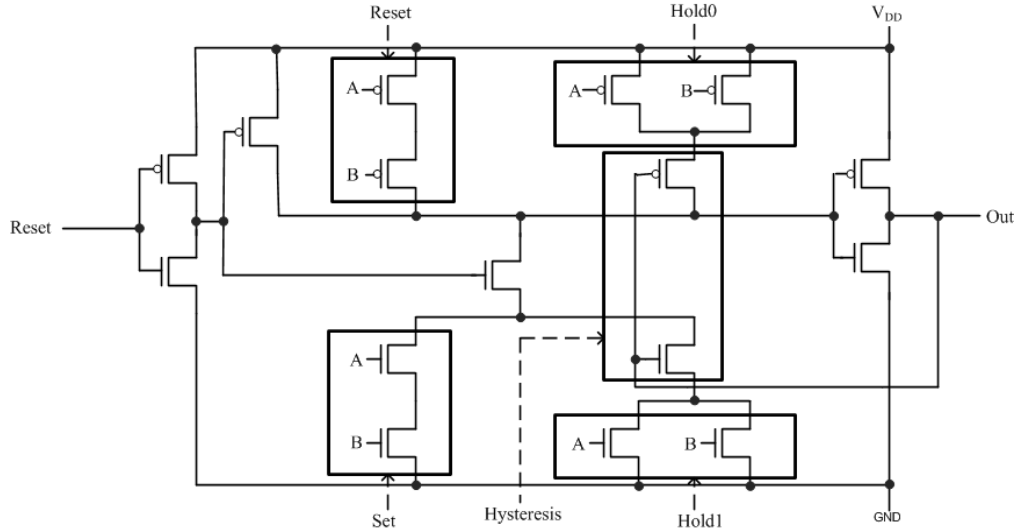


Figure 8. TH22n Transistor Level Schematic

In order for NCL circuits to maintain delay-insensitivity, they must be input complete and observable. An input complete circuit requires that: 1) all outputs of the circuit may not transition from NULL to DATA until all inputs have transitioned from NULL to DATA; and 2) all outputs of the circuit may not transition from DATA to NULL until all inputs have transitioned from DATA to NULL. An observable circuit requires that no orphans are allowed to propagate through a gate. An orphan is a wire that transitions to logic 1 during the processing of an input set but is not used in the determination of the output. Orphans occur wherever there is a wire fork and are neglected through the isochronic fork assumption as long as they do not cross a gate boundary [2].

2.2.2 NCL Registers

Threshold gates used in a combinational circuit can only process sets of DATA and NULL signals, also known as *DATA/NULL wavefronts*. These wavefronts need to be processed cyclically to maintain the circuit's validity. Elements that maintain this cyclical distribution of wavefronts are called NCL registers. These elements store complete DATA/NULL wavefronts

until notified that the wavefront has been successfully processed. At this point, the next corresponding wavefront of the cycle can be stored. These elements also communicate which wavefront type they currently hold so that the wavefront's source knows when to send the next wavefront. An example of a single bit dual-rail register is shown in Figure 9.

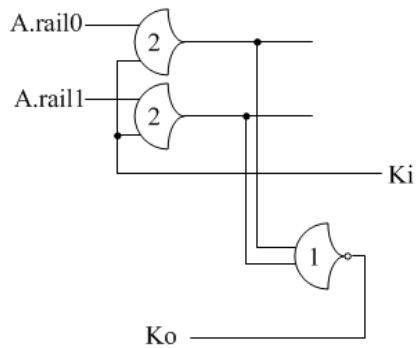


Figure 9. Single Bit Dual-Rail Register

This register receives a control signal, called Ki , which dictates what can be stored or passed next DATA or NULL. When Ki is logic 1, only a DATA can be passed through the threshold gates; when Ki is logic 0, only a NULL can be passed through the threshold gates. The acknowledgment signal, called Ko , communicates what wavefront the register will accept next. Ko becomes logic 0 when a complete DATA is accepted, indicating readiness to accept a NULL; Ko becomes logic 1 when a complete NULL wavefront is accepted, indicating readiness to accept a DATA.

Single-bit dual-rail registers can be combined to form larger n -bit registers. As an example, a 4-bit dual-rail register is shown in Figure 10.

For n -bit registers a completion block – called COMP block – is included that combines the Ko of each bit into one Ko signal that represents the state of the entire register. A COMP block, shown in Figure 11, consists of a $\log_4 m$ tree of TH_{nn} gates, where m is the register's bit width. In a 4-bit register's, the COMP block is one TH_{44} gate.

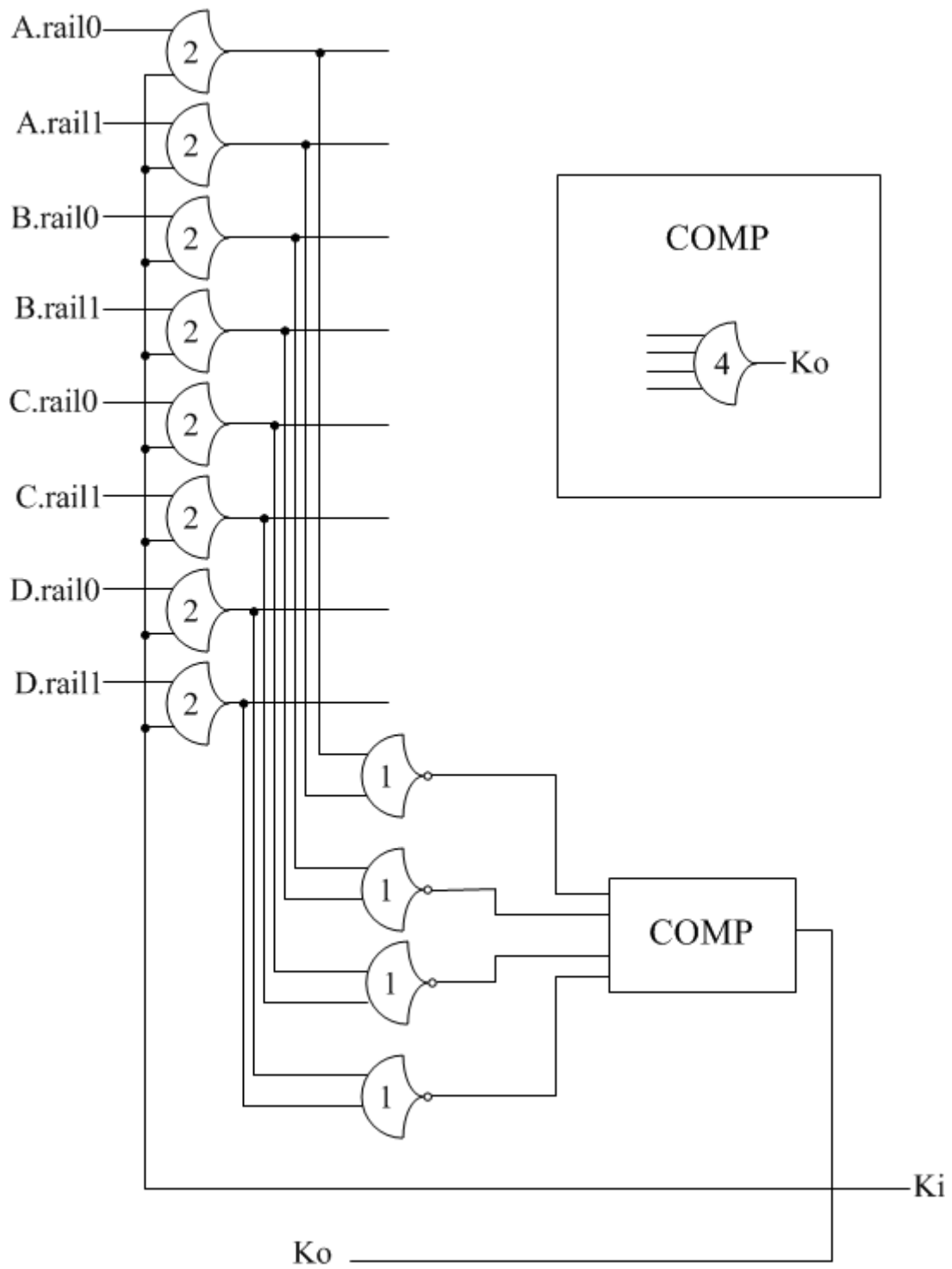


Figure 10. 4-Bit Dual-Rail Register

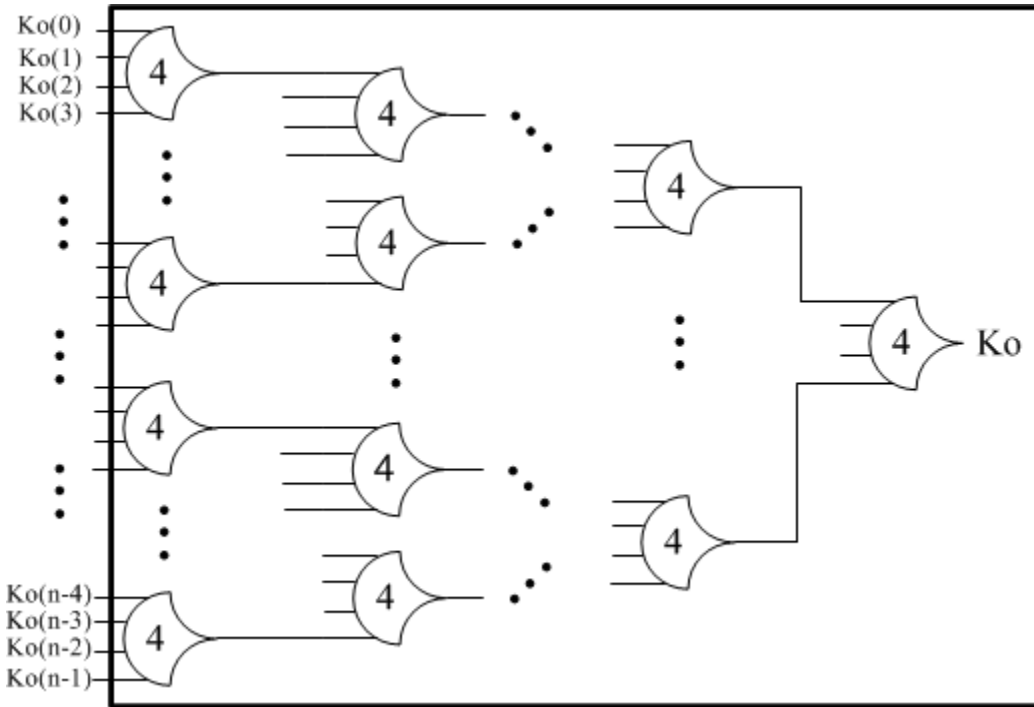


Figure 11. COMP Block

Generally, for a single combinational circuit, two registers placed on each side of the circuit are used to control wavefronts to the circuit. Figure 12 shows such a configuration.

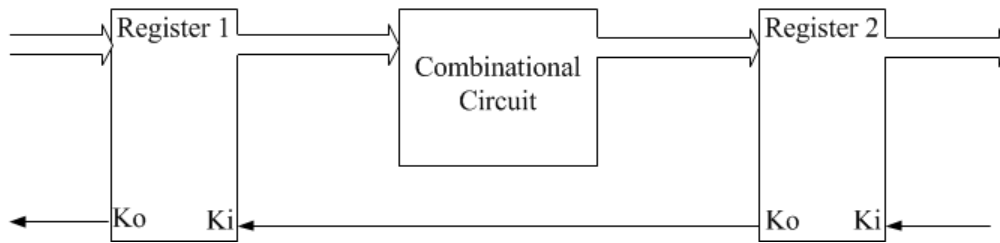


Figure 12. NCL Pipeline Stage

For this and following sections, the use of the terms DATA and NULL is used to mean DATA wavefront and NULL wavefront. Assuming that the two registers and consequently the combinational circuit begin in a NULL state, both registers' **Ko** signals will be logic 1 indicating their readiness to receive a DATA. Register 2's **Ko** signal is used directly as the **Ki** signal of

Register 1. So, Register 1 is communicating with its ***Ko*** signal that it is ready to receive a DATA, and its ***Ki*** signal allows the register to store that DATA. The order that these signals become the required values for storing the DATA is inconsequential. Once a DATA is stored by Register 1, its ***Ko*** will become logic 0 indicating it is ready for a NULL. Register 1's source may receive this signal and send a corresponding NULL, but that NULL will not be stored until the DATA has been processed by the combinational circuit and stored by Register 2. When the DATA reaches Register 2, the ***Ki*** to that register should logic 1 allowing that DATA to be stored. The ***Ko*** of Register 2 will become logic 0 to signify the reception of a DATA and request for a NULL. This will allow a NULL to be passed through the circuit and Register 2, allowing the cycle to repeat in a maintainable order.

Due to the unpredictability of CMOS gates on startup, registers require the use of resetting gates. Replacing the TH22 gates shown in Figure 9 with the proper TH22n or TH22d gate allows for the initialization of a NULL or DATA and is required for the construction of registers and consequently standard storage elements.

The standard storage and data flow element of NCL is called a three-ring register. It is necessary for storing DATA sets through the process of a NULL set, and for acting as a wavefront source. A block diagram of an 8-bit three-ring register is shown in Figure 13.

With ***Reset*** at logic1, Register 1 and Register 2 are initialized to store all NULL values, and Register 3 is initialized to store the value DATA0, which is DATA0 for all its values. Assume that ***Ki*** begins at logic 0 which forces the output of the COMP block to logic 0. With that, the states of the three registers are as follows: Register 1 is being presented with DATA from Register 3, but ***Ki1*** is logic 0, disallowing the passing of that DATA; Register 2 is being presented with NULL, and ***Ki2*** is logic 0 allowing that NULL to pass through; Register 3 is

being presented with NULL, but $Ki3$ is logic 1, disallowing the passing of that NULL. As long as Ki does not change, the states of these registers and their corresponding Ki and Ko will not change. A steady state has been reached; this state is called the NULL state. During this state the output of the 3-ring register is NULL. Once Ko becomes logic 1 indicating that the 3-ring register is in its NULL state, the reset signal may become logic 0.

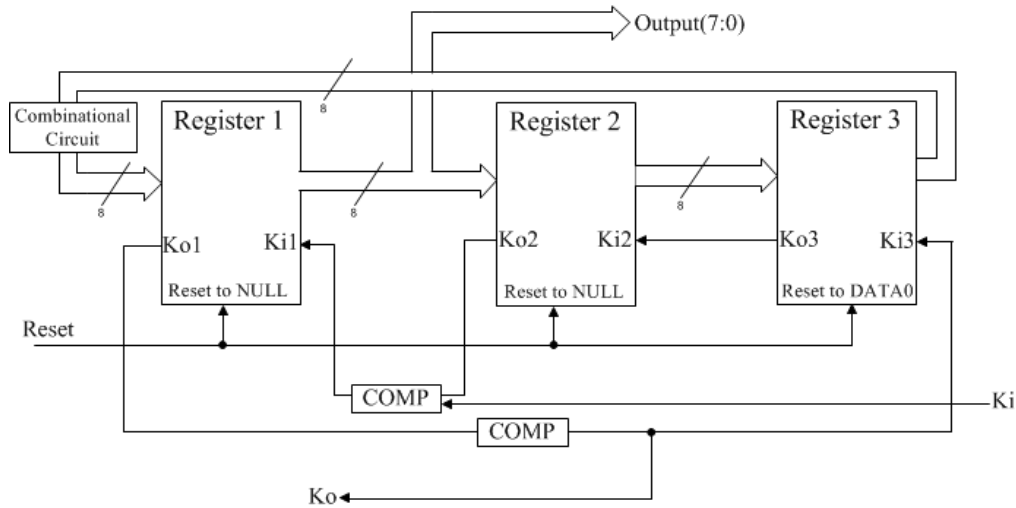


Figure 13. 8-bit 3-Ring Register

When Ki becomes logic 1, a series of events occur resulting in a steady state. First, $Ki1$ becomes logic 1, and the DATA being presented Register 1 will pass through to the second register. The $Ko1$ Register 1 becomes logic 0 after the DATA is passed; this will allow the NULL being presented to Register 3 to be passed through to Register 1. $Ko3$ becomes logic 1 after NULL is passed; this will allow the DATA being presented to Register 2 to be passed through to Register 3. $Ko2$ will go low, and a static state has been reached again, the DATA state. The output of the 3-ring register is now issuing whatever DATA Register 3 has been reset to, in this case, DATA0.

2.2.3 Timing Considerations

With the elements discussed in this section – threshold gates, registers, and 3-ring registers – complex computational circuits and control systems can be created. However, there is one caveat to using hysteretic threshold gates for circuit design. Hysteresis in threshold gates introduces a timing dependency that must be examined before attempting to build complex systems. In a threshold gate, the output of the gate is fed back to its *Hold* blocks for correct maintenance of the output. In the case that this feedback signal is slower than successive wavefronts to the combinational circuit, errors will occur. The required delay ratio for correct circuit operation is shown in Equation 1.

$$\frac{Delay_{feedback}}{Delay_{wavefront}} < 1 \quad \text{Equation 1}$$

For errors to emerge in such a system, the feedback signal's delay would have to be greater than the time needed for the acknowledgment signal to change and the arrival and acceptance of the next wavefront. The gate's feedback path, generally only a transistor or two delay, is significantly faster than the acknowledgment path, which consists of several gate delays. Following standard design practices, this delay relationship is usually close to 0.

2.3 Multi-Threshold NULL Convention Logic (MTNCL)

MTNCL is an asynchronous, delay-insensitive design methodology that combines MTCMOS (Multi-threshold CMOS) power-gating method with regular NCL. Developed in [6-9], MTNCL offers significant improvements in active energy and leakage power savings, throughput enhancement, and area reduction over traditional NCL counterparts, making it a promising candidate for delay-insensitive (DI) asynchronous systems.

The most notable difference between MTNCL and NCL is the usage of the sleep

mechanism inside each MTNCL logic gate to facilitate DATA/NULL wavefronts. When a gate's sleep signal is logic0, that gate is said to be awake. In this state, the gate can process DATA/NULL values. When a gate's sleep signal is logic1, that circuit is said to be asleep. In this state, the gate's output is pulled low facilitating a NULL value. This transfers up to the pipeline level such that when all the gates in a pipeline stage are awake (sleep = 0), a DATA/NULL wavefront can be propagated; when all the gates in a pipeline stage are asleep (sleep = 1), a NULL wavefront is generated. Unfortunately, this change in DATA/NULL wavefront propagation compromises delay-insensitivity through the potential passing of a partial NULL wavefront [24]. Mitigating this problem requires a new completion method called Fixed Early Completion (FEC). FEC ensures that the DATA/NULL wavefront being presented to a pipeline stage's Register matches the requested wavefront of the subsequent pipeline stage before allowing DATA/NULL to be propagated. An example of the resulting pipeline is shown in Figure 14.

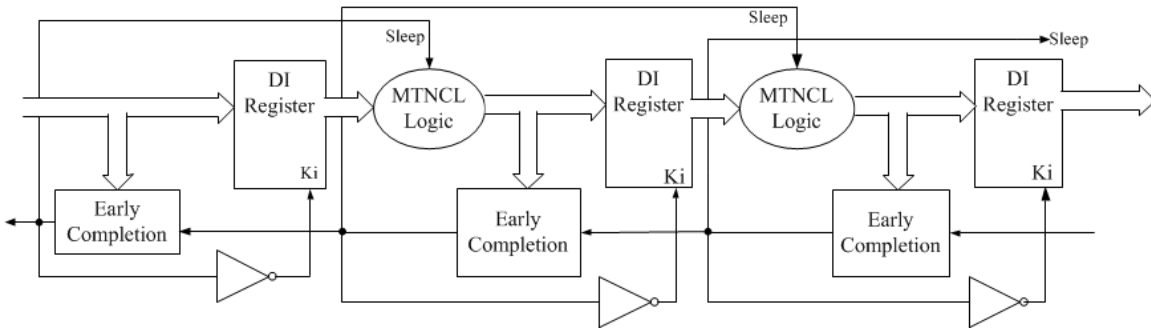


Figure 14: Original Architecture of an MTNCL Pipeline with FEC

Incorporating the MTCMOS structure inside a NCL threshold gate (block structure shown for reference in Figure 15(a)) actually reduces the number of transistors needed to achieve that gate's four functions: *Reset*, *Set*, *Hold1*, and *Hold0*. For one, the *Reset* circuitry is no longer

needed. With the MTNCL sleep mechanism, a gate's output will become logic0 when *Sleep* is logic 0. This facilitates the *Reset* function. Furthermore, all gates in a pipeline stage are forced to sleep by the same *Sleep* signal, so hysteresis is no longer required. As such, the *Hold1* circuitry and corresponding NMOS transistor are removed, and the PMOS transistor is removed to maintain the complementary nature of CMOS logic. These modifications result in the general gate structure shown in Figure 15(b). During active mode, the *Sleep* signal is logic0 and the gate functions normally. During sleep mode, *Sleep* is logic1, and the low- V_t pull-down transistor is turned on to force the output to logic0. At this time, the output inverter's high- V_t PMOS transistor is gated, and all high- V_t NMOS transistors are off which reduces leakage. As an example, Figure 15(c) shows a TH23 gate's MTNCL implementation.

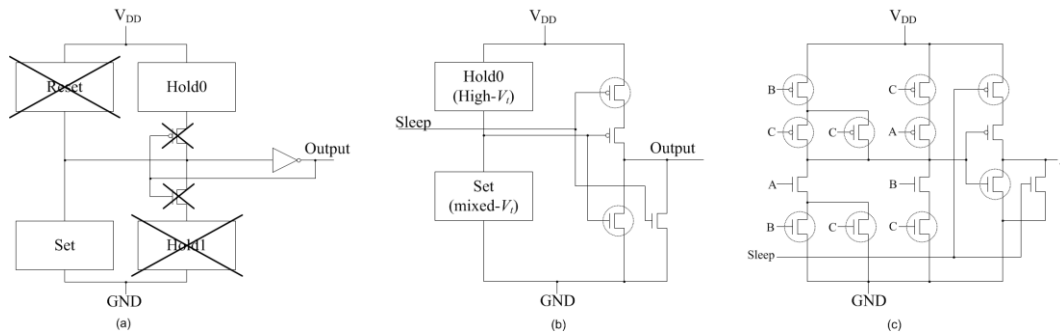


Figure 15: (a) Block Reduction from MTCMOS Power Gating, (b) MTNCL Gate Structure, (c) TH23 MTNCL Implementation (High- V_t Transistors are Circled)

Since the inception of the MTNCL architecture, there have been two enhancements to it: slept completion logic and slept registers [3]. For these enhancements, the *Ko* of a pipeline stage is used not only to sleep the subsequent stage's MTNCL logic, but also the register and completion logic. For slept completion logic, all of the regular NCL gates in an EC block are replaced with their MTNCL version except for the last gate which facilitates maintenance of the *Ko* signal through sleep. For the slept register, the register is redesigned such that after the

register propagates a DATA wavefront, it can only propagate a NULL wavefront when sleep becomes logic1. The resulting pipeline is shown in Figure 16.

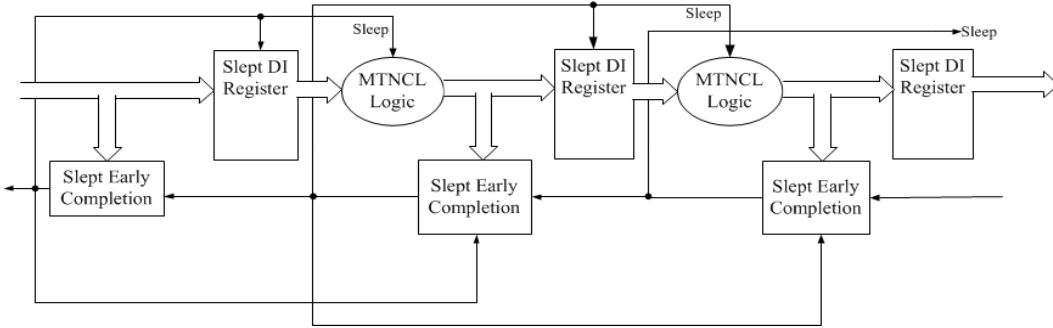


Figure 16: Current Architecture of an MTNCL Pipeline with EC

Compared to MTCMOS synchronous and regular NCL counterparts, MTNCL offers several unique and significant advantages: 1) Leakage reduction in both active and idle modes – since the combinational logic block in each stage goes to sleep after every DATA cycle, leakage power is reduced substantially even when the circuit is actively processing data, in contrast to MTCMOS synchronous circuits where leakage is only reduced when the entire circuit is in sleep mode; 2) Reduced area overhead and active energy – MTNCL architecture ensures input-completeness and observability, such that input-incomplete logic functions can be used to design the circuit, which significantly decreases area overhead and active; and 3) Improved performance – due to the early completion and sleep mechanisms, the throughput of MTNCL pipeline is improved compared to NCL [10].

2.3.1 Timing Considerations

While this most recent MTNCL architecture brings with it several advantages over its architectural predecessors, a new delay ratio is imposed as shown in Equation 2. The addition of the slept DI register mechanic introduces a timing dependency between that register and

subsequent stages of the pipeline. For a given register, if the sleep path, i.e., the path from the subsequent COMP block's output to the given register's sleep input, is faster than the wakeup path, the path from the subsequent COMP block's output to the subsequent register's sleep input, then it is possible to sleep the given register before waking up the subsequent register thereby destroying data. Generally this situation only needs to be examined when the capacitive load of the wakeup path is significantly higher than that of the sleep path, yielding situations where buffering between those nets may become unbalanced.

$$\frac{Delay_{sleep}}{Delay_{wakeup}} > 1 \quad \text{Equation 2}$$

2.4 Dynamic Voltage Scaling (DVS)

DVS is a power management technique where the supply voltage (V_{DD}) of a system is scaled in accordance with a certain power usage strategy. In low power designs, this often means scaling the voltage in accordance with the workload of the system to achieve energy-optimal or ultra-low power performance. Regardless of the strategy, power savings derivations are rooted in the CMOS power equation.

Equation 3 shows the CMOS digital circuit power equation. This power equation has three terms: 1) dynamic power – power consumed during charging/discharging of the capacitive load, C_L ; 2) short-circuit power – power consumed during switching when both PMOS and NMOS networks are on; and 3) leakage power – power consumed regardless of switching activity.

$$P = C_L V_{DD}^2 f_{(0 \rightarrow 1)} + t_{sc} V_{DD} I_{peak} f_{(0 \rightarrow 1)} + V_{DD} I_{leakage} \quad \text{Equation 3}$$

Of the three terms, dynamic power and leakage power have the most impact on overall

power consumption. Reducing V_{DD} linearly provides a quadratic reduction in dynamic power and a linear reduction in leakage power. Typically, at larger process nodes (>90nm), dynamic power is the dominant term in the power equation; however, as process scales down to 65nm and lower, leakage power begins to become the dominant term [1]. This rationale provides incentive to not only scale V_{DD} as responsively and aggressively as possible during active times, but to also scale V_{DD} as low as possible during idle times.

In a synchronous system supply voltage and frequency are directly related; as the supply voltage scales, the frequency must scale with it. Due to the overhead investment and energy-savings per validation time, synchronous systems employing DVS for low-power operation typically have a small set of voltage-frequency (VF) pairs which the system can operate at. If the periodicity of the system's workloads is predictable, switching between VF pairs can be done between workloads to ensure no loss in performance. However, if the periodicity of the system's workload is unpredictable, switching between VF pairs has to be done during workload processing. Since some time is needed to adjust the voltage and frequency, this adjustment must take place within a cycle's slack time in order to maintain system performance. Usually this is not possible for systems focused, to some degree, on performance, so, instead, during a run-time VF switch, it is assumed that several cycles will be lost. VF switching becomes even more problematic when considering changes in V_t due to process variation, temperature variation, and aging which can cause circuit behavior to fall outside the profiled bounds for a VF pair. As variation increases, the circuit is less likely to be able to take full advantage of VF switching without the addition of some variation-aware circuitry.

Due to the delay-insensitivity features of MTNCL, there is no performance loss associated with supply voltage switching during workload processing as in a synchronous

system. If the voltage is scaled up during workload processing, that workload will be processed faster; if the voltage is scaled down during workload processing, that workload will be processed slower. No data is lost as long as the transistors are switching. Furthermore, since MTNCL is correct-by-construction, V_t changes stemming from process variation, temperature variation, and aging will only affect an MTNCL system if V_t shifts are so extreme that they permanently turn on/off a transistor, effectively disabling it from switching. Fluctuations in variation can even be recognized by monitoring the system's handshaking signals.

2.4.1 Literature Review

In recent years, there has been extensive study on DVS as it applies to synchronous systems in areas like workload observation, inter-task and intra-task algorithms, variation-aware techniques, multi-core VF island strategies, course-grain/fine-grain implementations, etc. This section examines current works in several of these areas to better understand the landscape of DVS in synchronous systems.

There are two main approaches to workload-based DVS algorithms for synchronous systems: offline profiling using statistical models and run-time estimation. [19, 20] are examples of works that focus on offline profiling which use statistical modeling of workloads to develop optimal DVS algorithms for hard real-time systems. This type of approach is limited to stationary workloads and requires extensive profiling to utilize a costly DVS algorithm with runtimes of, in the case of [19], $O(n^2)$. For highly variable workloads, a run-time estimation technique is required. [13] proposes such a technique which uses a Kalman filter [15] to estimate the execution time of future workloads. This approach, of course, still incurs deadline misses and lost cycles because of VF switching.

[11] investigates a combination of inter-task and intra-task DVS on systems with

unpredictable workloads. A large portion of their study assumes an ideal model where the processor speed can be tuned continuously and unrestrictedly, and the speed change overhead is ignored. They develop DVS schemes that are optimal for varying DVS strategies under the ideal model which they then apply to a realistic system model. They show that their DVS schemes offer energy savings over other schemes currently held as optimal.

[12] investigates the interplay of DVS with DPM (Dynamic Power Management – sleeping idle off-chip devices at run-time). As voltage/frequency lowers, execution time increases. This causes the idle time for devices to decrease requiring them to be active for longer. When voltage/frequency rises execution time decreases, and the idle time for devices increases. They use this idea to formulate an $m \log m$ (m = number of off-chip devices) algorithm to minimize system-wide energy that combines DVS with DPM.

For chip multi-processors (CMPs), [16] evaluates several different voltage-frequency island (VFI) strategies and [17] presents a variation-aware DVS technique. [21] explores core allocation in an 80-core TeraFLOPS processor for four application types: 1) low compute, low communication; 2) low compute, high communication; 3) high compute, low communication; and 4) high compute, high communication. Clock and power gating strategies are applied to idle cores to show energy savings for each application type. Other explorations involve voltage island granularity coupled with variation-aware design.

Local voltage dithering (LVD), a fine-grain DVS technique is presented in [22] and applied to a 32-bit Kogge-Stone Adder. This technique uses locally embedded power switches to allow for more responsive and customizable DVS. The power savings is achievable only if voltage changes occur in the same time-scale as the altering workload, and, due to the energy overhead, lowering supply voltage requires operation in this lower energy state for a certain

number of clocks to benefit from the switch.

Lastly, a robust summary of modern DVS techniques for logic and memory design, as well as, novel analog power delivery components are presented in [18].

Much of the research on synchronous DVS systems focuses on mitigating the effects of timing fluctuations caused by the voltage scaling itself, process variation, thermal variation, or other timing sensitivities at deep sub-micron process nodes. All these problems can be mitigated through the use of a DI asynchronous methodology like NCL/MTNCL. As examples, an NCL ALU implemented on the 0.5 μ m IBM SiGe 5AM process has shown to work across a continuous voltage range of 3.3V to 0.36V, and an 8031 microcontroller implemented using the same process has shown to work from 3.3V to 2.2V with no features specifically tailored for low-voltage operation.

2.5 Parallelism

Parallelism is an architectural technique where a computational block is instanced multiple times and these instances are set up to run concurrently with each other. In the past, parallelism has generally been thought of as technique to improve performance, but, more recently, parallelism has been employed along with DVS to achieve more energy-efficient systems. Due to the reduction in power as supply voltage is reduced, parallelism combined with DVS can create higher throughput, lower power (energy-efficient) systems at the cost of area [13].

A technique to reduce area overhead, introduced in [4], compares two systems where: 1) one system, PA1, is comprised of only one core with area A; and 2) one system, PA2, is comprised of N cores with each core having area A/N. In this way, the total area of the second system is comparable with the reference system. This is done by manipulating transistor channel

widths. Since the computational capability of each copied core is degraded, the PA2 system requires a supply voltage of γV_{DD} and clock frequency of ηf to process the same input data, where $0 < \{\gamma, \eta\} < 1$. The dynamic power of the PA2 system is $PPA2 = C(\gamma V_{DD})^2 \eta f = \gamma^2 \eta P_{ref}$. Although PA2's dynamic power savings may be smaller than PA1's, the substantial area savings in PA2 can reduce leakage power significantly.

For NCL circuits, some preliminary work has been done in [14] which examines the effects of applying parallelism and DVS on a 4×4 NCL multiplier. The highest performance system required four copies of the multiplier while the most energy-efficient system required only two copies. The results indicate that parallelism coupled with DVS can apply to NCL, and likely MTNCL, systems for improvements to performance and power.

3. Architecture

3.1 High-Level Design

Figure 1 in Chapter 1 shows the system's high-level design. It is comprised of components that can be separated into three categories based on their function: 1) Core components— responsible for performing the type of computation required by the application; 2) Parallelism components – responsible for the distribution and sequencing of data; and 3) Voltage Control Unit (VCU) components – responsible for generating, selecting, and regulating the Cores' V_{DD} .

3.2 Core Design

Several Cores were used in the process of creating a functional system. All Cores are based on the 4×4 pipelined MTNCL multiplier shown in Figure 17. This design was used in the inaugural system and is referred to as Core 1. A system that uses Core N is hereafter referred to as System N . For subsequent Cores/Systems, multiplier chaining, shown in Figure 18, was used. Essentially a number of multipliers are chained together such that the output of one multiplier feeds the input of the subsequent multiplier. Invert blocks are placed between each multiplier to ensure the result does not tend towards 0.

Cores 2, 3, and 4 use an unpipelined version of the 4×4 MTNCL multiplier and contain 1, 2, and 4 multipliers per stage, respectively. Systems 2, 3, and 4 are comprised of 8 stages with 1 Ko per stage. This yields a total of 32 Ko signals for fullness observation. Core 5, used in the final system, uses the pipelined version of the 4×4 MTNCL multiplier and contains 4 multipliers per stage. System 5 is comprised of 1 stage with 32 Ko signals per stage. This yields a total of 128 Ko 's for fullness observation.

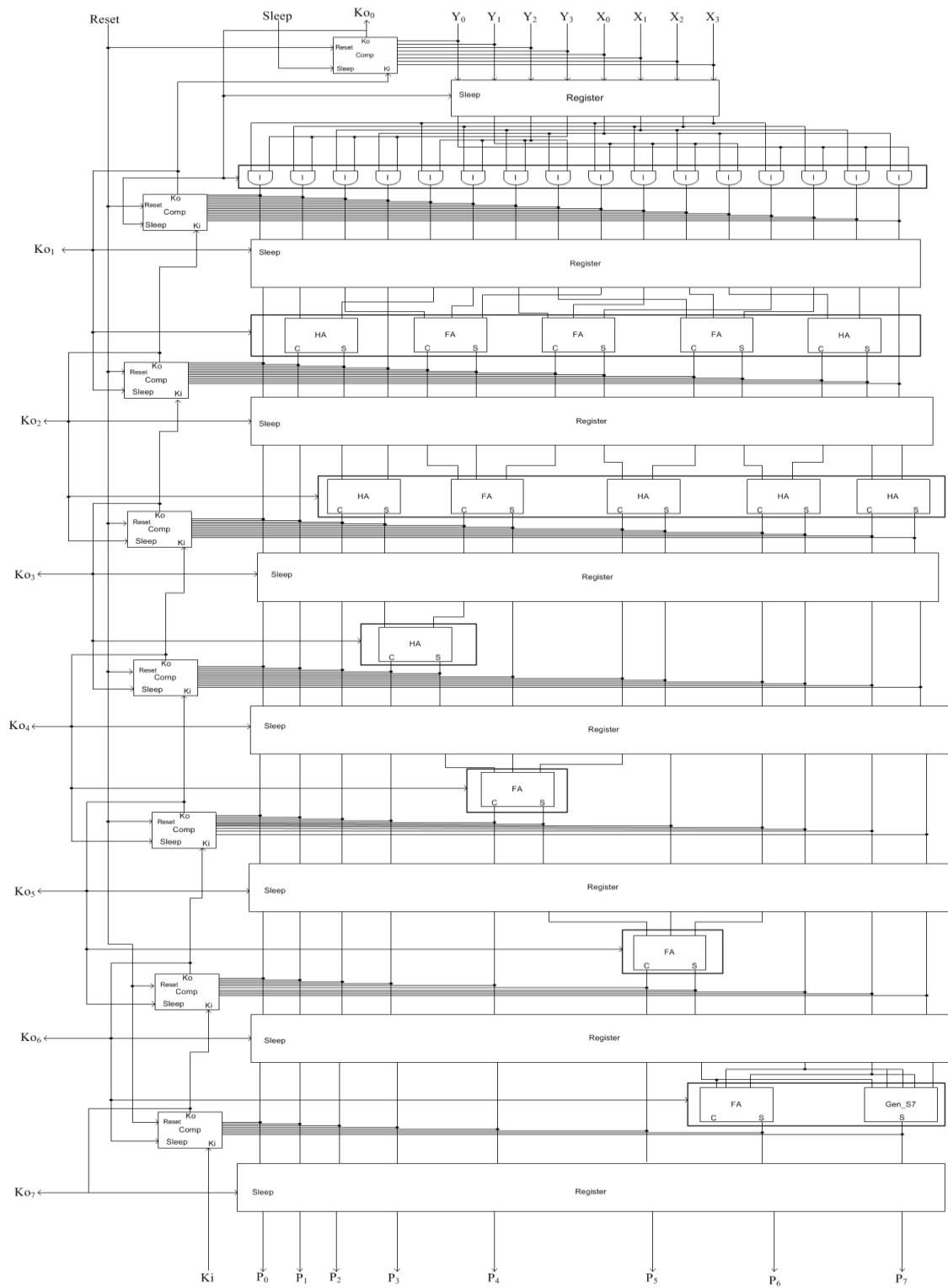


Figure 17: 4x4 Pipelined MTNCL Multiplier

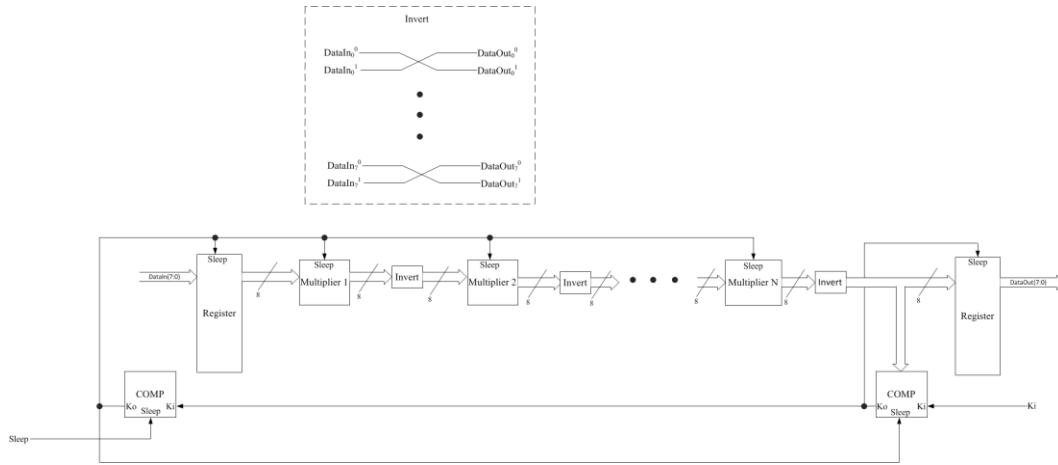


Figure 18: Multiplier Chaining

3.3 Parallelism Design

From Figure 1, the components responsible for parallelism are the Sequencers, the Demultiplexer, and the Multiplexer. These components were designed as generics and their designs are shown in Figure 19, 20, and 21, respectively.

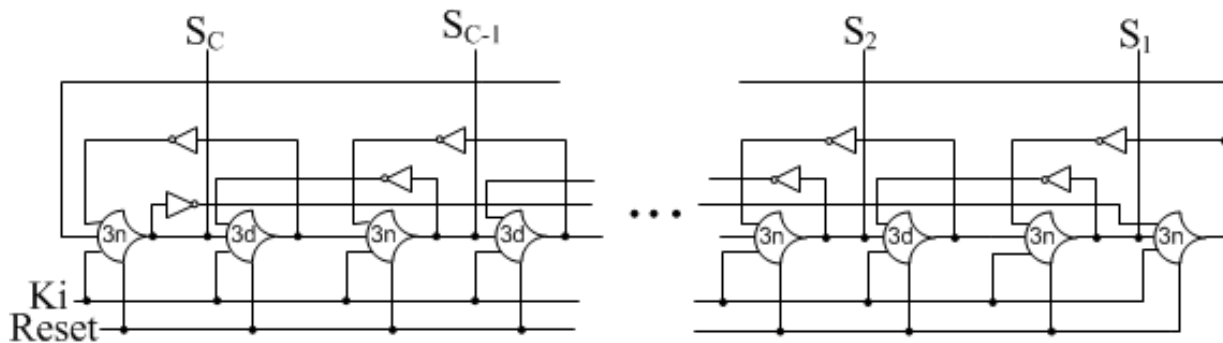


Figure 19: Generic Sequencer

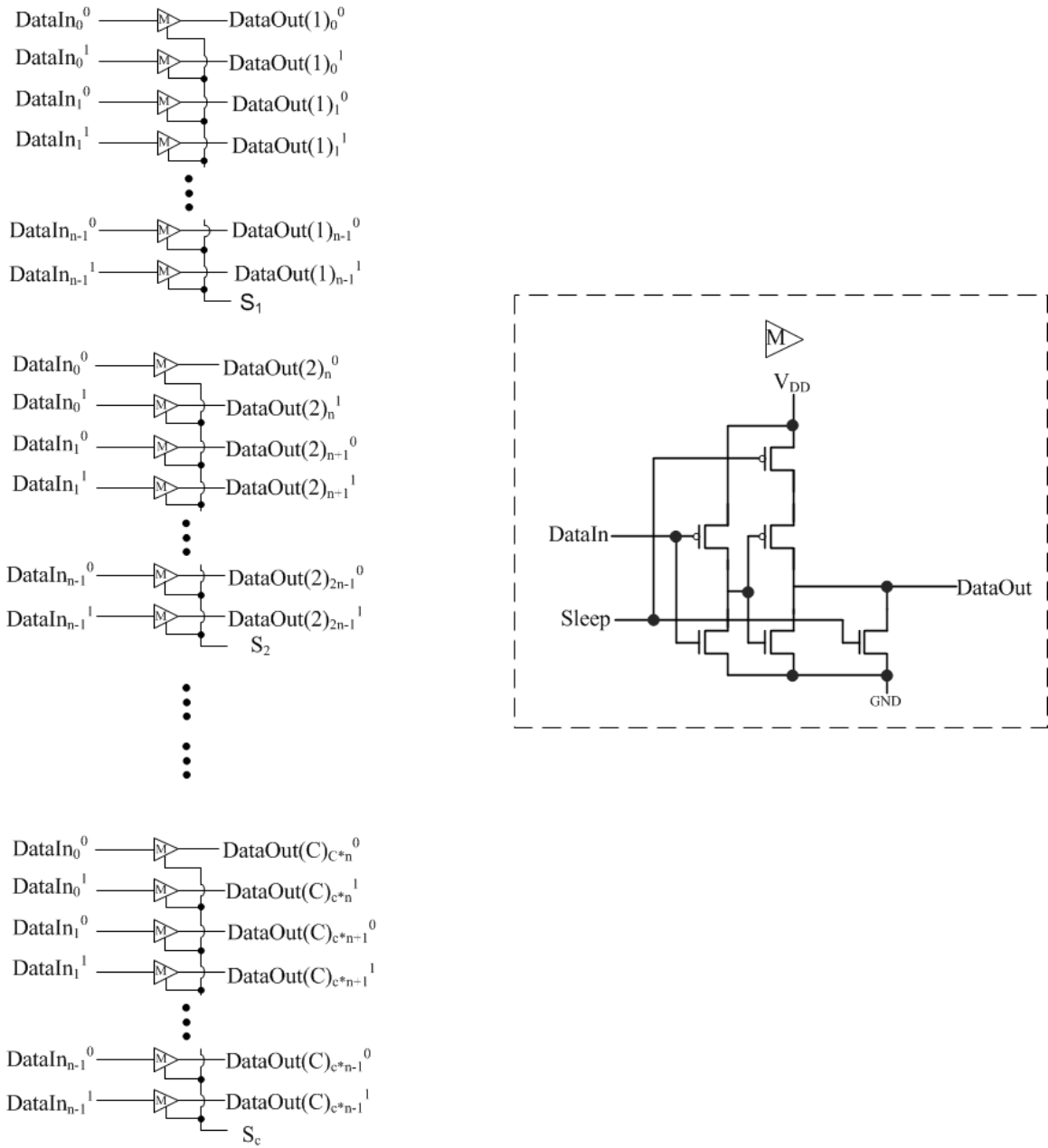


Figure 20: Generic Demultiplexer

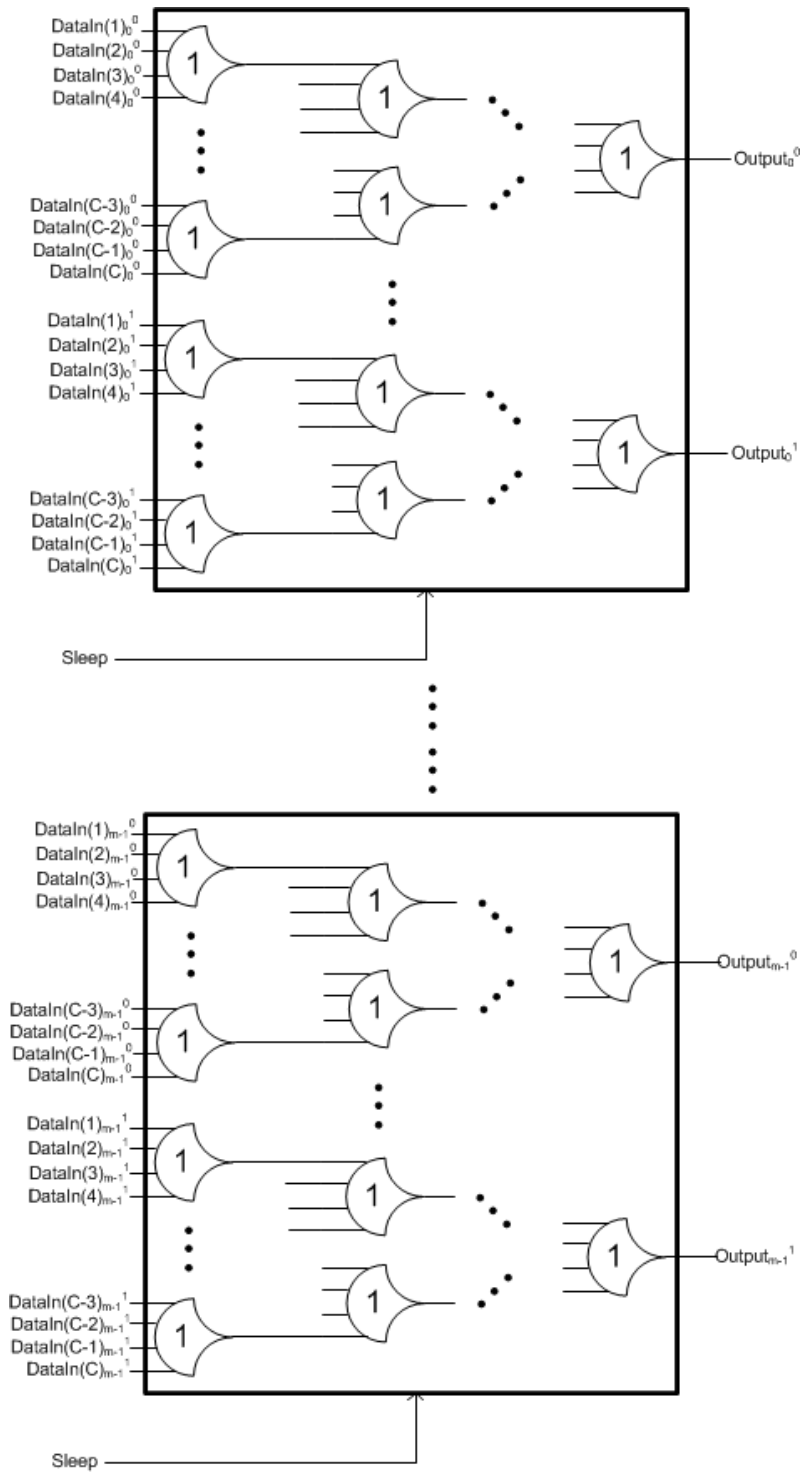


Figure 21: Generic Multiplexer

3.4 Voltage Control Unit (VCU) Design

The VCU's main function is to use the Cores' Ko signals and other inputs indicating the desired Core fullness to select an appropriate V_{DD} for the Cores and supply that V_{DD} with adequate current. The approach of the VCU's high-level design began with breaking down this function into three smaller blocks: V_{ref} Selector, V_{ref} Generator, and Voltage Regulator. A high-level block diagram of the VCU is shown in Figure 22.

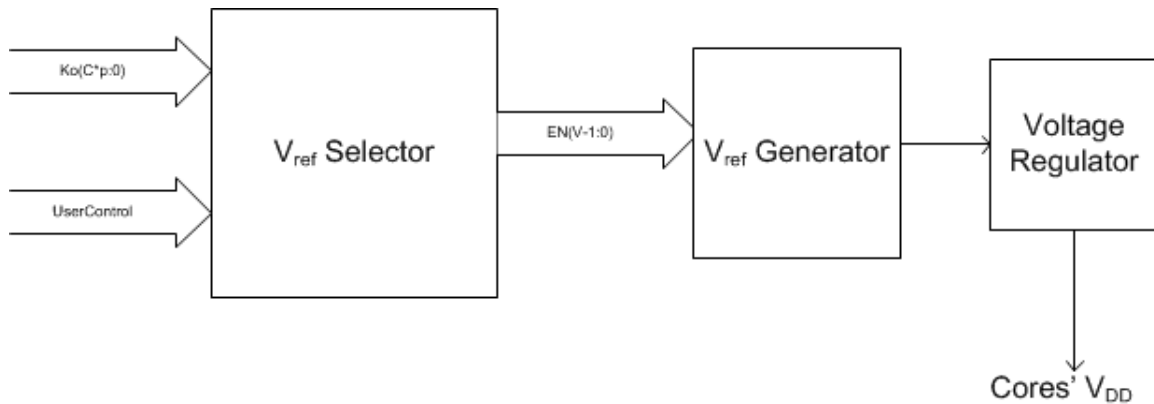


Figure 22: VCU Block Diagram

3.4.1 V_{ref} Generator

The V_{ref} Generator's function is to generate a number of reference voltages and output the one selected by the V_{ref} Selector. Initially this was done using a simple voltage divider with a pass-through gate for each reference voltage, which is shown in Figure 23. While responsive this solution turned out to be flawed for two reasons: 1) it was overkill – such small variations in V_{DD} do not produce observable variations in fullness; 2) it was power inefficient – it consumed power even if the circuit was idle or if the user desired a V_{DD} of 0 V.

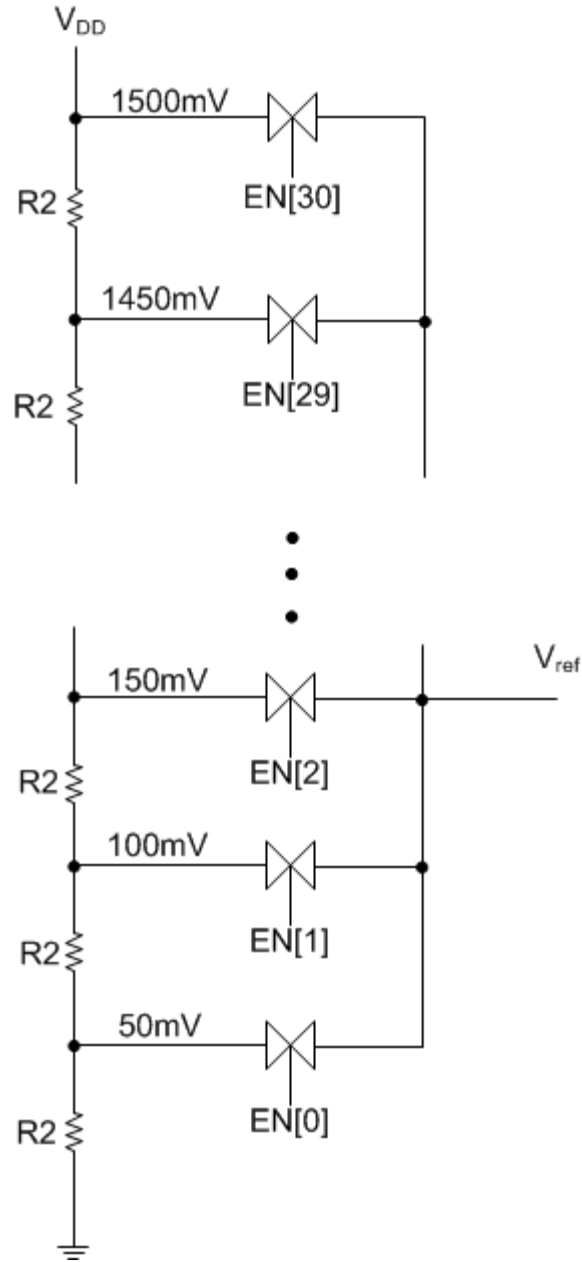


Figure 23: Original V_{ref} Generator

To alleviate the problems of the original V_{ref} Generator, a new one was designed that included voltage reference path gating and a shutdown mechanism. This design is shown in Figure 24.

The new V_{ref} Generator has two modes of operation that depend on if the system is

processing data or idle. When the *Processing* signal is logic1, the system is currently processing data. In this mode, the value of the enabled voltage divider path is allowed to pass through to V_{ref} . When *Processing* is logic0, the system is currently idle – it is processing no data. In this mode, V_{ref} becomes 0 V which gates power to all the Cores' elements connected to the adaptable supply. In this way minimal power is wasted if the system is idle. For this benefit, the design trades responsiveness and ease of device sizing both of which are incurred from the addition of an NFET in each voltage divider path.

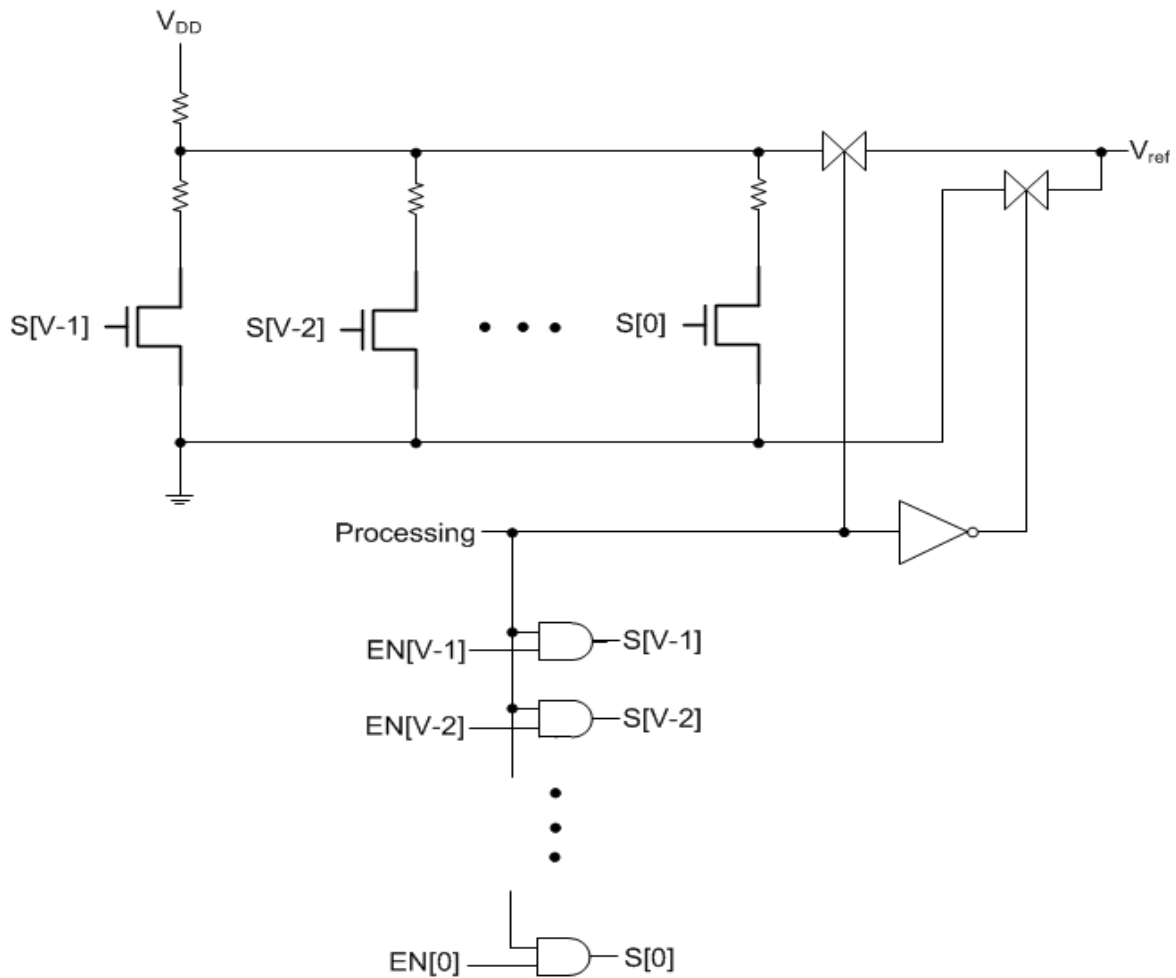


Figure 24: V_{ref} Generator

The function of the *Processing* signal emerged because of a slight modification to the Cores' V_{DD} distribution which was originally intended to increase fullness fluctuation. In the original scheme, there were two voltage islands, one for the Cores' V_{DD} which can be modulated at run-time and one for all other components' V_{DD} which is fixed at 1.2V. The Cores' V_{DD} was originally planned to be distributed to every element of each core, from the COMP block to the last register. Due to low fullness range observed in the original system (System 1), the fixed 1.2V supply was applied to the first register and COMP block of every core. As an example, a color-coded two-core system is shown in Figure 25. All colored blocks have a fixed 1.2V supply while the rest are connected to the adaptable supply. This facilitates the calculation of the *Processing* signal as shown in Figure 26.

When *Reset* is logic1, *Processing* is held at logic1. During this time, all of the *Ko* signals will be reset to logic 1 as well. Each AND block represents a $\log_4 n$ tree of Boolean AND gates (n = number of inputs to the block), so during reset the output of each AND block becomes logic1. When reset drops to logic0, if all *Ko* signals are still logic1 – this indicates the system is idle – *Processing* becomes logic0. This causes V_{ref} to become 0 V which in turn causes the adaptable supply to become 0 V. When the adaptable supply becomes 0 V, all of the completion blocks connected to the adaptable supply also become 0 V. This will cause the top AND block in Figure 26 to become logic 0. Now if data enters the system, the first Core's first *Ko* will become logic0 causing the bottom AND block's output to become logic0. *Processing* will become logic1 and V_{ref} will become the voltage currently being selected by the V_{ref} Selector (typically 1.2V after reset). *Processing* will remain logic1 until all *Ko* signals become logic 1 at which point this process may start over.

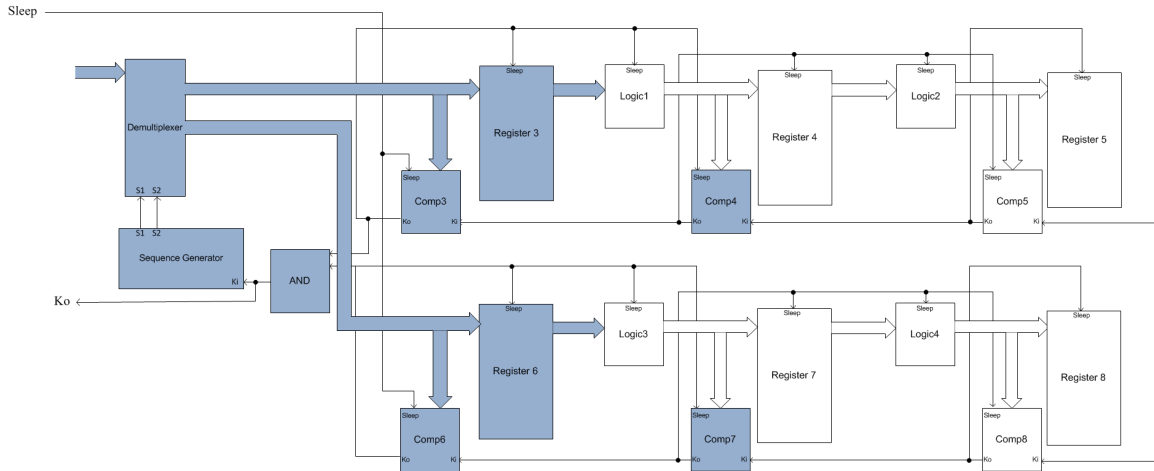


Figure 25: V_{DD} Distribution Example

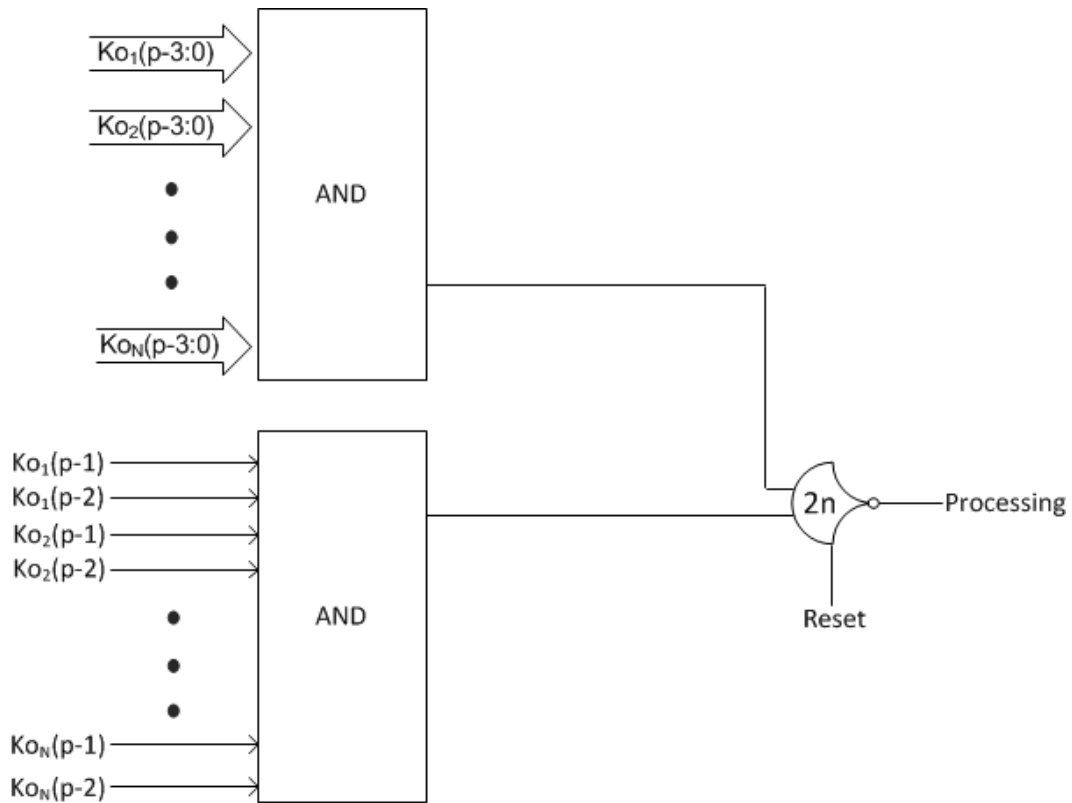


Figure 26: Processing Signal Computation

3.4.2 Voltage Regulator

The Voltage Regulator's main function is to take a voltage reference and supply that voltage with sufficient current to a load, in this case the system's Cores. The design, shown in Figure 27, is essentially a unity gain buffer amplifier. Transistors P1, P3, P4, N1, and N2 form an operational amplifier. Combined with the pass device formed by N4 and R2, the negative feedback loop keeps the output V_{out} the same value of V_{ref} . The V_{out} node has big drive capability (0 – 50 mA). P1 and P2 form a current mirror to provide the operation current for the operational amplifier. R1 is set to let the current flowing through P2 be 20 μ A. N3 works as a bypass capacitor, which is used to improve the stability of the negative loop. Table 4 lists the device sizes for the circuit.

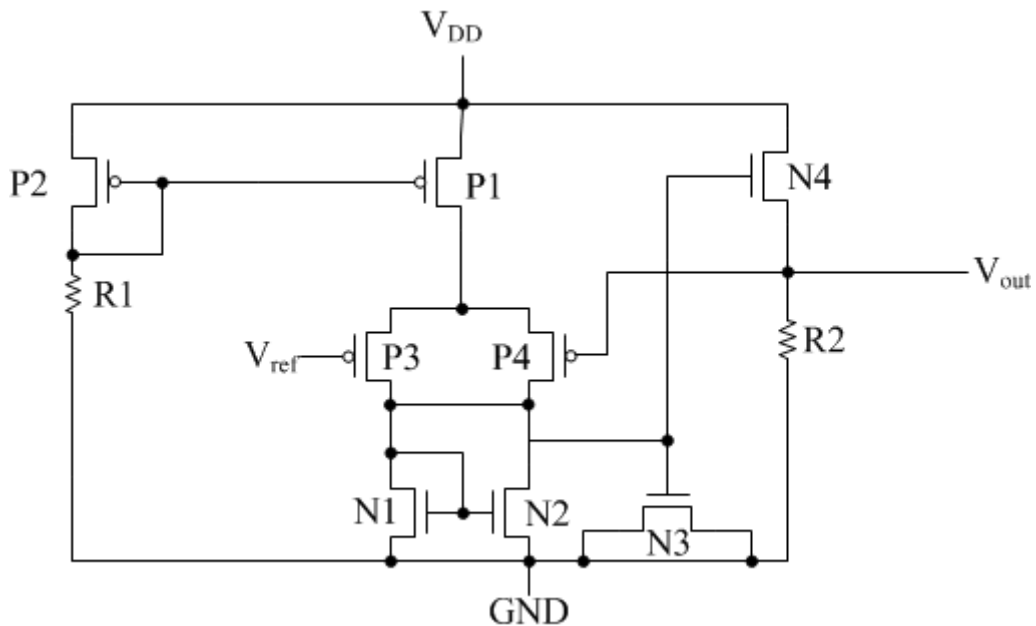


Figure 27: Voltage Regulator Schematic

Originally, this regulator was implemented on the 65nm IBM CMOS 10lpe process. With a system comprised of four pipelined MTNCL4×4 multipliers, the regulator scaled the supply voltage of the cores from 0.6V to 1.2V in approximately 20ns. When the system is running with maximum data rate and maximum V_{DD} , at most 5 data can be input/output to/from the system in this time. As shown in the next section, this response time is more than adequate to detect abrupt changes in workload. In the current system, which uses the IBM 8RF 130nm CMOS process, the regulator is replaced with a VerilogA model that uses the delay information obtained from simulations with the IBMcmos10lpe process.

Component	Parameter	Value
P1	W/L	5 μ m/1 μ m
P2	W/L	5 μ m/1 μ m
P3	W/L	20 μ m/1 μ m
P4	W/L	20 μ m/1 μ m
N1	W/L	10 μ m/0.5 μ m
N2	W/L	10 μ m/0.5 μ m
N3	W/L	5 μ m/0.5 μ m
N4	W/L	10 μ /0.1 μ m
R1	Resistance	80k ohms
R2	Resistance	10k ohms

Table 4: Voltage Regulator Device Sizes

3.4.3 V_{ref} Selector

The V_{ref} Selector's function is to use the Cores' Ko signals and the *UserControl* input, which indicates desired fullness, to select an appropriate V_{DD} for the Cores. This function can be broken down into two parts: a trigger mechanism (clock or handshaking signal) and a combinational circuit. Due to the Cores' free-flowing feed-forward pipeline structure, the Cores' Ko signals switch in an autonomous fashion. Thus they cannot be used in a DI asynchronous combinational circuit unless they are made to switch in a sequential fashion. The area overhead for the extra handshaking severely impacts throughput to the point of implementing parallelism pointless. As such, it was clear from the onset that the combinational circuit would be Boolean in nature; however, as the trigger was not necessarily based off the Cores' Ko signals, several design styles were considered for its implementation

3.4.3.1 Design Style Analysis

Initially there were several goals set for the Selector's implementation: 1) Minimize power; 2) Minimize throughput impedance; 3) Maximize computation resolution; and 4) Minimize area. Goals 1 and 4 depend mainly on the Selector's algorithm. Goal 2 depends on how well the Selector can work in parallel with the system. Goal 3 depends on the event used to trigger the Selector's computation. For maximum resolution the computation is triggered whenever the fullness changes which, from Figure 1, is whenever Ko/Ki switches. With these ideals in mind, three design styles were considered for the Selector's trigger: asynchronous, synchronous, and bounded-delay.

When considering an asynchronous design style, the initial problem was to determine how the Selector's computation would be triggered. The first, most obvious way was to use the system's Ko/Ki for the Selector's trigger (Ki signal). Unfortunately, due to the independent

switching of Ko and Ki , only one of them can be used as the Selector's trigger without breaking delay-insensitivity, and this has serious implications on the Selector's resolution.

Using Ko for the Selector's trigger, the Cores' V_{DD} can be modulated as long as data is entering the system, but, if this stops, V_{DD} may remain at an undesired value as the system empties. A reciprocal problem occurs when Ki is used as the system's trigger. From an empty state, the system could become quite full before modulating V_{DD} in a desired manner. This problem can be averaged somewhat if another sequencing step is placed in the middle of the Cores' pipeline and that sequencing step's Ko is used. However, the increase in area and decrease in throughput does not gain much; the Selector's resolution is still only half of its ideal value. Furthermore, in order to avoid impeding system throughput, the Selector's computation would have to finish in parallel with Ko/Ki 's stage. It may not be possible to pipeline the Selector so that the delays are matched. For these reasons, this scheme seemed unappealing. In response, the Cores' internal Ko signals were examined as a means to trigger the Selector.

The use of the Cores' internal Ko signals is even less effective because of their aforementioned switching behavior. This independent switching activity restricts the Cores' Ko signal usage to one Core's Ko . This poses two problems: 1) Computation resolution would be even less than the first scheme, only triggering for one of the N core's stages; and 2) the Selector's logic may not be pipelined in a way that can match the selected stage's computation time. These asynchronous approaches yielded less than optimal resolution time and uncertainty with regards to throughput impedance. This leads to the consideration of a synchronous design style.

With a synchronous design style, the Selector's trigger is a clock signal which, in order to ensure ideal resolution, can be set to Ko/Ki 's average switching time when the system was

operating at maximum V_{DD} and maximum data rate. Unfortunately, this would be an egregious waste of power should the data rate decrease or the user desire slower system operation. To counter this, the clock's frequency could be modulated along with the voltage; however, this presents overhead in the form of clock generation, clock selection, and timing analysis. This approach seems unappealing given that Ko/Ki , by default, indicates every change in fullness, and should be able to trigger the computation more directly. This leads to the consideration of a bounded-delay design style.

Using a bounded-delay design style, an implementation closely aligned with the proposed goals was derived. This implementation is shown in Figure 28. In this implementation, *Modulate* essentially acts as a clock signal for the combinational circuit responsible for computing V_{ref} . Note there are two delay elements, QD1 and QD2. QD1's path disables setting of the NOR latch when *Modulate* is logic1. QD2's path is responsible for resetting the NOR latch. These paths' functions dictate the delays of QD1 and QD2. On a rising edge, QD1 needs to be just long enough for *Modulate* to latch the system's Ko signals; on a falling edge, QD1 needs to be long enough to prevent a concurrent set and reset of the latch. On a rising edge, QD2 needs to be as long as the worst-case delay in the Selector's pipeline; on a falling edge, QD2 can be as short as possible. Ideally the delay looks like the waveforms shown in Figure 29. This leads to the QD1 path being quick to rise and slow to fall, while the QD2 path is slow to rise and quick to fall. This can be implemented by imbalanced PFET/NFET sizing in the QD1/QD2 delay chains or through a VerilogA model as shown in Figure 30. While the trigger mechanism accomplishes all four of the initially proposed goals, it introduces some small timing dependencies due to the required delays. This imposes some level of timing analysis for a complete system; however, the analysis should be brief as each delay is only based on one event.

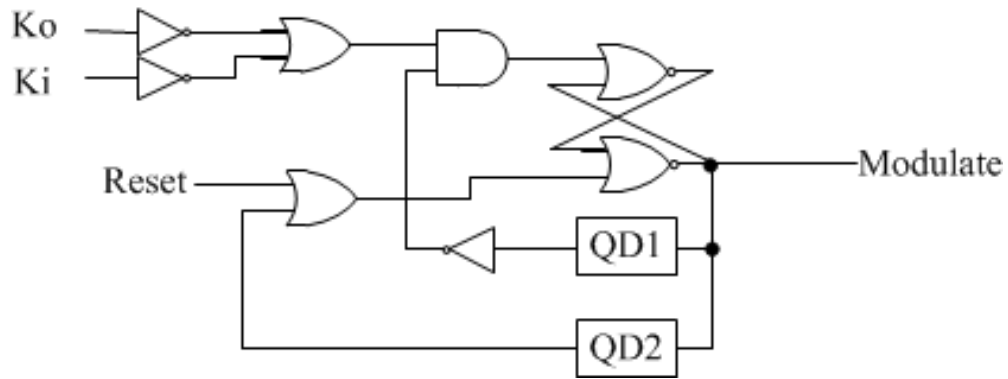


Figure 28: V_{ref} Selector Enable

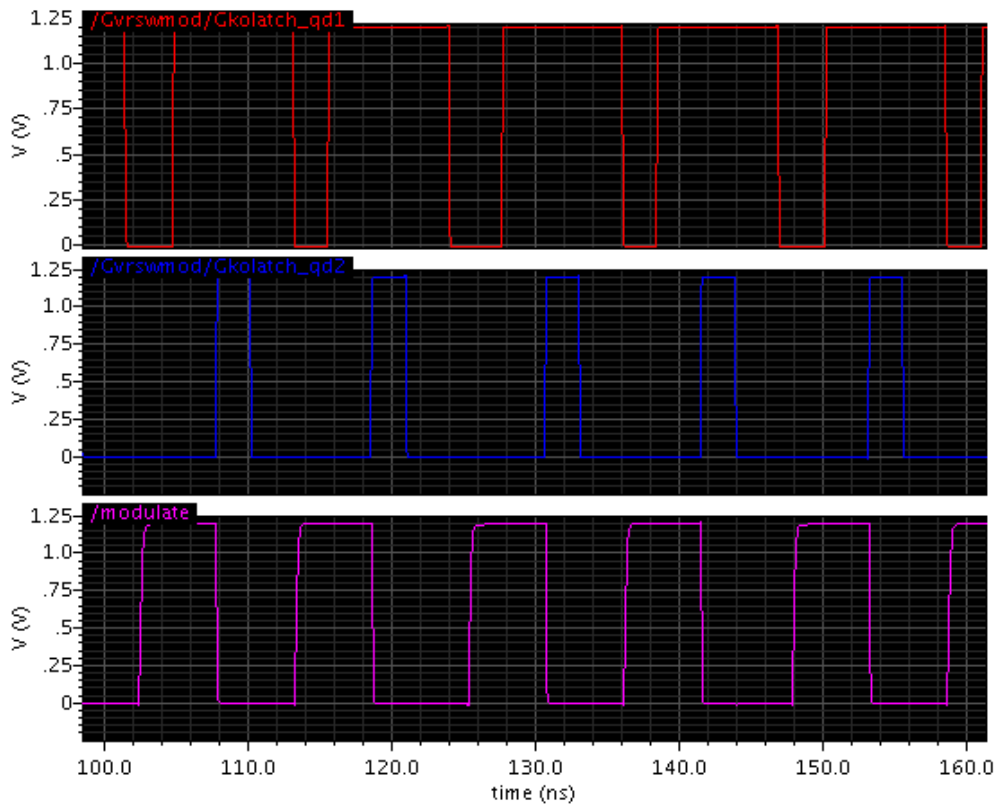


Figure 29: Ideal Modulate, QD1, QD2 Relationship

```

analog begin
@ (initial_step)
begin
    qd1delay = 2n;
    qd2delay = 5n;
end
@(cross(V(q)-0.8, +1, 0.2, 0.05))
begin
    qd1delay = 2n;
    qd2delay = 5n;
end
@(cross(V(q)-0.4, -1, 0.2, 0.05))
begin
    qd1delay = 5n;
    qd2delay = 2n;
end
V(qd1) <+ absdelay(V(q), qd1delay, maxd);
V(qd2) <+ absdelay(V(q), qd2delay, maxd);

```

Figure 30: VerilogA Delay Block

3.4.3.2 High-Level Design

With a triggering mechanism in place, the next problem was to decide if the combinational circuit should be referential – include the current V_{ref} selection in the computation – or non-referential – do not include the current V_{ref} selection in the computation.

The non-referential approach would require a well-characterized system which implied a very large register bank that stored fullness levels for a large variety of user inputs and data rates. Enough fullness information would have to be stored to allow logic to accurately guess what the next V_{ref} should be based only on the current fullness, user input, and stored fullness information. Even though the logic itself would have the benefit of being purely combinational, this approach seemed undesirable due to the potential area overhead, amount of system profiling required, and inflexible nature of the algorithm.

The referential approach uses the currently selected V_{ref} along with the difference between current fullness and desired fullness (specified by the user) to determine what the next

V_{ref} should be. This feedback path implies the addition of a register for V_{ref} and a potential third delay path for the triggering mechanism. Furthermore, some number of fullness levels corresponding to supply voltages ranging from 1.2V to the lowest operating voltage would need to be stored as reference points. The combinational circuit can use this to determine how much higher/lower the next V_{ref} should be from the current one and the overhead should be minimal compared to non-referential approach. Lastly, the logic itself should be small as the circuit is only comparing the system's current fullness with some referenced fullness levels and making a selection. For these reasons, this approach seemed most aligned with the system's goals. The selection of a referential approach leads to the high-level design in Figure 31.

The *Modulate* signal is used to clock the Cores' *Ko* signals into Register 1. In MTNCL pipelines, a pair of sequential stages is woken up before the pair's first stage slept. This allows the clock signal to switch at any time during data processing without fear of missing data's presence. After the *Ko* signals are clocked into Register 1, they pass through the Data Detector. The Data Detector divides the *Ko* signals into four-bit sequences and determines how many data are present in each sequence. The output is a two-bit number for every four-bit sequence. The two-bit numbers are input to a generic Ripple-Carry Adder tree which outputs a $\log_2(C*p-1)$ -bit number representing the number of data currently in the Cores, i.e., the current fullness. The combinational circuit uses this fullness, along with the desired fullness, and the V_{ref} generator's enable vector (*EN*) to produce a new *EN*. This new *EN* is clocked into Register 2 by *Modulate*.

The combinational circuit's high-level design is shown in Figure 32. The current fullness is compared with the desired fullness. This results in a difference magnitude (positive value) and the magnitude's direction, i.e., whether $A > B$ or $A < B$. The difference magnitude is sent through the SetRange block which produces a one-hot signal where each bit represents a group of non-

overlapping magnitude ranges, .e.g., 0 to 6, 7 to 14, 15 to 30, etc. Thus, the difference magnitude's range and direction are used to modulate V_{ref} .

With this model in place, the only unknown left was the profiled fullness levels. To determine a good set of fullness levels, the system fullness was observed across a wide range of voltages.

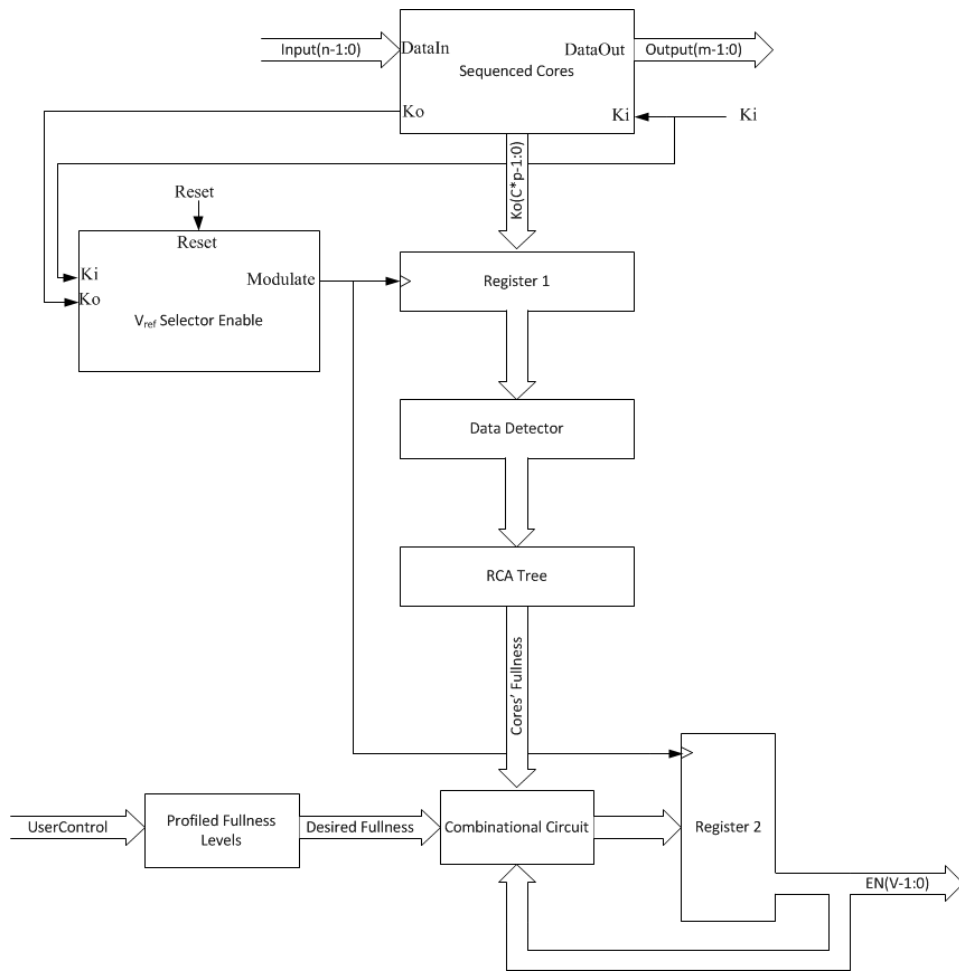


Figure 31: V_{ref} Selector High-Level Design

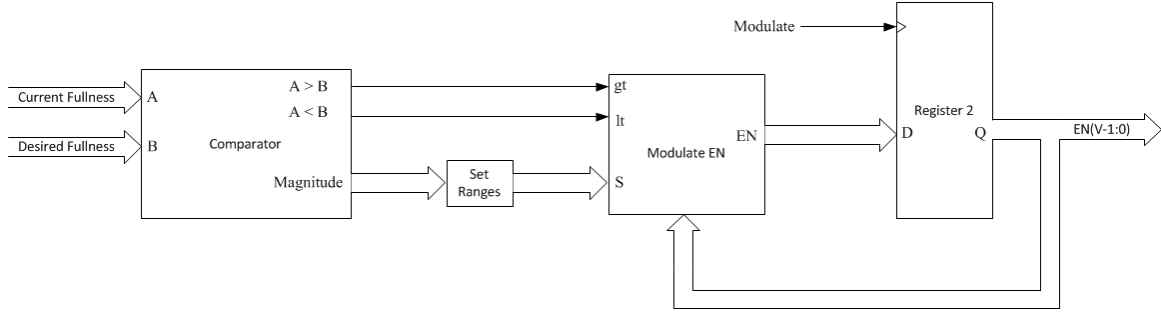


Figure 32: V_{ref} Selector Combinational Circuit

3.4.3.3 System Profiling

Initially the Core's fullness was observed with Core V_{DD} scaled from 1.2V to 0.6V and with maximum data rate. To ensure maximum data rate, a 3-ring register implementing an 8-bit internal lineal feedback shift register (LFSR) provided the system with wavefronts. The fullness observations from these simulations are listed in Table 5.

As can be seen, the fullness only varied by at most 4 across the entire voltage range. These results countered the initial conceptions that DVS should have a noticeable impact on Core fullness range. It was thought that perhaps the total delay of each Core was too short allowing data to leave the system too quickly. This prompted a reconfiguration of the Cores. The multiplier's pipeline elements were removed. A new system, System 2, was configured using this multiplier to have: 4 Cores, 8 stages per Core, and 1 Multiplier per stage. This system's fullness was observed with Core V_{DD} scaled from 1.2V to 0.7V and with maximum data rate. The results are shown in Table 6.

The fullness range was even smaller this time. Two more systems were created to flesh out the delay relationship. System 3 consisted of: 4 Cores, 8 stages per Core, and 2 Multipliers per stage. System 4 consisted of: 4 Cores, 8 stages per Core, and 4 Multipliers per stage. The

observed fullness from these systems' simulations are shown in Table 7 and 8, respectively.

Core V_{DD} (Volts)	Fullness
1.2	8
1.1	8
1.0	10
0.8	12
0.7	10
0.6	11

Table 5: System 1 Fullness

Core V_{DD} (Volts)	Fullness
1.2	2
1.1	2
1.0	2
0.9	2
0.8	2
0.7	2

Table 6: System 2 Fullness

These sets of simulation results proved conclusively that some other delay relationship was the fullness range's primary factor. This could be the case if data was somehow entering each Core at close to the same rate that it was being processed. To ensure that data was entering the Core as fast as possible, V_{DD} distribution was changed. This change is explained in Section 3.4.1 and exhibited in Figure 25. This change had little effect on Systems 2, 3, and 4's fullness range; however, System 1 showed marginal improvement. To increase variance further, additional pipelined multipliers were chained inside a given Core to increase each Core's total delay. The final system, System 5, consisted of: 4 Cores, 4 Multipliers per Core, 8 stages per Multiplier, and 1 Ko per stage. System 5's observed fullness is shown in Table 9.

Core V_{DD} (Volts)	Fullness
1.2	3
1.1	4
1.0	3
0.8	3
0.7	5
0.6	6

Table 7: System 3 Fullness

Core V_{DD} (Volts)	Fullness
1.2	4
1.1	4
1.0	4
0.9	4
0.8	4
0.7	5

Table 8: System 4 Fullness

Core V_{DD} (Volts)	Fullness
1.2	7
1.1	7
1.0	8
0.9	10
0.8	14
0.75	15
0.725	16
0.7	20

0.65	36
0.6	51
0.55	56
0.54	56
0.535	56
0.5325	54
0.53	63
0.525	64
0.51	64

Table 9: System 5 Fullness

Fullness was observed from 1.2V, nominal V_{DD} , to 0.51V, which was the lowest operating V_{DD} . System 5's results showed fullness following V_{DD} as originally predicted. Fullness peaked at 64, which means that half of the 128 stages are processing data. In MTNCL, this corresponds to maximum fullness. These results suggested that a wide variety of V_{ref} values could produce observable differences in voltage. To be sure, fullness variance, the amount fullness varies in a steady supply state, needed to be observed. For this study, the Core's fullness was observed at several voltages from 1.2V and 0.53V with maximum data rate. As examples, waveforms from the 1.2V and 0.53V simulations are shown in Figure 33 and Figure 34, respectively. In each figure, the *dc* signal indicates system fullness.

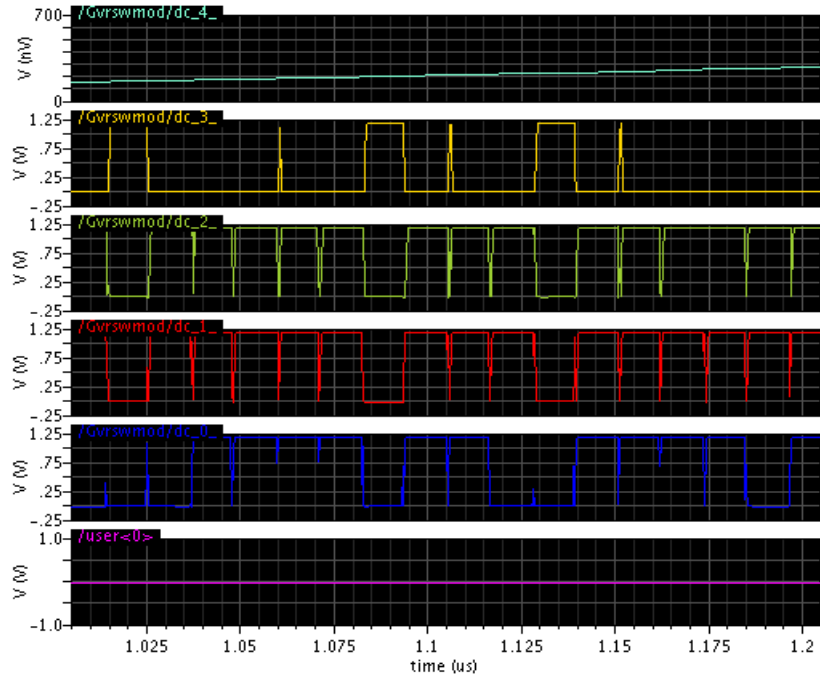


Figure 33: Fullness Variance at 1.2V

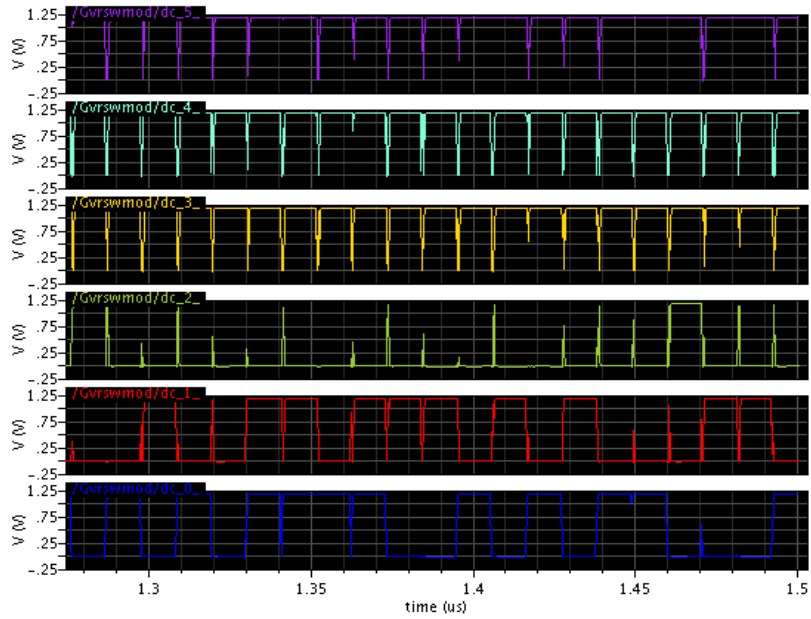


Figure 34: Fullness Variance at 0.53V

At 1.2V the fullness variance was only 2; however, as voltage scaled down fullness variance increased peaking at 4 with a 0.53V supply. Hence reference fullness values must differ by at least 4 to be observable; to be conservative this difference is increased to 6. With that in mind, a set of fullness values was selected such that the differences between subsequent values are close-to-equal and at least greater than 6. The chosen values correspond to the voltages: 1.2V, 0.7V, 0.65V, 0.6V, and 0.53V. With the profiled fullness levels set, the SetRange and ModulateEN blocks could be fleshed out.

SetRange takes the difference magnitude from the comparator and produces a one-hot signal, S . Each bit of S represents a group of non-overlapping magnitudes. One group encompasses the fullness variance, in this case, [0,6]. This group, represented by $S(0)$, indicates that the fullness of the system has not changed enough to warrant a change in EN . The other groups are likewise based on events that cause changes in magnitude. These events are: 1) *UserControl* switches; and 2) data rate changes. To determine how these events can affect magnitude, the minimum and maximum effect of each event is observed.

Event 1 can cause a change at least equal to the minimum difference between subsequent fullness levels which in this case is 12. Conversely the maximum change Event 1 can cause is equal to the difference between the highest and lowest fullness levels which in this case is 63. Data rate changes have the most effect in the following two cases: 1) Abrupt switch to minimum data rate when the system is full; and 2) Abrupt switch to maximum data rate when the system is empty. As Figures 35 and 36 show, in either case, the maximum magnitude change during a Modulate cycle is about 4. From these observations, two more groupings were created: 1) [7, 13], represented by $S(1)$, indicating a small change in user control or some significant change in data rate; and 2) [14, 63], represented by $S(2)$, indicating a large magnitude change due to user

control switch.

Essentially, when $S(0)$ is logic1, magnitude is within its variance range and V_{ref} should not change; when $S(1)$ is logic1, V_{ref} should change by one level as we have observed a magnitude outside of variance range; when $S(2)$ is logic1, V_{ref} should change by several levels as the user's fullness desire has sharply changed. For simplicity, the V_{ref} Generator was limited to the 5 voltages corresponding to the profiled fullness levels. This leads to equations shown in Figure 37.

From Figure 37, $V(4:0)$ represents the five possible V_{ref} values: 1.2V, 0.7V, 0.65V, 0.6V, and 0.53V, respectively. If a large change in magnitude is detected, the highest/lowest voltage is selected depending on if the current fullness is greater/less than the desired fullness. This allows for the quickest response to the user's desires. Changes in data rate, even the most abrupt ones, are detectable and handled by selecting a higher/lower voltage depending on if the data rate decreases/increases. In this way, voltage can be modulated in a simple, responsive manner in real-time with little area/power/design-time overhead. In fact, the only parts of the Selector that cannot be designed as generic components are the V_{ref} Selector Enable, SetRange, and ModulateEN blocks. These blocks require some system profiling before their derivation is possible, but all other blocks can be made in a generic, regular fashion.

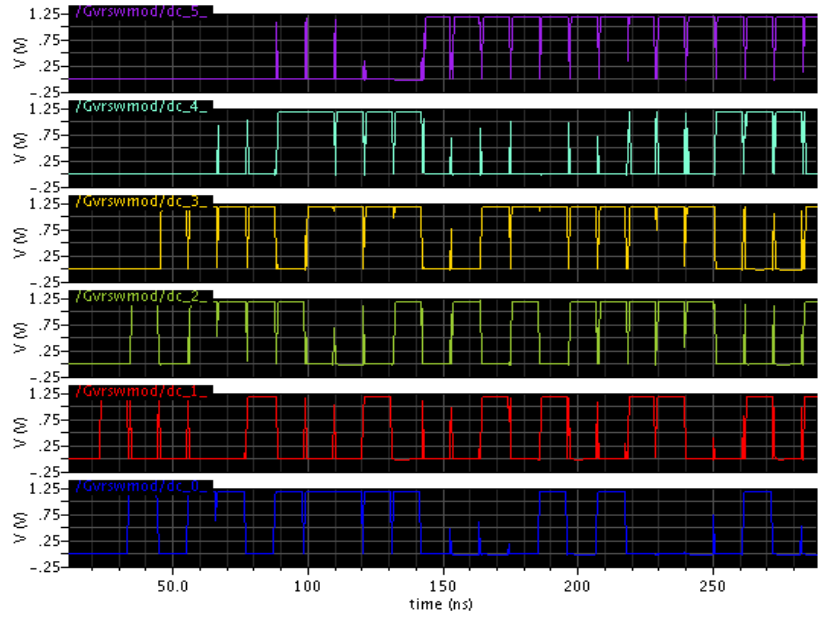


Figure 35: Max Data Rate's Fullness Effect on Empty System

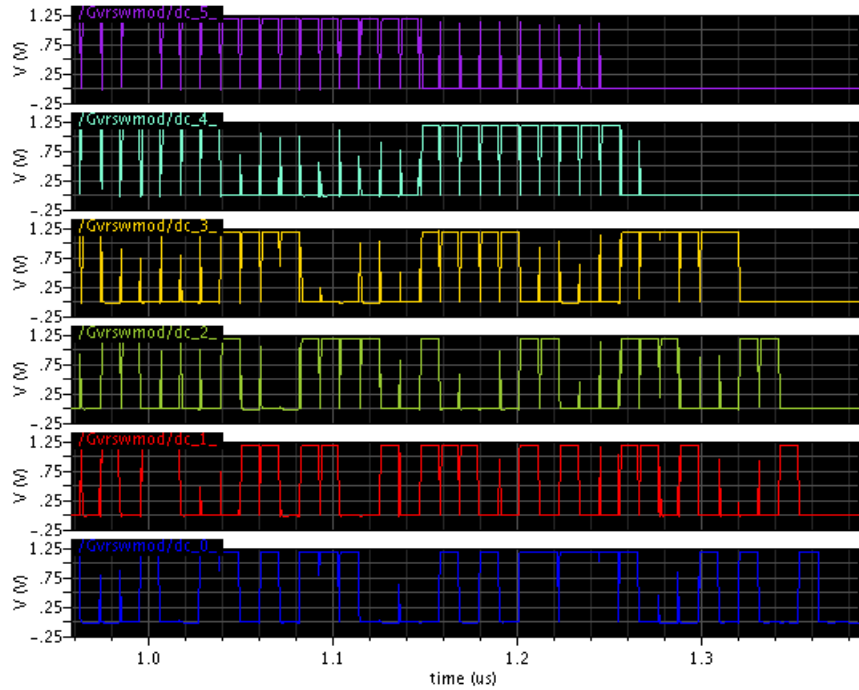


Figure 36: Min Data Rate's Fullness Effect on Full System

$$V_4 = V_4S_0 + V_4S_1G + V_3S_1G + S_2G$$

$$V_3 = V_3S_0 + V_4S_1L + V_2S_1G$$

$$V_2 = V_2S_0 + V_3S_1L + V_1S_1G$$

$$V_1 = V_1S_0 + V_2S_1L + V_0S_1G$$

$$V_0 = V_0S_0 + V_0S_1L + V_1S_1L + S_2L$$

Figure 37: Modulate Equations

4. Methodology, Results, and Analysis

4.1 Methodology

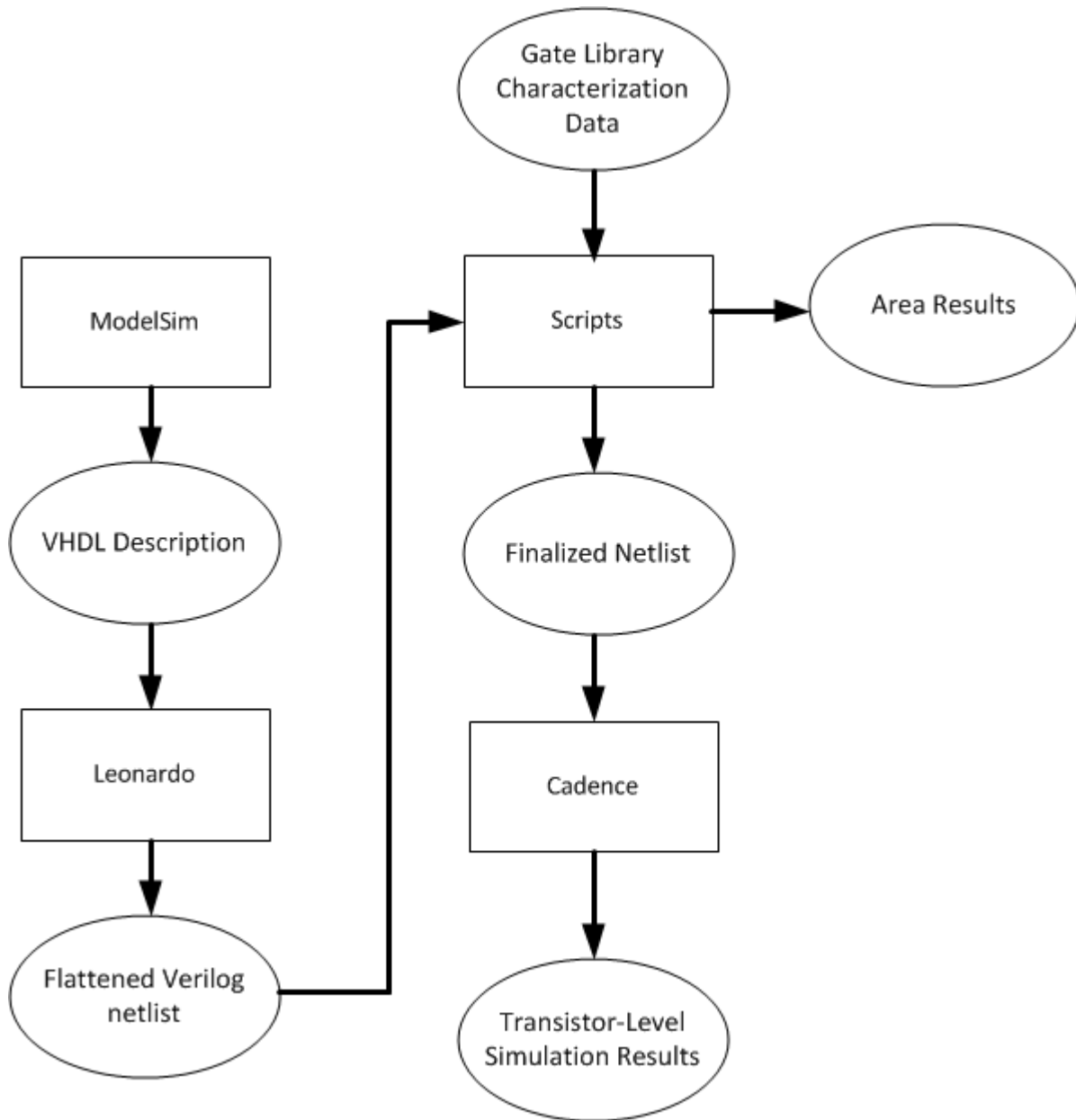


Figure 38. Design Flow

Figure 38 shows the design flow used for the final system's creation and analysis. The

final system's Parallelism, Core, and VCU logic were initially coded and simulated in VHDL using ModelSim. After behavioral verification, the VHDL description is converted into a gate-level Verilog netlist using Mentor's Leonardo. This netlist is run through several scripts which achieve the following functions: 1) Buffering; 2) Power connection and partitioning; and 3) Area comparison.

The buffering script uses gate characterization data to buffer high capacitive nets prioritizing gate replacement over buffer insertion. To achieve this, gate libraries are created for all of the 27 fundamental MTNCL gates and the limited number of NCL and Boolean gates required. Each gate is designed three times with each design having a different drive strength – A, B, or C. Drive strength A corresponds with minimum drive strength; drive strength B corresponds with roughly 4x drive strength; and drive strength C corresponds with roughly 16x drive strength. A gate's drive strength is denoted by appending $_{(a/b/c)}$ to its name, .e.g., $th23x0m_c$. The script ensures that the drive strength to capacitive load ratio for every net falls within a certain bound. When trying to meet these bounds, gate replacement (replacing a gate with a stronger version of itself) is prioritized over buffer insertion.

The power connection and partitioning script adds power/ground connections to each gate with the power connection dependent on the gate's system-level function. A gate's system-level function is determined through its instance name which represents its placement in the original hierarchical design.

The area comparison script groups gates based on their system-level functions and computes the gate-by-gate area for each system-level function.

After running through the aforementioned scripts, the resulting netlist was imported into Cadence and implemented at the transistor-level with the 130nm IBM 8RF-DM process. The V_{ref}

Generator and Voltage Regulator were also designed in Cadence using the same 130nm process, though later they were converted to VerilogA models.

4.1.1 Testbench

The final system's testbench is shown in Figure 39. The testbench consists of several VDC components and two main blocks: *Gvrsfin* and *Controller*. *Gvrsfin* consists of all the gate-level logic required to implement the system. *Controller* is a VerilogA model that has several responsibilities: 1) Manipulate data rate according to application scenario; 2) Generate and supply Cores' voltage; 3) Verify output data; and 4) Log output receipt times. The VDC components, along with one output from *Controller*, supply power to the system's four separate voltage domains named: 1) *Core_fixed* – the Core components receiving a fixed 1.2 V supply; 2) *Core* – Core components receiving the adaptable supply; 3) *Parallelism* – Parallelism components which receive a fixed 1.2 V supply, and 4) *Control* – VCU logic which receives a fixed 1.2 V supply. The three-ring-register supplying input patterns is in a separate voltage domain which is excluded from energy calculations.

4.1.2 Simulation Procedure

All simulations are performed in Cadence's Analog Design Environment using the UltraSim simulator. The system is simulated at five different Core domain voltage levels: 1.2 V, 0.7 V, 0.65 V, 0.6 V, 0.53 V. These voltage levels correspond to the profiled fullness values and 0.53 V is the lowest operating voltage for the system. For these simulations, the three-ring-register is allowed to run at its maximum rate. This facilitates a near-maximum input data rate which equates to a DATA-to-DATA time of approximately 3.4ns. At each voltage level, the system computes the same sequence of 100 data patterns. This computation's execution time is

recorded as well as each voltage domain's current draw. From this information, each domain's energy consumption is calculated, and each voltage level's energy efficiency is determined.

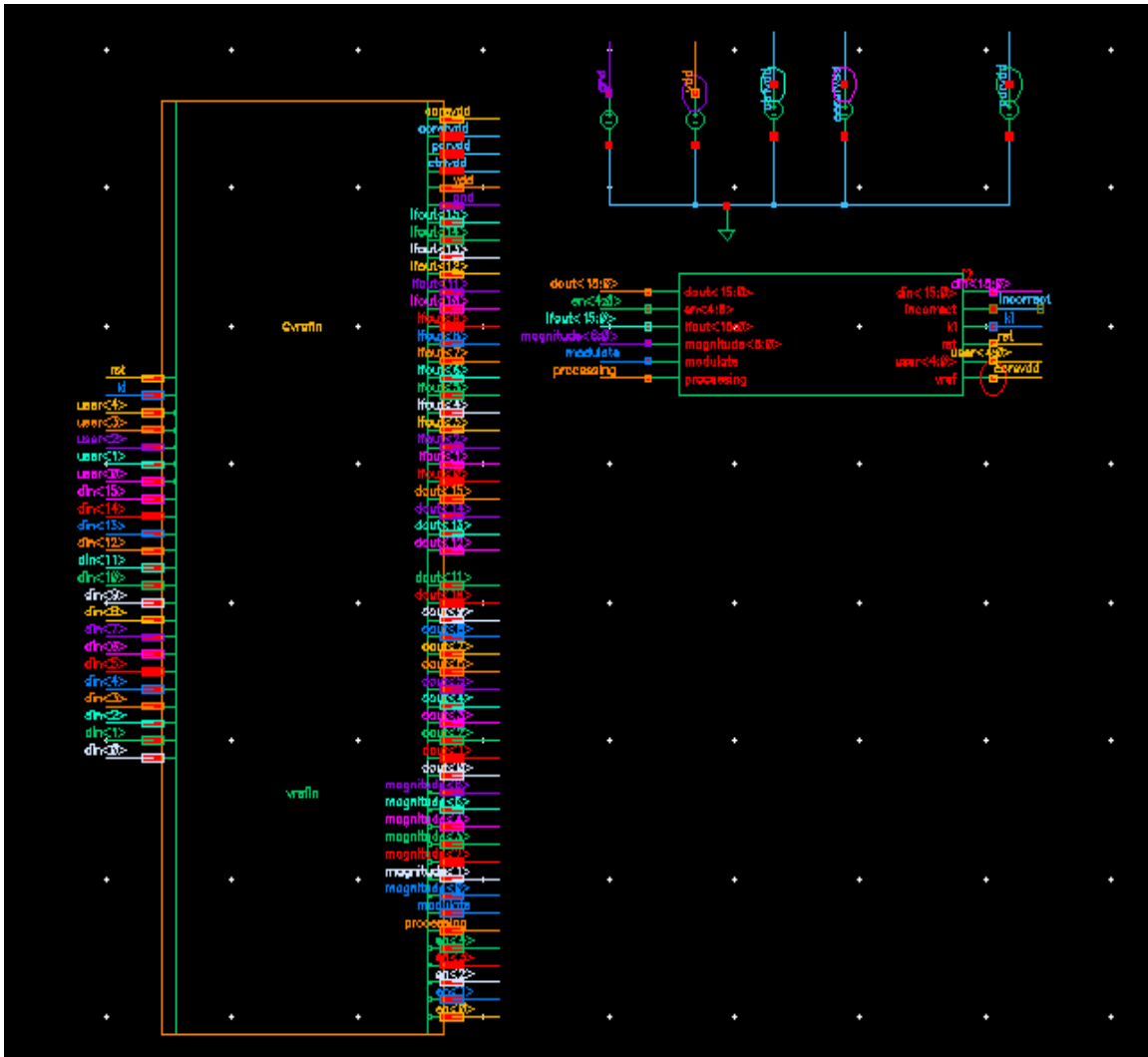


Figure 39. Simulation Testbench

4.2 Energy Results and Analysis

Figure 40 shows system execution time at each of the 5 simulated voltage levels (1.2 V, 0.7 V, 0.65 V, 0.6 V, 0.53 V). For the same 5 voltage levels, Figure 41 shows each voltage domain's energy usage and Figure 42 shows the total energy consumed per data computation

(100 total data computed at each voltage level).

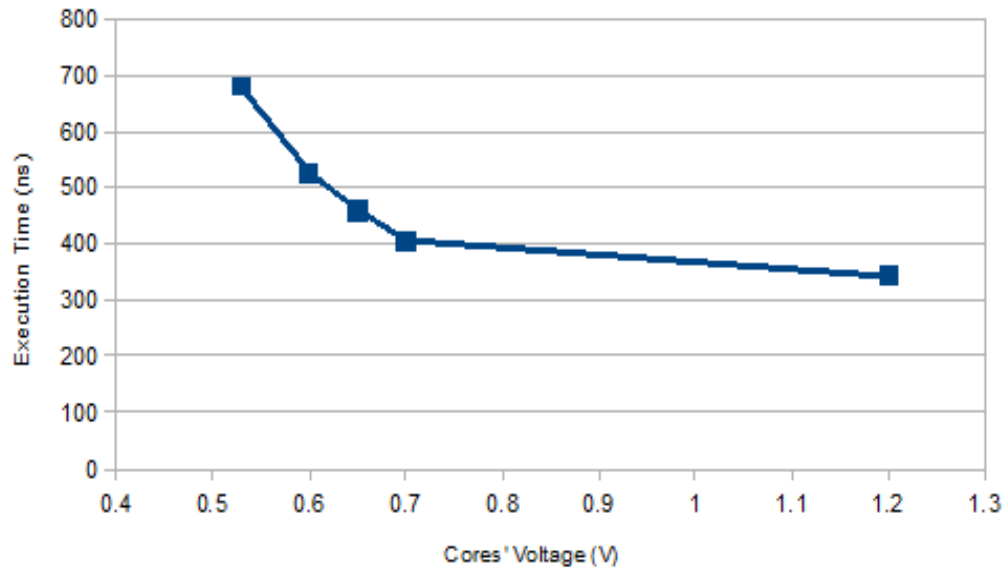


Figure 40. Execution Time

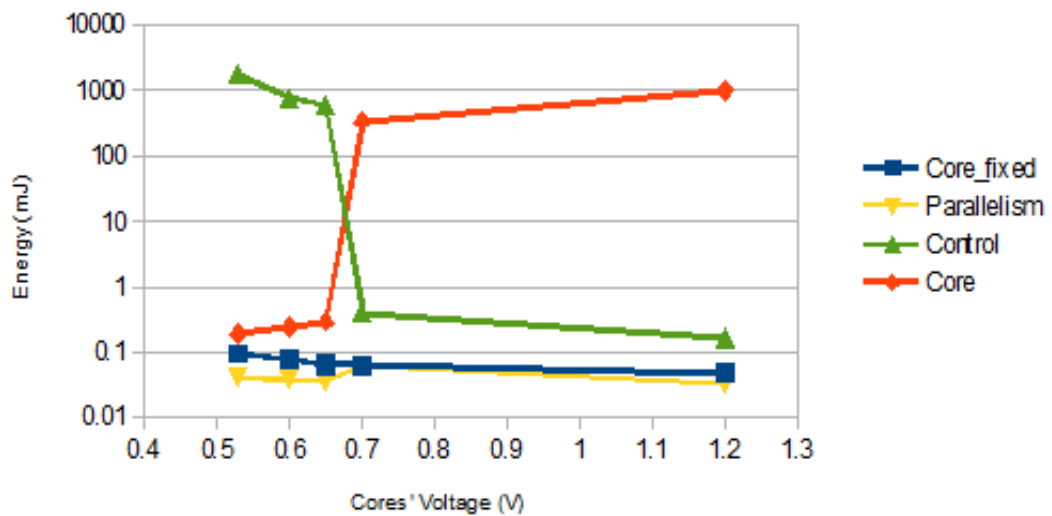


Figure 41. Energy Usage by Voltage Domain

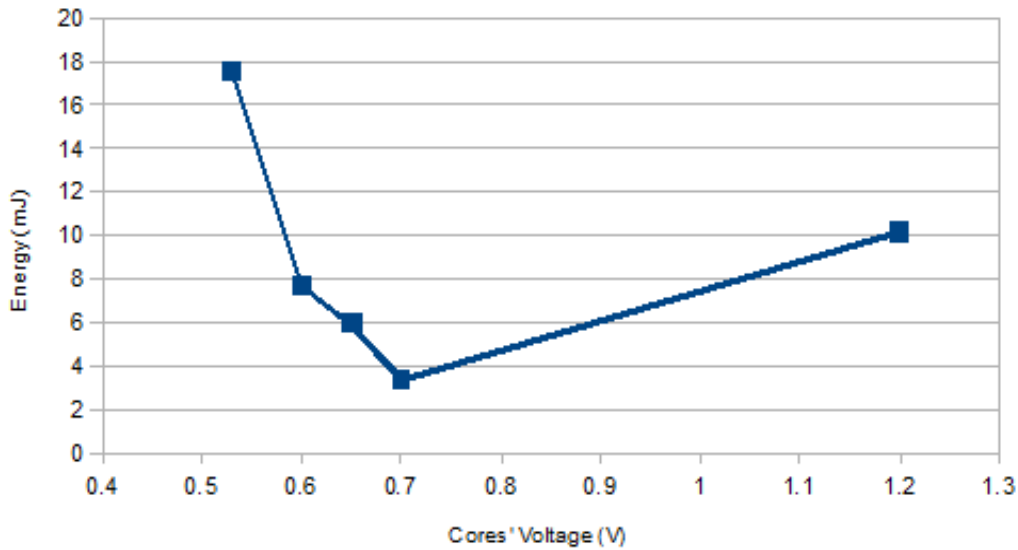


Figure 42. Energy per Computation

From Figure 41, Core_fixed and Parallelism domains' energy consumption remained relatively constant for each simulation. This is expected as each domain has a fixed 1.2 V supply and performs the same operations regardless of the Cores' supply. Core_fixed energy usage is slightly higher as that domain consists of slightly more gates than the Parallelism domain.

The Core domain's energy trends as expected. It follows a CMOS transistor's energy usage in the active/sub-threshold leakage regimes. The threshold voltage for this process is 0.6 V. As the Cores' voltage scales from 1.2 V to 0.65 V, the Core domain's energy decreased somewhat exponentially. This is expected as the transistors are still switching in the active regime where dynamic switching current is dominant. At 0.6 V, a sharp decrease in energy is observed indicating entry into near/sub-threshold regime where dynamic switching current is no longer a factor. At 0.53 V, a linear decrease from 0.6 V is observed as leakage current is now dominant.

Despite the increase in execution time as voltage scaled down, total Core energy for the computation decreased along the entire operational supply range. At 0.53 V, the system's fullness

is maximized which means that the number of “awake” stages is maximized. In the active regime, this should consume more energy, but in the sub-threshold regime the extra energy used by “awake” stages does not outweigh overall reduction in energy.

Figure 42 shows the system's total energy usage per computation. For this graph, the total energy consumed by each of the system's domains is summed and divided by the 100 data computed during the simulation. The graph essentially shows the system's energy efficiency at each of the five voltage levels. The system's energy efficiency follows a parabolic trend largely due to the Control domain's energy usage. Core domain energy decreases across the voltage range while Control domain energy increases. The Control domain's increase in energy even outpaces the Core domain's decrease in energy resulting in the system being less energy efficient at 0.53V than at 1.2 V.

The Control domain's energy usage is inversely proportional to the Core domain's energy. Figure 43 and Figure 44 show the Control domain's current waveform at Core voltages of 1.2 V and 0.53 V, respectively. Both the quiescent current and switching current is significantly higher at 0.53V. The peak current at 1.2 V often lies around 7.5mA; the peak current at 0.53 V often lies around 20 mA. The increased peak current is expected because, at lower voltages, there is significantly more switching which is attributed to the system's fullness level. A fuller system increases the likelihood of switching in the *Ko* signal register, DATA Detector, and RCA Tree; an emptier system greatly decreases the likelihood components switching. The increase in quiescent current is caused by low-voltage inputs being applied to gates on the 1.2V domain. Specifically, this occurs at the *Ko* signal register where most of its inputs are driven by gate outputs connected to the Core domain. With the *Ko* signal register's gates on a 1.2 V domain, at lower Core domain voltages, there is a significant potential difference from gate to source.

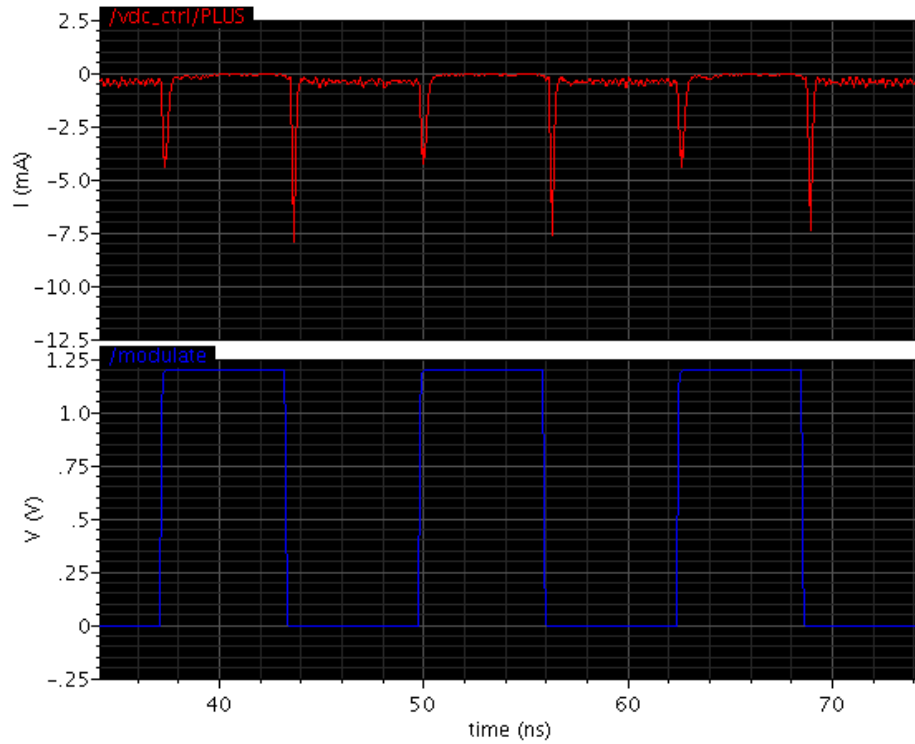


Figure 43. Control Domain Current at 1.2 V

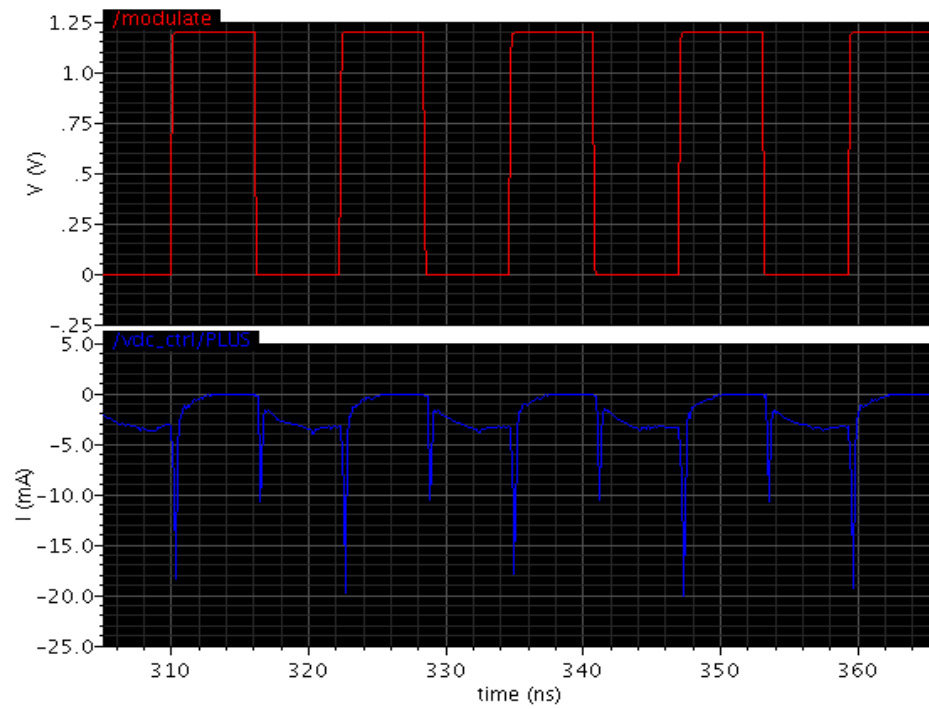


Figure 44. Control Domain Current at 0.53 V

To mitigate this, a level-converter was placed between each Ko signal and its input at Ko signal register. This resulted in the energy results shown in Figure 45 and Figure 46. The Core_fixed, Parallelism, and Core domains' energy trends remain the same. The Control domain's energy now trends in a more expected manner, increasing slightly as Core voltage is lowered. Furthermore the energy per data computed now decreases from 1.2 V to 0.60 V. Due to the exponential increase in computation time, the energy per computation increases slightly at 0.53 V.

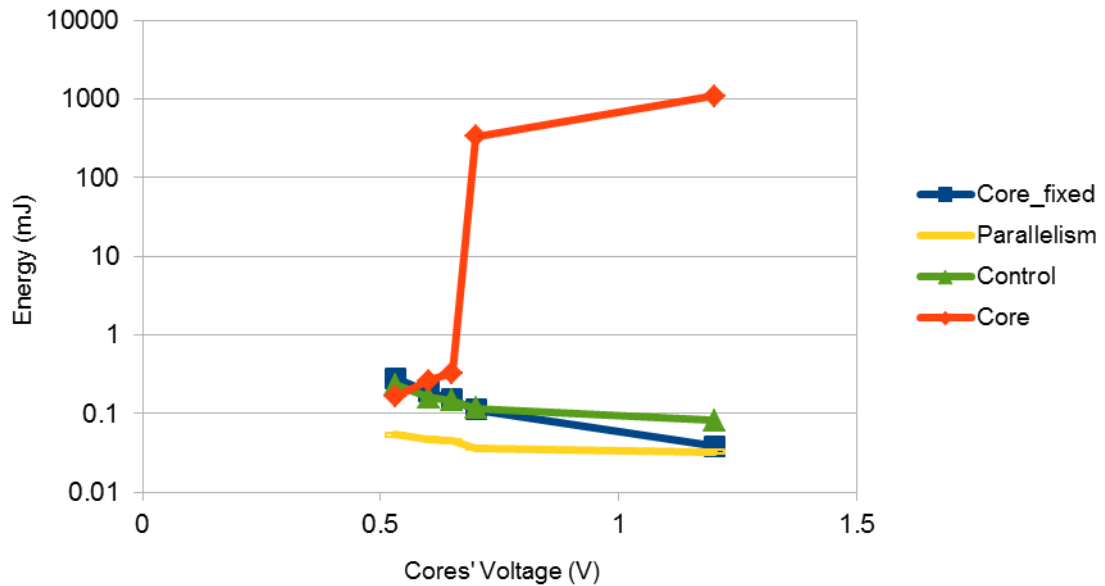


Figure 45. Energy Usage per Domain post Level-Converters

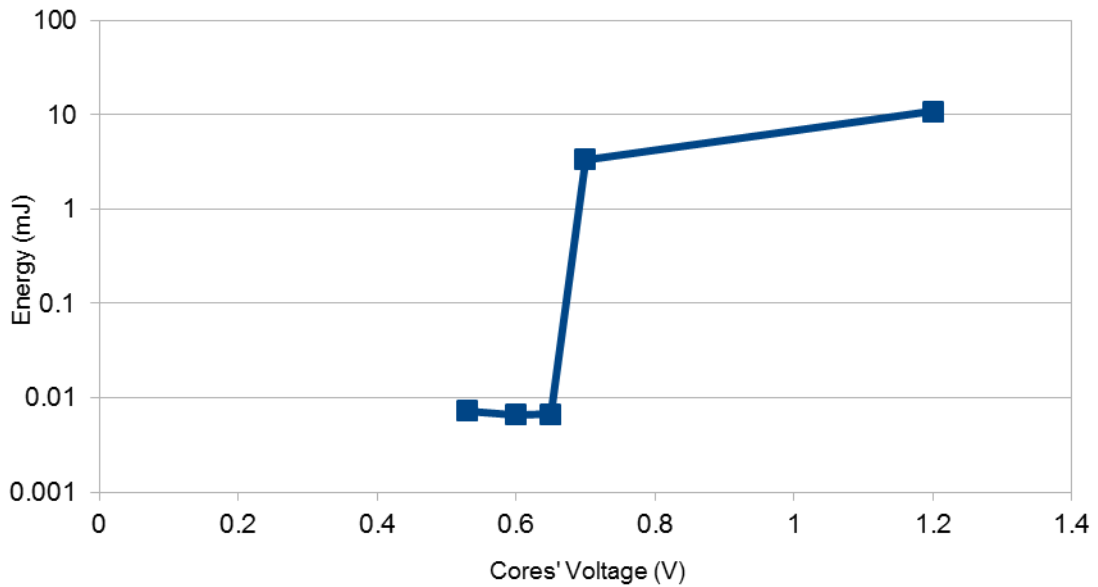


Figure 46. Energy per Computation post Level-Converters

4.3 Area Results and Analysis

Figures 47, 48, and 49 show the area required for the following logic sectors: 1) Core – this includes gates in the Core_fixed and Core domains; 2) Parallelism – this includes gates in the Parallelism domain; and 3) Control – this includes gates in the Control domain. Gate area measurements are taken from fabrication-ready layouts.

Core logic accounts for roughly 90% of the system's area. Most of this area consists of registration components. Registers, in the form of *th12x0m_a* gates, account for over half the area, while completion components, mainly consisting on *th24comp0m_a* gates, account for the next highest percentage, roughly one-fifth of Core logic's area.

Parallelism logic uses a small amount of area, taking up only 1% of the system's total

area. The demultiplexer, multiplexer, and sequencers use a fairly equal amount of this area; roughly one-third of that 1% is used by each.

Control logic accounts for about 9% of the total area, the largest portion of which is D flip-flops for the *Ko* and *EN* signal registers. Control logic overhead is largely based on the ratio between *Ko* signals and the average stage's area. As the ratio of *Ko* signals to the average stage's area increases, the area overhead of the Control logic increases; as the ratio of *Ko* signals to the average stage's area decreases, the area overhead of the Control logic decreases. Considering the basic Core design (4x4 pipelined multiplier) and its area per stage, the Control logic's area overhead would likely be less for larger designs. Of course, a significant increase in either the number of voltage references (unlikely due to fullness variance) or modulation algorithm complexity could challenge this idea.

Gate	Gate Count	Area (μm^2)	Count * Area (μm^2)
bufx0m_a	64	9.6	614.4
invx0_a	20	5.76	115.2
th22dx0_b	4	29.24	116.96
and4x0_b	1	16.17	16.17
and4x0_a	1	15.36	15.36
th14x0m_a	16	31.476	503.616
th33nx0_a	10	27.864	278.64
th33dx0_a	6	26.66	159.96
Total	122		1820.306

Figure 47. Parallelism Area

Gate	Gate Count	Area (μm^2)	Count * Area (μm^2)
th12x0m_a	3296	25.456	83902.976
th24comp0m_a	928	31.476	29209.728
th22nx0_a	128	27.864	3566.592
invx0_a	124	5.76	714.24
th22x0m_a	400	24.596	9838.4
th23x0m_a	224	37.496	8399.104
th34w2x0m_a	224	40.506	9073.344
th44w2x0m_a	32	39.302	1257.664
invx0_c	4	5.76	23.04
th44x0m_a	160	32.164	5146.24
th33x0m_a	48	25.886	1242.528
buffer_d	140	26.66	3732.4
buffer_e	16	26.66	426.56
Total	5724		156532.816

Figure 48. Core Area

Gate	Gate Count	Area (μm^2)	Count * Area (μm^2)
and2x0_a	379	9.6	3638.4
DFFS_E	129	36.48	4705.92
invx0_a	196	5.76	1128.96
or2x0_a	16	9.6	153.6
nor2x0_a	37	7.68	284.16
invx0_b	1	5.76	5.76
xnor2x0_a	6	13.44	80.64
and4x0_a	46	15.36	706.56
or3x0_a	100	11.96	1196
and3x0_a	173	13.4	2318.2
or4x0_a	45	15.64	703.8
DFFR_E	4	36.48	145.92
th22xn0_a	1	27.864	27.864
xor2x0_a	166	13.44	2231.04
buffer_d	5	26.66	133.3
Total	1304		17460.124

Figure 49. Control Area

5. Conclusions

The technical approach and results presented herein provide a basic platform for combining parallelism and DVS as they apply to DI asynchronous systems. A design flow is built which facilitates, in a mostly modular fashion, the implementation of parallelism and DVS for feed-forward MTNCL designs. Using this flow a system implementing these features is created and explored.

5.1 Summary

An MTNCL system's Ko signals indicate how many data it is currently processing. This is referred to as fullness. If the input data rate is faster than the output data rate, i.e., the rate at which data enters the system is faster than the rate at which data leaves the system, modulating the system's supply voltage can produce changes in fullness. However, to elicit an observable change in fullness, supply voltage must be modulated by an amount that is unique to each system. This introduces a theoretical limit to voltage precision. However, two other characteristics, fullness variance and fullness range, have a larger impact on voltage precision. Consideration of these characteristics leads to a voltage modulation algorithm which sustains system throughput across various data-rates. To accomplish the voltage modulation, a low-power voltage reference generator incorporating a novel shutdown technique is designed along with a voltage regulator. The control logic's area overhead and system's energy efficiency are also evaluated.

5.2 Future Work

While in this work an MTNCL system implementing parallelism and DVS has been created and unique system characteristics like fullness range and variance have been observed and explored, specific relationships between these characteristics and system parameters need to be studied further. Observations show a link between supply voltage and energy-efficiency. A connection between fullness and energy-efficiency needs to be determined for potential improvements to the voltage modulation algorithm. Although the VCU's logic overhead is within an acceptable margin and its energy usage trended as expected, VCU optimizations that further reduce the area and energy draw need to be explored. Lastly, the VCU shown herein takes a reactive approach to voltage modulation. Some investigations are needed to determine the viability and effectiveness of a VCU that is predictive in nature.

References

- [1] Flynn, M. (2007). *Computer Architecture*. John Wiley & Sons, Inc..
- [2] K. M. Fant and S. A. Brandt. "NULL Convention Logic: A Complete and Consistent Logic for Asynchronous Digital Circuit Synthesis." *International Conference on Application Specific Systems, Architectures, and Processors*, pp. 261-273, 1996.
- [3] L. Zhou, R. Parameswaran, R. Thian, S. C. Smith, and J. Di. "MTNCL: An Ultra-Low Power Asynchronous Circuit Design Methodology." Technical Report, 2010.
- [4] V. De. "Energy Efficient Designs with Wide Dynamic Range." Invited Talk, 2011 *IEEE Subthreshold Microelectronics Conference*, September 2011.
- [5] Sobelman, G. E., & Fant, K. "CMOS circuit design of threshold gates with hysteresis." *IEEE International Symposium on Circuits and Systems*, Vol. 2, pp. 61-64, June 1998.
- [6] A. Bailey, A. Al Zahrani, G. Fu, J. Di, and S. Smith. "Multi-Threshold Asynchronous Circuit Design for Ultra-Low Power." *Journal of Low Power Electronics*, Vol. 4, NO. 3, pp. 337-348, December 2008.
- [7] L. Zhou, S. Smith, and J. Di. "Bit-Wise MTNCL: an Ultra-Low Power Bit-Wise Pipelined Asynchronous Circuit Design Methodology." *2010 IEEE Midwest Symposium on Circuits and Systems*, August 2010.
- [8] A. Alzahrani, A. Bailey, G. Fu, and J. Di. "Glitch-Free Design for Multi-Threshold CMOS NCL Circuits." *2009 Great Lake Symposium on VLSI*, May 2009.
- [9] A. Bailey, J. Di, S. C. Smith, and H. A. Mantooth. "Ultra-Low Power Delay-Insensitive Circuit Design." *2008 IEEE Midwest Symposium on Circuits and Systems*, August 2008.
- [10] S. C. Smith and J. Di. *Designing Asynchronous Circuits using NULL Convention Logic (NCL)*. Morgan & Claypool Publishers, 2009.
- [11] Xu, R., Moss'e, D., and Melhem, R. "Minimizing expected energy consumption in real-time systems through dynamic voltage scaling." *ACM Transactions on Computer Systems* Volume 25, Issue 4, Article 9, December 2007.
- [12] V. Devadas and H. Aydin. "On the interplay of dynamic voltage scaling and dynamic power management in Real-Time embedded applications." *7th ACM International Conference on Embedded Software (EMSOFT'08)*, pp. 99-108, October 2008.

- [13] S. Y. Bang, K. Bang, S. Yoon, and E. Y. Chung. “Run-time adaptive workload estimation for dynamic voltage scaling.” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 9, pp. 1334–1347, 2009.
- [14] A. Philip. “Investigation of Energy and performance of delay insensitive asynchronous circuits with concurrency.” Master’s Thesis, University of Arkansas, Fayetteville, Arkansas, 2010.
- [15] S. Haykin. *Adaptive Filter Theory*, Upper Saddle River, NJ : Prentice Hall, 1996.
- [16] S. Herbert and D. Marculescu. “Analysis of Dynamic Voltage/Frequency Scaling in Chip-Multiprocessors”. *International Symposium on Low-Power Electronics and Design*, pp. 38–43, 2007.
- [17] S. Herbert and D. Marculescu. “Variation-Aware Dynamic Voltage/Frequency Scaling.” *IEEE 15th Symposium on High Performance Computer Architecture*, pp. 301-312, February 2009.
- [18] A.P. Chandrakasan *et al.* “Low-power CMOS digital design.” *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, 1992.
- [19] C. Xian, and Y. H. Lu. “Dynamic Voltage Scaling for Multitasking RealTime Systems with Uncertain Execution Time.” *ACM Great Lakes Symposium on VLSI*, pp. 392-397, 2006.
- [20] S. Hong, S. Yoo, B. Bin, K. M. Choi, S. K. Eo, and T. Kim. “Dynamic Voltage Scaling of Supply and Body Bias Exploiting Software Runtime Distribution.” *Design Automation and Test in Europe*, pp. 242-247, 2008.
- [21] S. Dighe *et al.* “Within-Die Variation-Aware Dynamic-Voltage-Frequency-Scaling With Optimal Core Allocation and Thread Hopping for the 80-Core TeraFLOPS Processor.” *IEEE Journal of Solid-State Circuits*, Vol. 46, No. 1, pp. 184-193, 2010.
- [22] B. H. Calhoun and A. P. Chndrakasan. “Ultra-Dynamic Voltage Scaling (UDVS) Using Sub-Threshold Operation and Local Voltage Dithering” *IEEE Journal of Solid-State Circuits*, Vol. 41, No. 1, pp. 238-245, January 2006.
- [23] V. Gutnik and A. P. Chandrakasan. “Embedded power supply for lowpower DSP,” *IEEE Transactions on VLSI Systems*, Vol. 5, No. 4, pp. 425–435, December 1997.
- [24] H.Kawaguchi, G.Zhang, S.Lee, Y.Shin,and T.Sakurai. “A controller LSI for realizing VDD-hopping scheme with off-the-shelf processors and its application to MPEG4 system.” *IEICE Trans. Electron.*, vol.E85-C, no. 2, pp. 263–271, Feb. 2002.