Delay-insensitive ternary logic (DITL)

Ravi Sankar Parameswaran Nair

DELAY-INSENSITIVE

TERNARY LOGIC

(DITL)

by

RAVI SANKAR PARAMESWARAN NAIR

A THESIS

Presented to the Faculty of the Graduate School of the

UNIVERSITY OF MISSOURI-ROLLA

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN COMPUTER ENGINEERING

2007

Approved by

Scott C. Smith, Advisor

Waleed K. Al-Assadi

Daryl G. Beetner

ABSTRACT

This thesis focuses on development of a Single Rail Ternary Voltage Delay-Insensitive paradigm called Delay-Insensitive Ternary Logic (DITL), which is based on NULL Convention Logic (NCL). Single rail asynchronous logic has potential advantages over Dual-Rail logic such as reduction of Power and Interconnect as well as Logic Area.

The DITL concept is developed in steps of individual circuit components. These components are designed at the transistor level and are connected together to form a registered pipeline system. Some variations in pipeline design are also investigated. Equivalent circuits are then designed using standard NCL for comparison to the DITL systems.

For both NCL and DITL designs, the transistor level netlist of the system is simulated using a VHDL testbench along with Mentor Graphics' ADvanced Mixed Signal simulation (ADMS) tool. The DITL and equivalent NCL systems are compared in terms of Area, Energy Usage, and Performance.

ACKNOWLEDGMENTS

I am honored to work in such a positive environment provided by the University of Missouri Rolla and the people of the Computer Engineering Department.

I am extremely grateful to my advisor Dr. Scott C. Smith for giving me an opportunity to work with him. It was a very pleasant and encouraging experience which gave me lots of knowledge and insights in the field.

I convey my sincere thanks to Dr. Waleed Al-Assadi and Dr. Daryl G. Beetner for serving on my committee.

I would also like to thank Samarsen Reddy and Bonita Bhaskaran for giving timely help at the initial stages of my research. A special thanks to Ms. Kathy Whyte for providing me with fast solutions to a handful of software problems I faced during my run. Last but not the least, I thank my loving family for their support in everyway that held me strong in my quest for knowledge.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

1.1. OVERVIEW OF ASYNCHRONOUS CIRCUITS

For the last three decades, the focus of digital design has been primarily on synchronous, clocked architectures. However, as clock rates have significantly increased while feature size has decreased, clock skew has become a major problem. High performance chips must dedicate increasingly larger portions of their area for clock drivers to achieve acceptable skew, causing these chips to dissipate increasingly higher power, especially at the clock edge, when switching is most prevalent. As these trends continue, the clock is becoming more and more difficult to manage, while clocked circuits' inherent power inefficiencies are emerging as the dominant factor hindering increased performance. These issues have caused renewed interest in asynchronous digital design.

Asynchronous, clockless circuits require less power, generate less noise, and produce less electro-magnetic interference (EMI), compared to their synchronous counterparts, without degrading performance. Furthermore, ***delay-insensitive (DI)*** asynchronous paradigms have a number of additional advantages, especially when designing complex circuits, like Systems-on-Chip (SoCs), including substantially reduced crosstalk between analog and digital circuits, ease of integrating multi-rate circuits, and facilitation of component reuse.

As demand increases for designs with higher performance, greater complexity, and decreased feature size, asynchronous paradigms will become more prevalent in the multi-billion dollar semiconductor industry, as predicted by the International Technology Roadmap for Semiconductors (ITRS) [1], which envisions a likely shift from synchronous to asynchronous design styles in order to increase circuit robustness, decrease power, and alleviate many clock-related issues. Furthermore, ITRS states that asynchronous circuits will account for 19% of chip area within the next 5 years, and 30% of chip area within the next 10 years [2].

Asynchronous circuits can be grouped into two main categories: *bounded-delay* and *delay-insensitive* models. Bounded-delay models, such as *micropipelines* [3], assume that delays in both gates and wires are bounded. Delays are added based on worse-case scenarios to avoid hazard conditions. This leads to extensive timing analysis of worse-case behavior to ensure correct circuit operation. On the other hand, delay-insensitive circuits, like NCL, assume delays in both logic elements and interconnects to be unbounded, although they assume that wire forks within basic components, such as a full adder, are isochronic, meaning that the wire delays within a component are much less than the logic element delays within the component, which is a valid assumption even in future nanometer technologies. Wires connecting components do not have to adhere to the isochronic fork assumption. This implies the ability to operate in the presence of indefinite arrival times for the reception of inputs. Completion detection of the output signals allows for handshaking to control input wavefronts. Delay-Insensitive design styles therefore require very little, if any, timing analysis to ensure correct operation (i.e., they are correct by construction), and also yield average-case performance rather than the worse-case performance of bounded-delay and traditional synchronous paradigms.

1.2. THESIS OBJECTIVE

This M.S. thesis is intended to familiarize the reader with the asynchronous Delay-Insensitive NULL Convention Logic (NCL) paradigm [4], and to develop an alternative Single Rail Delay-Insensitive paradigm, based on NCL, called Delay-Insensitive Ternary Logic (DITL).

Ternary, or 3-valued, logic utilizes three distinct voltage levels (e.g., Gnd, Vdd, and ½ Vdd) on a single wire to encode information. This can be used to implement functions in a base 3 algebraic system [5], as opposed to standard base 2 Boolean algebra. Ternary logic's three distinct values can also be utilized to represent a Delay-Insensitive (DI) asynchronous circuit's three logic states (i.e., DATA0, DATA1, and NULL) using only one wire, instead of two wires required for traditional dual-rail logic.

The foreseen advantages of substituting the usage of NCL dual-rail signals by a single-wire ternary signal is that the interconnect area decreases by half and the combinational logic components now only need to work on a single wire of information, which can result in considerable decrease in the number of transistors. Another possible advantage is reduced power/energy, when taking into consideration the switching nature of DI circuits. A DI circuit signal always switches from a DATA to a NULL and from a NULL to a DATA. In NCL dual-rail, the range of switching voltage is |Vdd|, since only one rail of the 2-wire pair switches during a DATA to NULL or NULL to DATA transition. However, for a DITL signal, this switching range is only |½ Vdd|. Speaking theoretically, the dynamic switching power of a DITL circuit should therefore be one quarter of that for an NCL dual-rail circuit [6].

This thesis investigates the advantages of implementing Delay-Insensitive Ternary Logic circuits at the transistor level, and compares the results, in terms of area, energy consumption, and speed of operation, to their functionally equivalent dual-rail NCL implementations.

1.3. THESIS OVERVIEW

This thesis is organized into four sections. Section 2 presents an overview of NCL. Section 3 reviews the previous work in the field of asynchronous ternary logic. In Section 4, the concept of DITL is developed and registration, combinational logic, and completion logic blocks are designed and implemented at the transistor level. Section 4 also includes simulation of the basic DITL building blocks, as well as both pipelined and non-pipelined DITL systems, and compares these to the equivalent dual-rail NCL implementations. Section 5 highlights the contributions of this thesis and provides direction for future research.

2. OVERVIEW OF NCL

NCL offers a self-timed logic paradigm where control is inherent with each datum. NCL follows the so-called “weak conditions” of Seitz’s Delay-Insensitive signaling scheme [7]. As with other self-timed logic methods, the NCL paradigm assumes that forks in wires are isochronic [8]. The origins of various aspects of the paradigm, including the NULL (or spacer or idle) logic state from which NCL derives its name, can be traced back to Muller’s work on speed-independent circuits in the 1950s and 1960s [9].

2.1. DELAY-INSENSITIVITY

NCL uses symbolic completeness of expression [4] to achieve Delay-Insensitive behavior. A symbolically complete expression is defined as an expression that only depends on the relationships of the symbols present in the expression without a reference to their time of evaluation. In particular, Dual-Rail signals, Quad-Rail signals, or other *Mutually Exclusive Assertion Groups* (MEAGs) can be used to incorporate DATA and control information into one mixed signal path to eliminate time reference [10]. A Dual-Rail signal, D, consists of two wires, D^0 and D^1, which may assume any value from the set {DATA0, DATA1, NULL}. The DATA0 state ($D^0 = 1$, $D^1 = 0$) corresponds to a Boolean logic 0, the DATA1 state
($D^0 = 0$, $D^1 = 1$) corresponds to a Boolean logic1, and the NULL state ($D^0 = 0$,
$D^1 = 0$) corresponds to the empty set meaning that the value of D is not yet available. The two rails are mutually exclusive, so that both rails can never be asserted simultaneously; this state is an illegal state.

A Quad-Rail signal, *Q*, consists of four wires, Q^0, Q^1, Q^2, and Q^3, which may assume any value from the set {DATA0, DATA1, DATA2, DATA3, NULL}. The DATA0 state ($Q^0 = 1$, $Q^1 = 0$, $Q^2 = 0$, $Q^3 = 0$) corresponds to two Boolean logic signals, *X* and *Y*, where X = 0 and Y = 0. The DATA1 state ($Q^0 = 0$, $Q^1 = 1$, $Q^2 = 0$, $Q^3 = 0$) corresponds to X = 0 and Y = 1. The DATA2 state ($Q^0 = 0$, $Q^1 = 0$, $Q^2 = 1$, $Q^3 = 0$) corresponds to X = 1 and Y = 0. The DATA3 state ($Q^0 = 0$, $Q^1 = 0$, $Q^2 = 0$, $Q^3 = 1$) corresponds to X = 1 and Y = 1, and the NULL state ($Q^0 = 0$, $Q^1 = 0$, $Q^2 = 0$, $Q^3 = 0$) corresponds to the empty set meaning that the result is not yet available. The four rails of a Quad-Rail NCL signal are mutually exclusive, so no two rails can ever be simultaneously asserted; these states are defined as illegal states. Both Dual-Rail and Quad-Rail signals are space optimal 1-out-of-N Delay-Insensitive codes, requiring two wires per bit. Other higher order MEAGs may not be wire count optimal; however, they can be more power efficient due to the decreased number of transitions per cycle.

2.2. LOGIC GATES

NCL differs from many other Delay-Insensitive paradigms in that these other paradigms only utilize one type of state-holding gate, the *C-element* [9]. A C-element behaves as follows: when all inputs assume the same value then the output assumes this value, otherwise the output does not change. On the other hand, all NCL gates are state-holding. Thus, NCL optimization methods can be considered as a subclass of the techniques for developing Delay-Insensitive circuits using a pre-defined set of more complex components, with built-in *Hysteresis*, or State-Holding behavior.

NCL uses threshold gates for its basic logic elements [11]. The primary type of threshold gate is the THmn *gate*, where $1 \leq m \leq n$, as depicted in Figure 2.1. THmn gates have *n* inputs. At least *m* of the *n* inputs must be asserted before the output will become asserted. Because NCL threshold gates are designed with hysteresis, all asserted inputs must be de-asserted before the output will be de-asserted. This ensures a complete transition of inputs back to NULL before asserting the output associated with the next wavefront of input DATA. Therefore, a THnn gate is equivalent to an n-input C-element and a TH1n gate is equivalent to an n-input OR gate. In the representation of a THmn gate, each of the *n* inputs is connected to the rounded portion of the gate; the output emanates from the pointed end of the gate; and the gate's threshold value, *m*, is written inside of the gate.

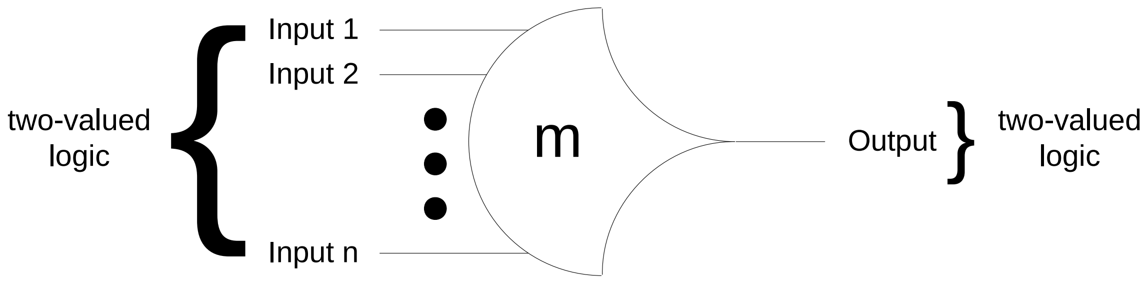
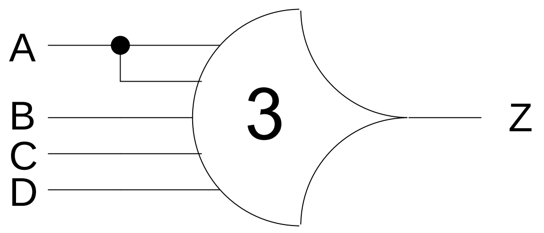


Figure 2.1 THmn Threshold Gate.

Another type of threshold gate is referred to as a Weighted threshold gate, denoted as THmnW$w_1w_2\ldots w_R$. Weighted threshold gates have an integer value, *m* ≥ w_R > 1, applied to *inputR*. Here 1 ≤ *R* < *n*; where *n* is the number of inputs; *m* is the gate's threshold; and w_1, w_2, …w_R, are the integer weights of *input1, input2*, … *inputR*, respectively. For example, consider a TH34W2 gate shown in Figure 2.2, whose *n* = 4 inputs are labeled *A*, *B*, *C*, and *D*. The weight of input *A*, *W (A)*, is therefore 2. Since the gate's threshold, *m*, is 3, this implies that in order for the output to be asserted, input *A* should be asserted along with either *B*, *C* or *D*; or all of *B*, *C*, and *D* should be asserted. NCL threshold gates may also include a *reset* input to initialize the gate's output. Resetable gates are denoted by either a *D* or an *N* appearing inside the gate, along with the gate's threshold, referring to the gate being reset to logic 1 or logic 0, respectively.



Z = AB + AC + AD + BCD

Figure 2.2 TH34W2 Weighted Threshold Gate and Output Set Equation.

Table 2.1 lists the 27 fundamental NCL gates, along with their corresponding Boolean equations, used to construct NCL circuits. These 27 gates constitute the set of all functions consisting of four or fewer variables. Since each rail of a NCL signal is considered a separate variable, a four variable function is not the same as a function of four literals, which would normally consist of eight variables. Twenty four of these gates can be realized using complex threshold gates, identical to the standard threshold gate forms for functions of four or fewer variables. The other three macros (i.e., THxor0, THand0, and TH24comp) could be constructed from threshold gate networks, but have been implemented as standard gates to provide completeness. Table 2.1 also contains the transistor count for these 27 gates.

By employing threshold gates for each logic rail, NCL is able to determine the output status without referencing time. Inputs are partitioned into two separate wavefronts, the NULL wavefront and the DATA wavefront. The NULL wavefront consists of all inputs to a circuit being NULL, while the DATA wavefront refers to all inputs being DATA, some combination of DATA0 and DATA1. Initially all circuit elements are reset to the NULL state. First, a DATA wavefront is presented to the circuit. Once all of the outputs of the circuit transition to DATA, the NULL wavefront is presented to the circuit. Once all of the outputs of the circuit transition to NULL, the next DATA wavefront is presented to the circuit. This DATA/NULL cycle continues repeatedly. As soon as all outputs of the circuit are DATA, the circuit's result is valid. The following NULL wavefront then transitions all of these DATA outputs back to NULL. When they transition back to DATA again, the next output is available. This period is referred to as the DATA-to-DATA cycle time, denoted as T_{DD}, and has an analogous role to the clock period in a synchronous system. One important thing to note is that NCL circuit outputs are glitch free and only change from a valid DATA value to NULL or from NULL to a valid DATA value. Therefore, an intermediate invalid output state (e.g., both rails of a dual-rail signal being simultaneously logic 1) cannot occur. This is ensured by utilizing the following two basic NCL design rules, input-completeness and observability.

Table 2.1. 27 Fundamental NCL Gates.

NCL Macro	Boolean Function	Transistor Count
TH12	A + B	6
TH22	AB	12
TH13	A + B + C	8
TH23	AB + AC + BC	18
TH33	ABC	16
TH23w2	A + BC	14
TH33w2	AB + AC	14
TH14	A + B + C + D	10
TH24	AB + AC + AD + BC + BD + CD	26
TH34	ABC + ABD + ACD + BCD	24
TH44	ABCD	20
TH24w2	A + BC + BD + CD	20
TH34w2	AB + AC + AD + BCD	22
TH44w2	ABC + ABD + ACD	23
TH34w3	A + BCD	18
TH44w3	AB + AC + AD	16
TH24w22	A + B + CD	16
TH34w22	AB + AC + AD + BC + BD	22
TH44w22	AB + ACD + BCD	22
TH54w22	ABC + ABD	18
TH34w32	A + BC + BD	17
TH54w32	AB + ACD	20
TH44w322	AB + AC + AD + BC	20
TH54w322	AB + AC + BCD	21
THxor0	AB + CD	20
THand0	AB + BC + AD	19
TH24comp	AC + BC + AD + BD	18

2.3. COMPLETENESS OF INPUT

The completeness of input criterion [4], which NCL combinational circuits must maintain in order to be Delay-Insensitive requires that:

1. All outputs of a combinational circuit may not transition from NULL to DATA until all inputs have transitioned from NULL to DATA, and
2. All outputs of a combinational circuit may not transition from DATA to NULL until all inputs have transitioned from DATA to NULL.

In circuits with multiple outputs, it is acceptable, according to Seitz's weak conditions [7], for some of the outputs to transition without having a complete input set present, as long as all outputs cannot transition before all inputs arrive.

2.4. OBSERVABILITY

There is one more condition that must be met to ensure delay-insensitivity for NCL circuits. No orphans may propagate through a gate [12]. An orphan is defined as a wire that transitions during the current DATA wavefront, but is not used in the determination of the output. Orphans are caused by wire forks and can be neglected through the isochronic fork assumption [8], as long as they are not allowed to cross a gate boundary. This observability condition, also referred to as indicatability or stability, ensures that every gate transition is observable at the output, which means that every gate that transitions is necessary to transition at least one of the outputs.

2.5. NCL COMPONENTS

NCL systems contain at least two Delay-Insensitive registers, one at both the input and at the output. Two adjacent register stages interact through their request and acknowledge signals, K_i and K_o, respectively, to prevent the current DATA wavefront from overwriting the previous DATA wavefront, by ensuring that the two DATA wavefronts are always separated by a NULL wavefront. The acknowledge signals are combined in the Completion Detection circuitry to produce the request signal(s) to the previous register stage. Dual-rail NCL registration is realized through cascaded arrangements of single-bit dual-rail registers, depicted in Figure 2.3. The register consists of two TH22 gates that pass a DATA value at the input only when K_i is *request for data* (rfd) (i.e., logic 1) and likewise pass NULL only when K_i is *request for null* (rfn) (i.e., logic 0). They also contain a NOR gate to generate K_o, which is *rfn* when the register output is DATA and *rfd* when the register output is NULL. The registers shown below are reset to NULL, since all TH22 gates are reset to logic 0. However, the register could be instead reset to a DATA value by replacing exactly one of the TH22n gates with a TH22d gate.

An N-bit register stage, comprised of *N* single-bit dual-rail NCL registers, requires *N* completion signals, one for each bit. The NCL completion component, shown in Figure 2.4, uses these *N* K_o lines to detect complete DATA and NULL sets at the output of every register stage and request the next NULL and DATA set, respectively. In full-word completion, the single-bit output of the completion component is connected to all K_i lines of the previous register stage. Since the maximum input threshold gate is the TH44 gate, the number of logic levels in the completion component for an N-bit register is given by $\lceil \log_4 N \rceil$. Figures 2.5 and 2.6 show the flow of DATA and NULL wavefronts through an NCL combinational circuit (i.e. an AND function) and an arbitrary pipeline stage, respectively.

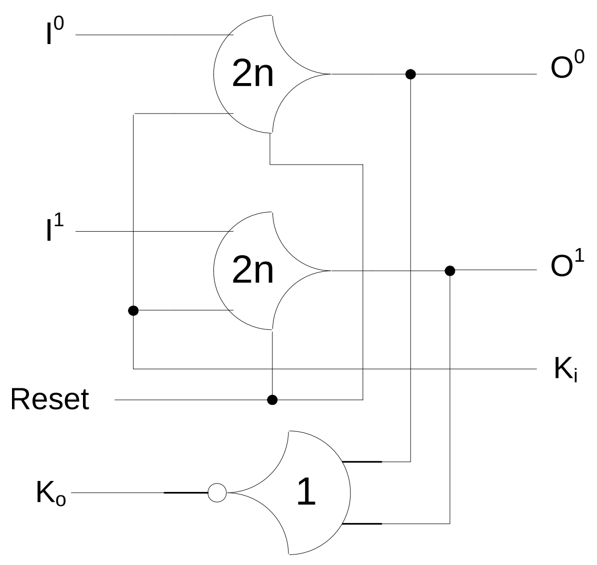


Figure 2.3 Single Bit Dual-Rail Register.

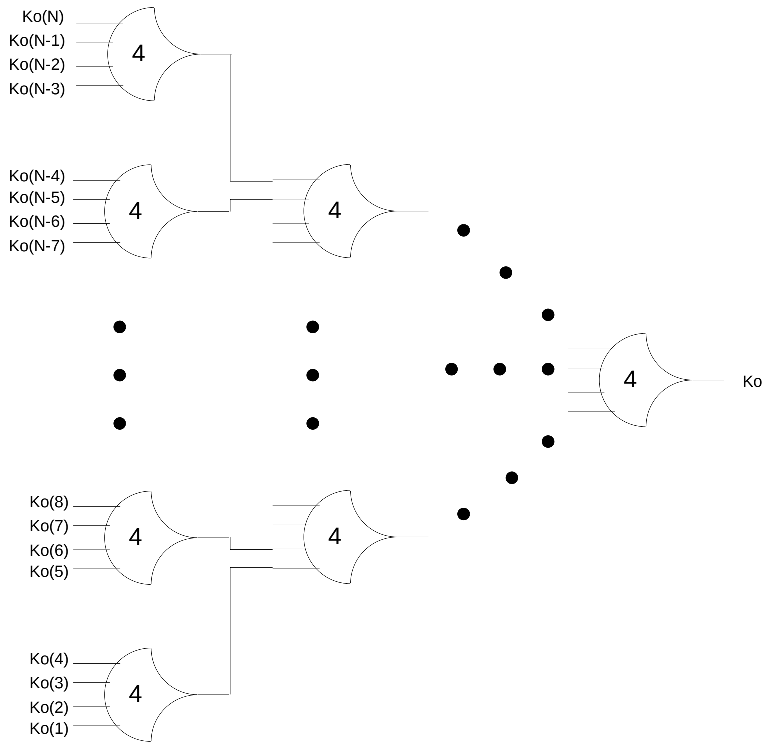


Figure 2.4 NCL Completion.

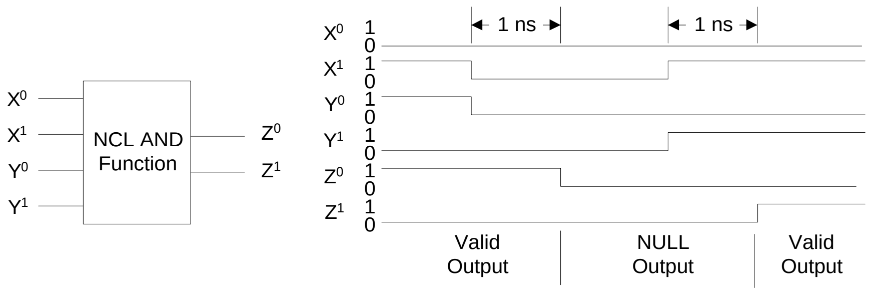
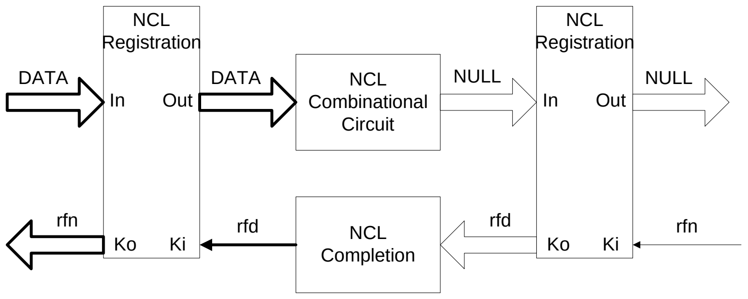
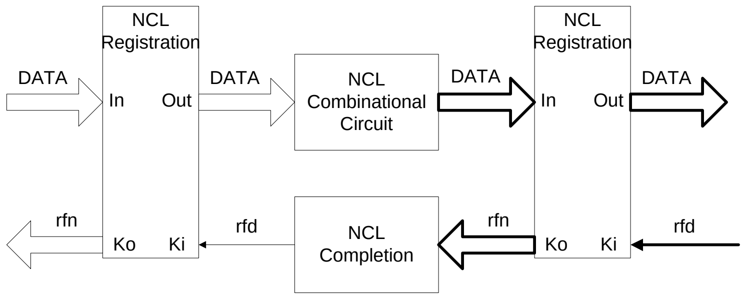


Figure 2.5 NCL AND Function: Z = X • Y.

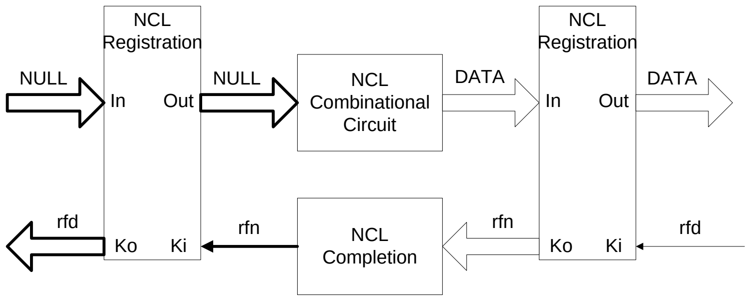
Initially *X*=DATA1 and *Y*=DATA0, so *Z*=DATA0; next *X* and *Y* both transition to NULL, so *Z* transitions to NULL; then *X* and *Y* both transition to DATA1, so *Z* transitions to DATA1.



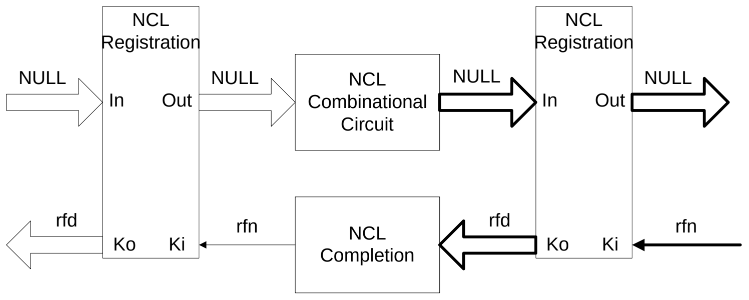
a) DATA flows through input register and combinational circuit.



b) DATA flows through output register and *rfn* flows through completion circuit.



c) NULL flows through input register and combinational circuit



d) NULL flows through output register and *rfd* flows through completion circuit.

Figure 2.6 NCL DATA/NULL Cycle.

3. PREVIOUS WORK

Ternary logic is a good alternative to dual-rail logic for implementation of asynchronous circuits, since it requires only one wire instead of two to represent the three logic values (i.e., DATA0, DATA1, and NULL). Vdd is used to represent DATA1, Gnd to represent DATA0, and ½ Vdd to represent NULL. This ternary encoding is optimal, yielding maximum noise margin with minimum switching power dissipation, since a wire always switches to NULL between two DATA values; hence, the voltage swing is always ½ Vdd. The previous work involving ternary logic for implementation of asynchronous circuits is described below.

References [13, 14] develop a ternary logic completion detection circuit for use with a Bounded-Delay self-timed paradigm; and [15, 16] develop a ternary Bounded-Delay self-timed paradigm, which is similar to micropipelines [3]. However, Bounded-Delay asynchronous paradigms are not as desirable as their Delay-Insensitive counterparts, as discussed in Section 1.1.

Reference [6] develops a delay-insensitive ternary logic transmission system, called Asynchronous Ternary Logic Signaling (ATLS), which converts dual-rail signals into ternary logic for transmission over a bus, in order to decrease transmission area and power. However, all of the logic processing is still done using dual-rail logic.

References [17, 18] develop a circuit called a *Watchful* as part of their proposed delay-insensitive ternary logic paradigm utilizing dynamic logic. However, as shown in the following timing diagram in Figure 3.1, their approach is not delay-insensitive because it assumes that the input will transition to NULL before *clear* is asserted, causing *full* to be deasserted. In order to be delay-insensitive, *full* must not be deasserted until both *clear* is asserted and *in* transitions to NULL. Otherwise, if the input remained at one DATA value (e.g., if no additional DATA needed to be processed at this time), this DATA value would continue to be utilized in subsequent operations instead of causing the system to become idle.

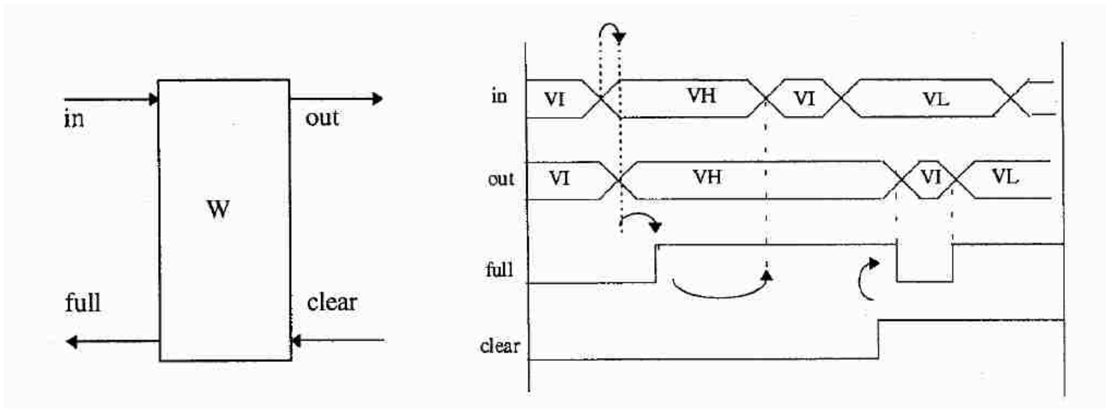


Figure 3.1 Watchful Timing Diagram [17].

Previous work in [19] utilizes shifted-threshold transistors in special inverters for detecting logic 0 and logic 1 in a CMOS ternary logic system, as shown in Figures 3.2 and 3.3, respectively. For Detect0, *in* must be lower than $-2\times Vt_P$ for the PMOS transistors to turn on and pull *out* to Vdd. Similarly for Detect1, *in* must be higher than $2\times Vt_N$ for *out* to be pulled down to Gnd. The truth table for Detect0 and Detect1 is given in Table 3.1. The Detect1 and Detect0 are used extensively in DITL and for an easier representation in schematics they will be replaced by oversized buffer symbols with a circle or a rectangle inside, respectively, as shown in the figures below.

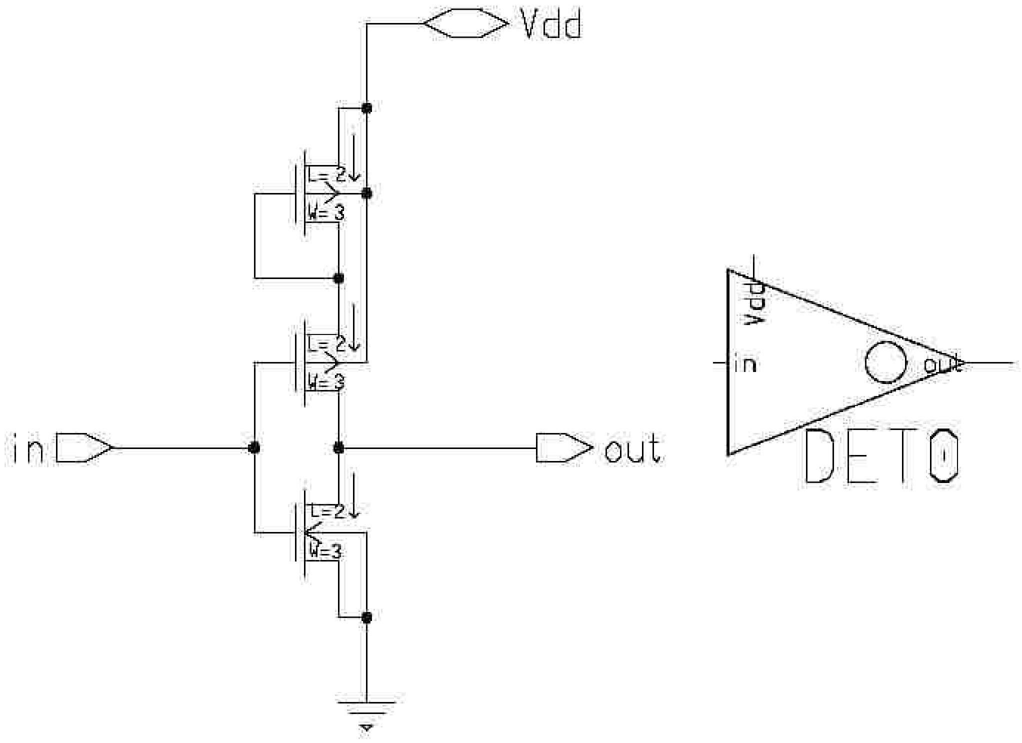


Figure 3.2 Schematic and Symbol of Detect0.

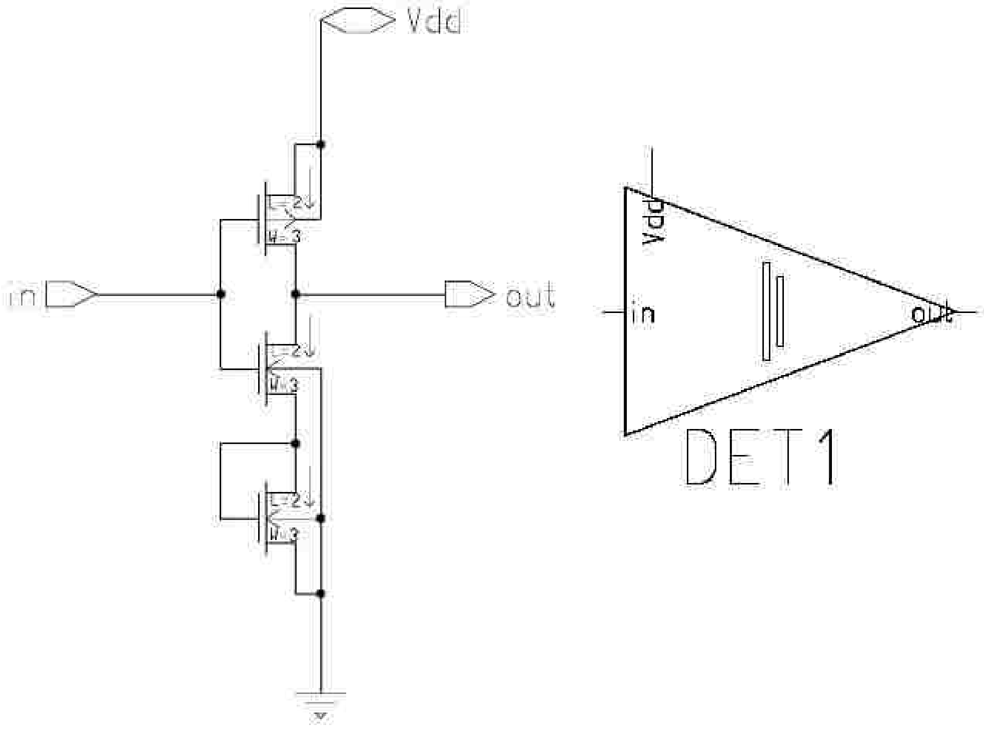


Figure 3.3 Schematic and Symbol of Detect1.

Table 3.1. Truth Table for Detect0 and Detect1.

Input A	Detect0 Output	Detect1 Output
Gnd or DATA0	1	1
½ Vdd or NULL	0	1
Vdd or DATA1	0	0

4. DEVELOPMENT AND DESIGN OF AN NCL BASED DELAY-INSENSITIVE TERNARY LOGIC

This section develops a fully delay-insensitive ternary logic paradigm, based on NCL, which utilizes static logic gates. Like other asynchronous ternary logic paradigms, DITL uses three voltage levels to represent the three states used for asynchronous signaling. Vdd represents DATA1; Gnd represents DATA0; and ½ Vdd represents NULL.

4.1. DITL CONCEPT DEVELOPMENT

A block diagram for one stage of a basic DITL system is shown in Figure 4.1. It consists of generic components such as Is-DATA, Completion Circuitry, Registration, and Ternary Combinational Logic. Register1 and Register2 are parallel load register stages. Inputs to Register1 may originate from a previous stage; and the Register2 outputs may be inputs to a subsequent stage.

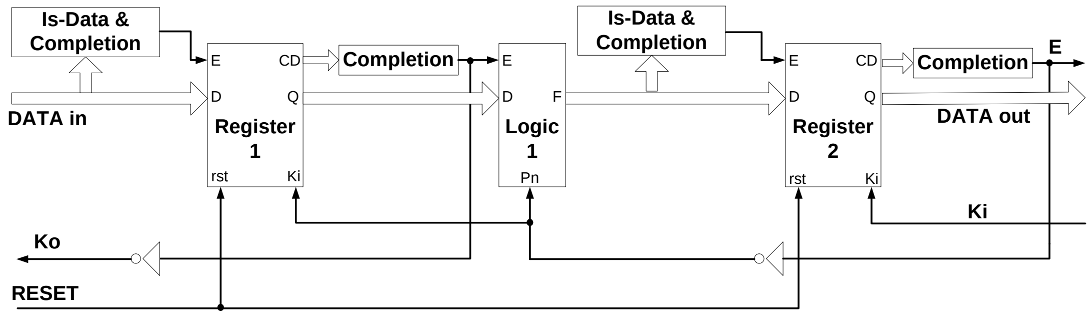


Figure 4.1 Block Diagram of a One-stage DITL System.

First, *RESET* is asserted to reset all registers to the NULL state, which results in all CD signals detecting NULL, which causes *Ko* to request DATA (i.e., become logic 1). Likewise, Register2 also requests DATA from Register1. As soon as a new DATA value is available at the input, Register1 latches it after the enable signal, *E*, produced by the Is-DATA component, is asserted. This DATA is then evaluated in the Logic1 component, when both *E* and *Pn* are asserted. Whenever the Logic1 block finishes evaluating, the resultant DATA appears at the input to Register2, and is latched after *Ki* is asserted. As a result, a request for NULL (i.e., logic 0) is generated from Register2 towards Register1. Since Register1's output is DATA, it requests for NULL by de-asserting *Ko*, at the same time that the Logic1 component is processing the DATA. When a NULL appears at the input to Register1, it is latched, only after *Ki* is logic 0. The NULL, now at the input of the Logic1 component, causes the enable signal, *E*, to be de-asserted, which along with *Pn* being logic 0, pulls the Logic1 output to NULL. Now, this NULL can be latched by Register2 after *Ki* is de-asserted. Hence, the outputs of both registers are now NULL, causing both to request the next DATA wavefront, which is the same as the initial state. This cycle repeats continuously. As shown in Figure 4.2, more than one Combinational Logic block can be integrated into a single pipeline stage.

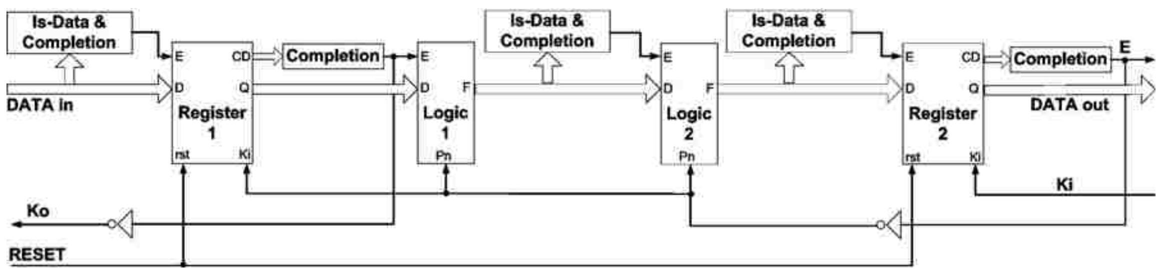


Figure 4.2 Block Diagram of a Double Logic Level DITL.

In this case, the Logic2 block is activated only after the Logic1 block is finished evaluating. Hence, arbitrary sized stages are possible. These circuits were first implemented and tested using VHDL, showing that the proposed ternary logic architecture and handshaking scheme was indeed delay-insensitive. The next step was transistor level realization of each of the basic ternary logic components.

4.2. DITL COMPONENTS AT TRANSISTOR LEVEL

4.2.1 Is-DATA. The Is-DATA component, shown in Figure 4.3, is used to detect when input *A* is either DATA or NULL, producing logic 1 on output *Y* when *A* is either DATA0 or DATA1 and logic 0 when *A* is NULL (i.e., ½ Vdd). It utilizes the *Detect0* and a modified version of the *Detect1* circuit, explained in Section 3. The Is-DATA component consists of 16 transistors.

For the original Detect1 circuit in Figure 3.3, a logic 1 input should produce a logic 0 output, which worked correctly as a standalone circuit and as part of a standalone Is-DATA component. However, when utilized as part of an Is-DATA component in a larger circuit, the output was sometimes too close to Vt_N, which caused the circuit to malfunction.

Figure 4.3 Schematic of Is-DATA Component.

To remedy this problem, a buffer (i.e., two series inverters) was added to the Detect1 circuit, as shown in Figure 4.4. Detect0 is always followed by an inverter in the Is-DATA component, so buffering was not required.

The simulation waveforms for the Is-DATA component are shown in Figure 4.5. The output of Detect0 is seen to be less than the ideal value of Vdd when the input is Logic 0; and the output of the unmodified Detect1 component is slightly higher than the ideal value of Gnd when the input is Logic 1. This is a direct consequence of threshold modification using shifted-threshold transistors.

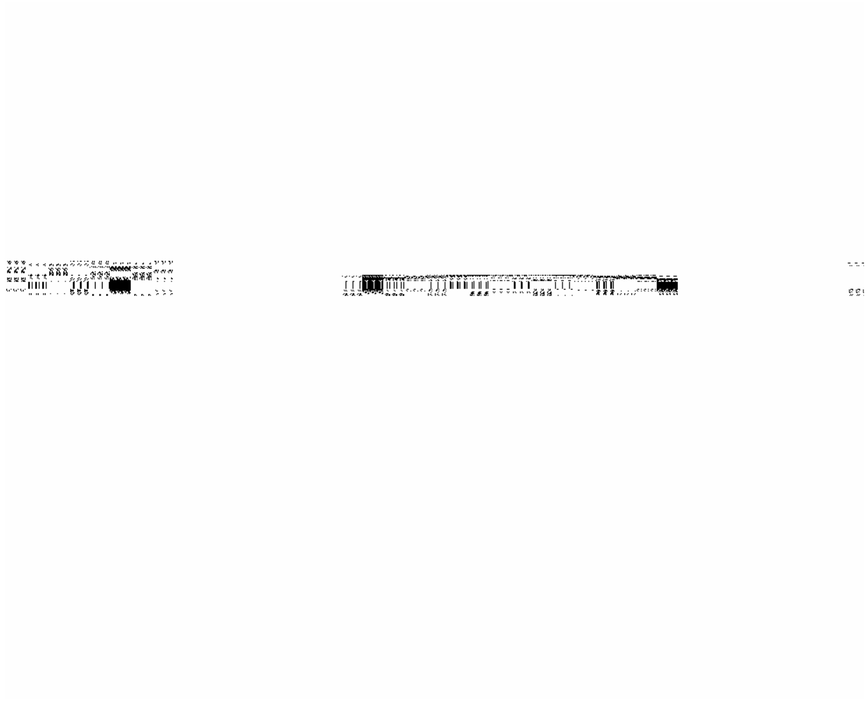


Figure 4.4 Modified Detect1 Circuit.

The Power dissipation waveform of the Is-DATA component is shown in Figure 4.5(b). It is found that a small but significant amount of energy is consumed continuously during intervals where the input is ½ Vdd. This is due to Static power dissipation and is seen as a non-zero slope line during the time intervals 10-20 ns,
30-40 ns, 50-60 ns, and 70-80 ns.

4.2.2 Completion. The Completion component combines multiple outputs of Is-DATA components into a single request signal. Since the Is-DATA outputs are either logic 0 or logic 1, and never ½ Vdd, the standard NCL completion component shown in Figure 2.4 can be used.

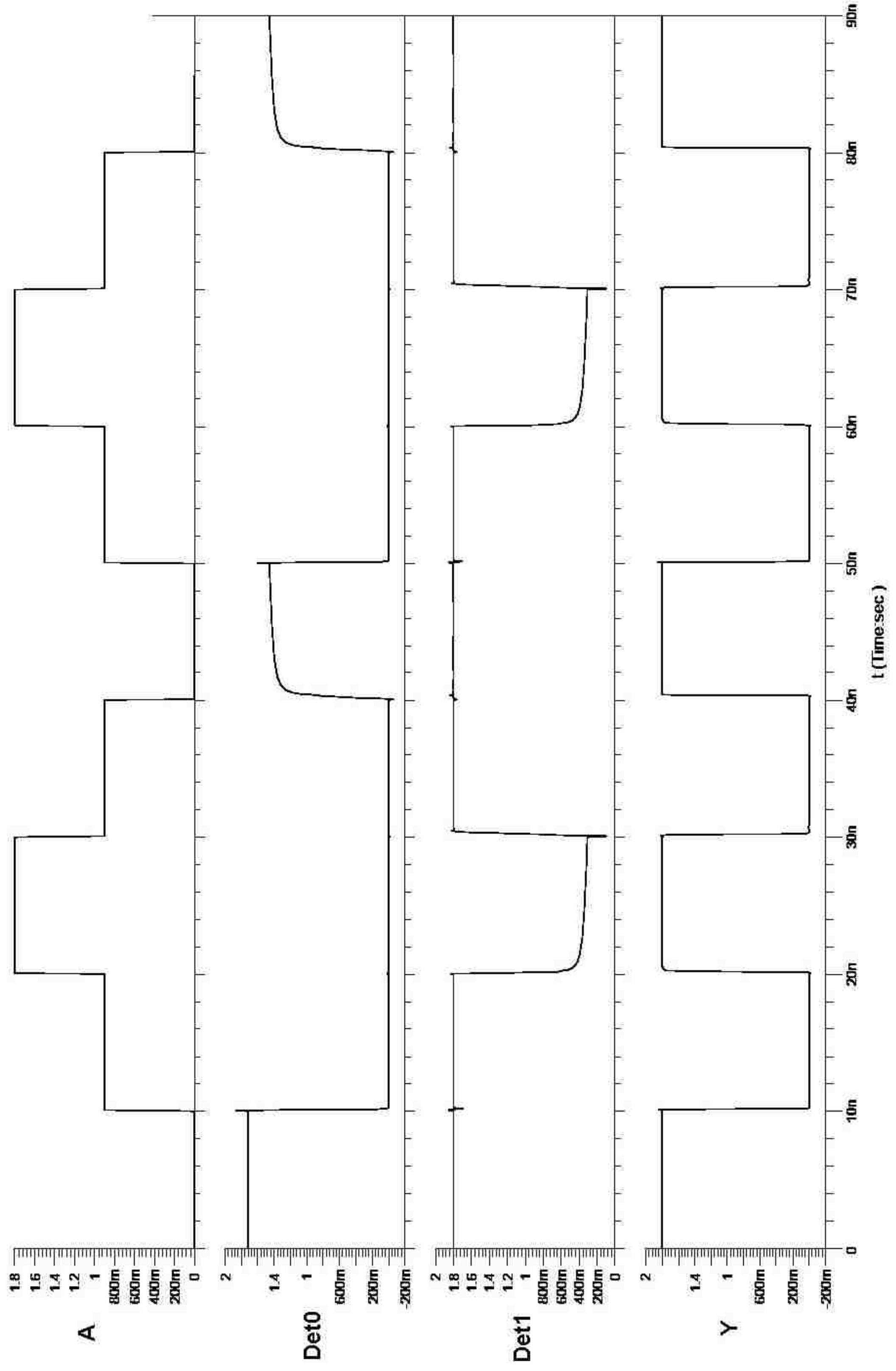


Figure 4.5 Simulation of Is-DATA. (a) Waveforms.

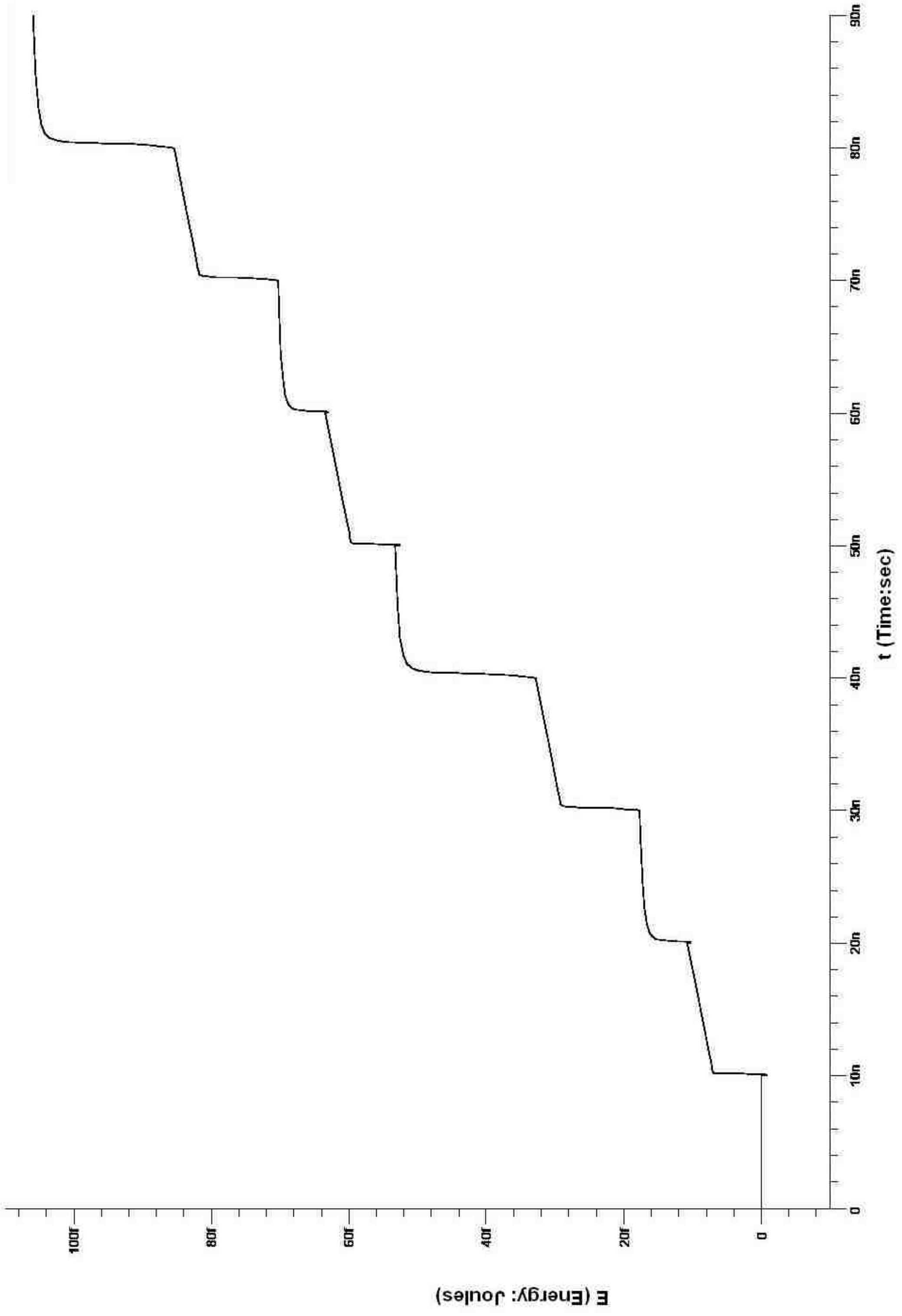


Figure 4.5 Simulation of Is-DATA (cont.). (b) Power.

4.2.3 DITL Register. The schematic of a DITL Register is shown below in Figure 4.6. It requires two voltage sources, Vdd and ½ Vdd (i.e., VddHf), and Gnd. It has a data input *D* and data output *Q*, both of which are ternary logic signals. A Boolean input, *E*, always changes following a change in *D* to show whether *D* is DATA or NULL. The handshake input signal, *Ki*, is obtained from the subsequent register (or primary input for the output register) and the output handshake signal, *CD*, is generated based on the status of *Q*. There is another input, *rst*, when asserted pulls output *Q* to the NULL state of ½ Vdd. Under normal system operation, *rst* is de-asserted. Note that *Ki*, *CD*, and *rst* are all Boolean logic signals.

When *Ki* is request for DATA (i.e., logic 1) and *D* is DATA, with *E* asserted, the pass transistors connect *D* to *Q*. The output, *Q*, is fedback into Detect1 and Detect0 circuits to produce a Hysteresis capability at *Q*, whereby the value of DATA at *Q*, whether DATA0 or DATA1, will not be modified until both *E* and *Ki* are de-asserted. *Q* switching to DATA causes output *CD* to become logic 1, indicating that the input has been latched. The Is-DATA circuit detailed earlier is exactly replicated here to generate *CD*.

When *Ki* is request for NULL (i.e., logic 0) and *D* is NULL, with *E* de-asserted, then a PMOS network connects ½ Vdd to *Q*, storing NULL in the Register output. Note that this forces the output to remain DATA until both a NULL is requested and the register input becomes NULL, thus fixing the problem in [17, 18] detailed in Section 3. If *E* or *Ki* is asserted before the other, *Q* becomes floating, charged to ½ Vdd, with no weak discharge paths to Gnd. If left floating long enough, the ½ Vdd will discharge, but will not be able to drive any subsequent logic, and will therefore not be mistaken for a DATA0.

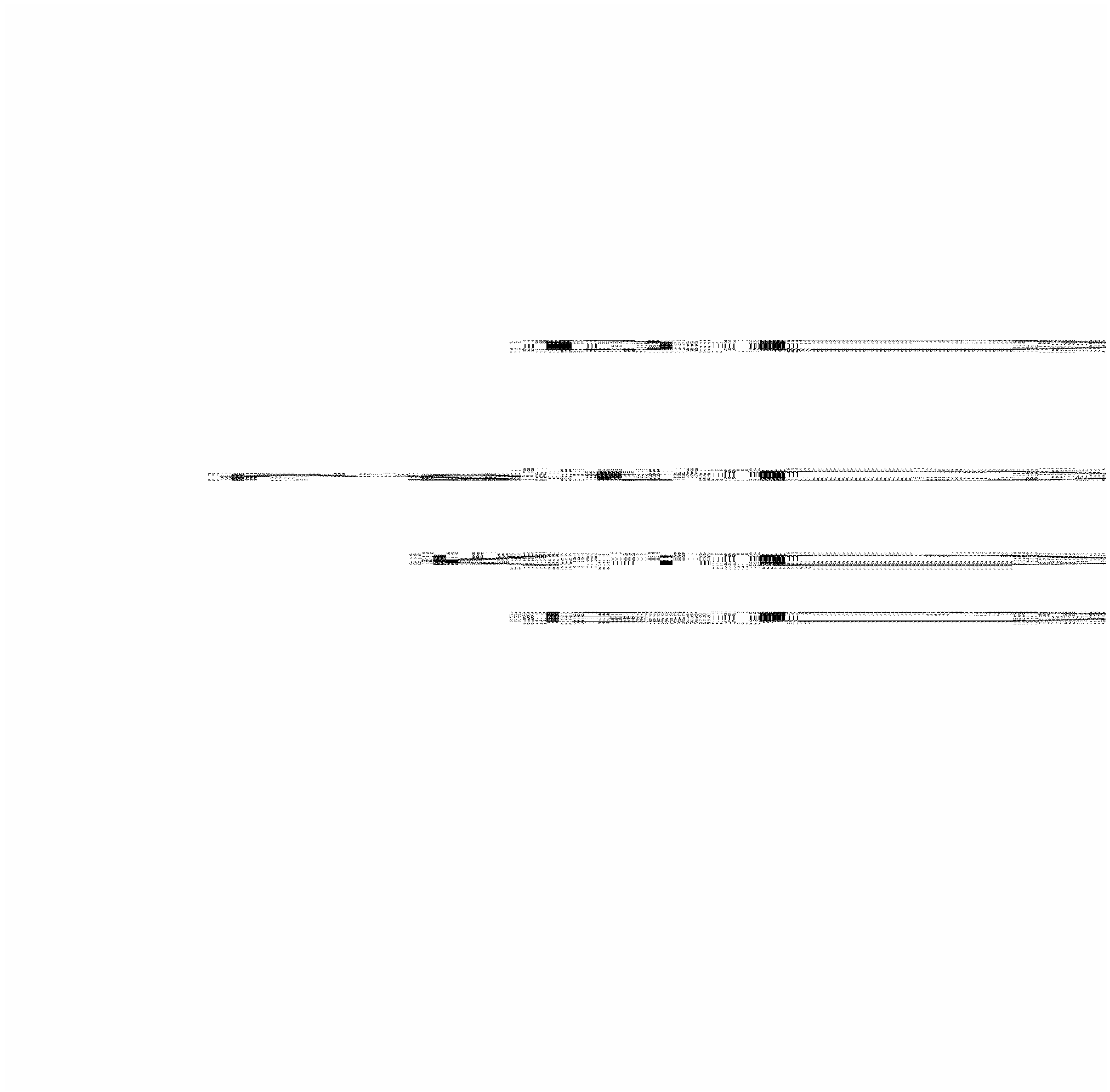


Figure 4.6 Schematic of DITL Register Reset to NULL.

The DITL Register consists of 39 transistors, one of which (i.e., PULL) is sized to be at least 3 times wider than the others, since it may need to overpower other transistors during reset. The register can also be modified to be reset to DATA1 instead of NULL by connecting the PULL transistor to Vdd instead of ½ Vdd, or this transistor can be replaced with an NMOS transistor connected to Gnd and controlled by *rst'* for resetting to DATA0. The simulation waveforms of the Register circuit are shown in Figure 4.7.

4.2.4 Combinational Logic. Unlike the NCL paradigm that utilizes special gates to implement logic circuits, DITL uses standard Boolean gates (e.g., AND, NAND, OR, NOR, XOR, XNOR, etc.), modified to accommodate ternary logic inputs and outputs. Take a 2-input DITL NAND gate shown in Figure 4.8 for example. Ternary inputs *D1* and *D2* are the data inputs; *E* shows the status of both *D1* and *D2* and is asserted after both become DATA and de-asserted after both become NULL. When *E* is asserted, the logic evaluates and produces the output DATA on *F*. *Pn* is another control input connected to the request line from the down stream register. When *Pn* is de-asserted, it is a request for NULL, but *F* does not become NULL until all inputs transition to NULL, indicated by *E* being de-asserted, thus enforcing input-completeness and preserving delay-insensitivity, as shown in the simulation of Figure 4.9.

In a similar fashion, DITL versions of all Boolean gates, or any arbitrary logic function, can be realized. However, utilizing standard Boolean gates is advantageous

Figure 4.7 Simulation of DITL Register.

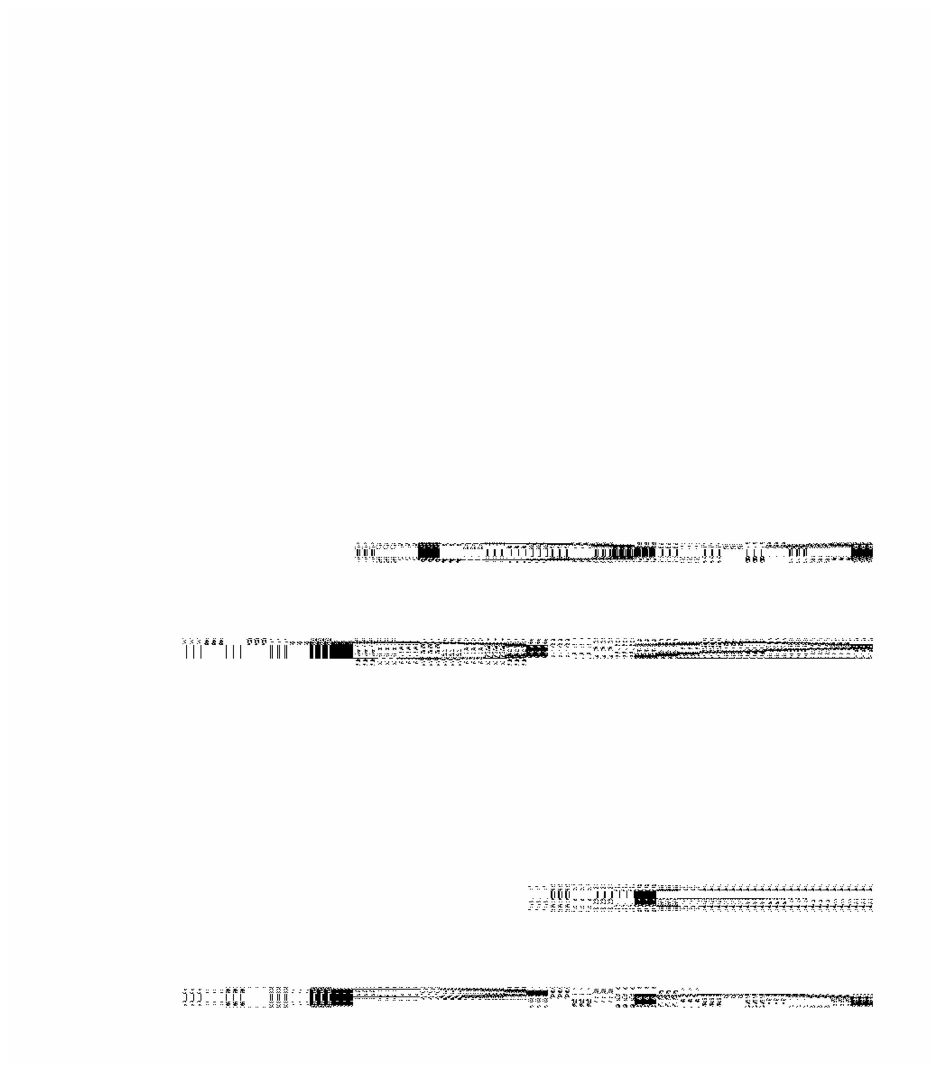


Figure 4.8 Schematic of a DITL NAND2 Gate.

because this will allow standard CAD tools to be utilized for synthesizing DITL circuits, requiring only slight modifications.

As an example, a DITL NAND4 gate is also shown below in Figure 4.10. The PMOS transistors controlled by *E* and *Pn*, used to pull *F* to ½ Vdd mainly determine the speed of DITL logic gates. This charging of *F* from Gnd to ½ Vdd can be sped-up by increasing the size of these PMOS transistors or replacing then with NMOS transistors.

4.3. SYSTEM LEVEL IMPLEMENTATION OF DITL

After creating the individual DITL components, they were connected together into registered pipeline systems as discussed in Section 4.1 and illustrated in Figure 4.11.

4.3.1 DITL Stages. A single-stage design with one logic block, call it *Stage1*, as shown in Figure 4.11 (a) was designed first. Stage1 has eight ternary inputs going into an eight-bit DITL register, followed by two DITL NAND4 gates, whose outputs are fed into a two-bit DITL register, after which they are taken as the system outputs. Thus, Stage1 has one level of Combinational Logic between two Registers.

Note that there is extra logic added before the input register, called Ternary Voltage Augmentation, which is required for VHDL controlled transistor-level simulation of ternary circuits to generate the ½ Vdd logic level, since the simulator only allows binary inputs. This extra logic is not required in the actual physical implementation, where three voltage levels will be used as inputs. Furthermore, the circuits could have been simulated without interfacing with a VHDL testbench, which also would not have required the extra logic; however, utilizing VHDL is very advantageous, as detailed in Section 4.4. This extra logic consists of pass-transistors that connect the *D0-D7* circuit inputs to the register inputs when *N* is asserted, and set *D0-D7* to ½ Vdd when *N* is de-asserted.

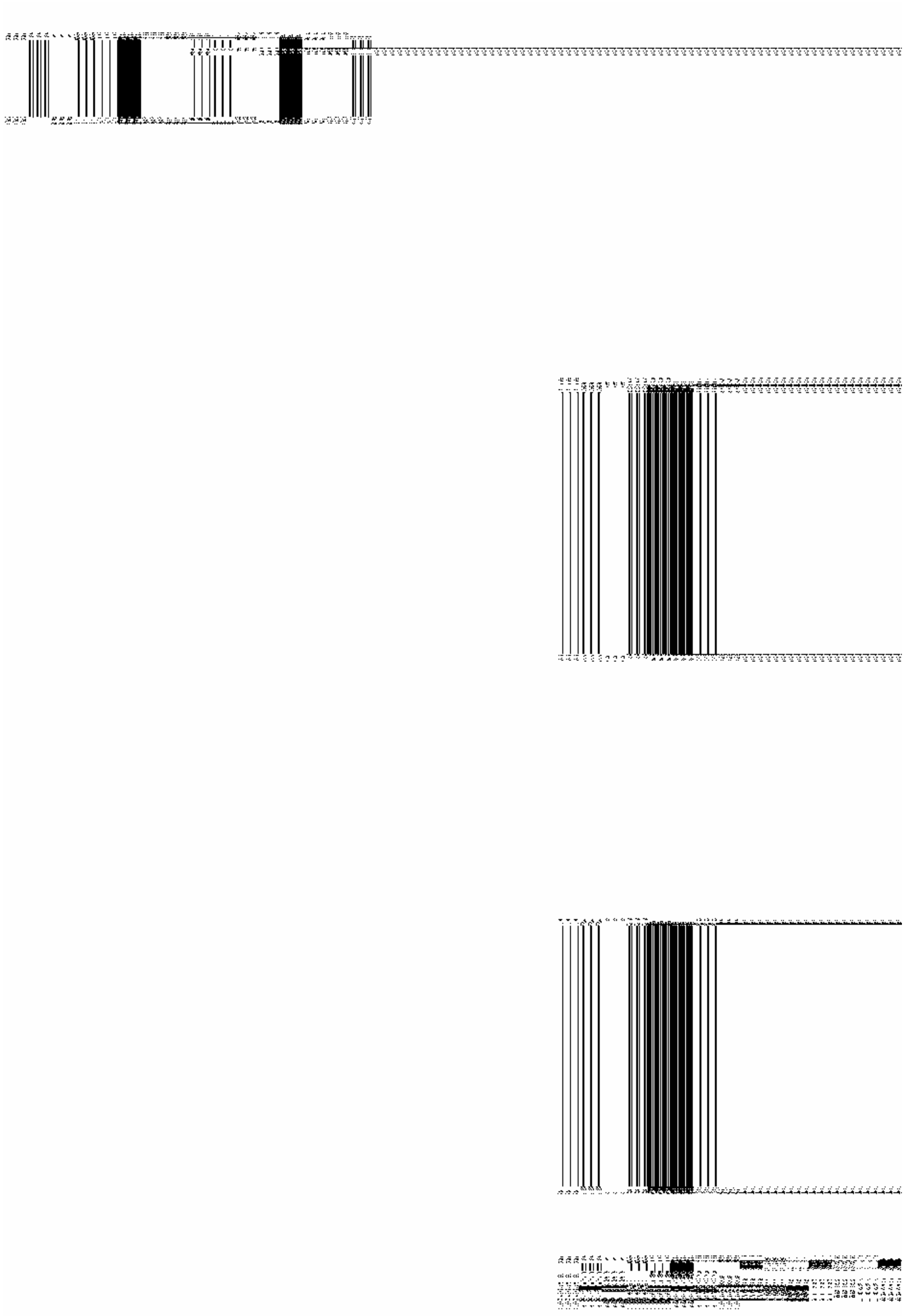


Figure 4.9 Simulation of DITL NAND2 Gate.

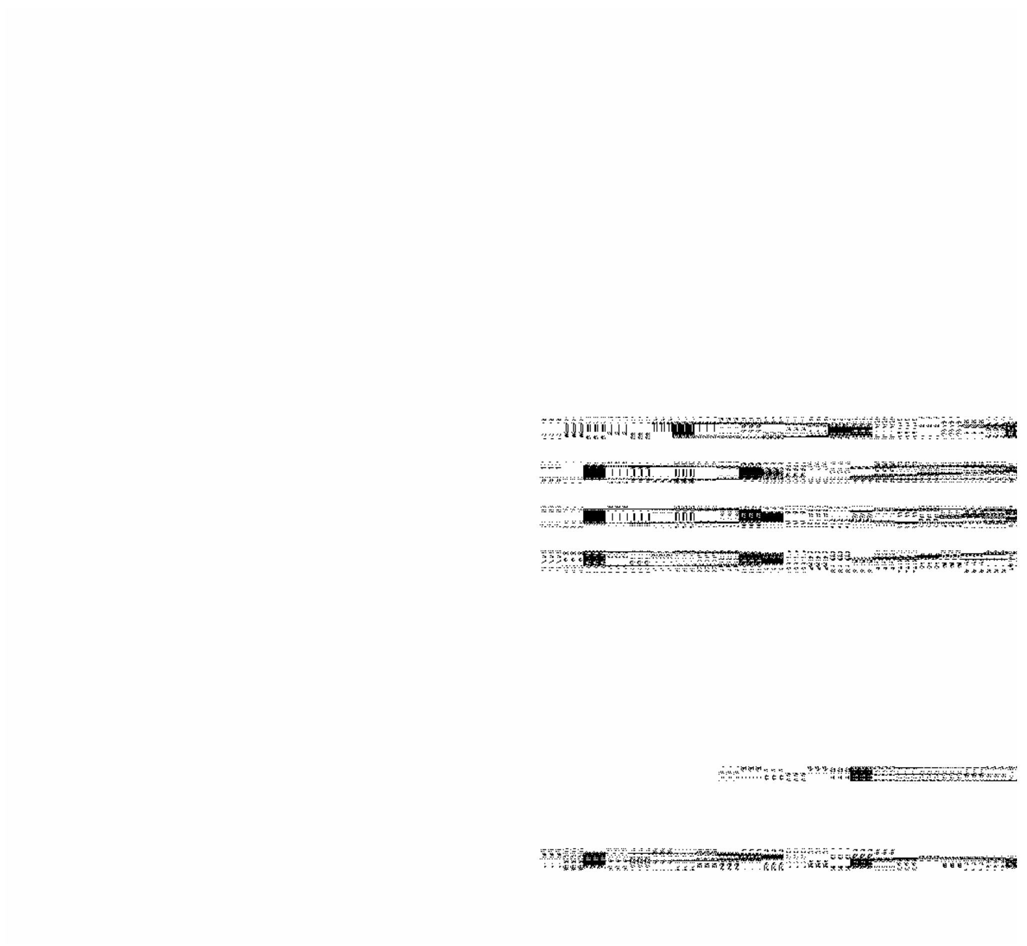


Figure 4.9 Schematic of a DITL NAND4 Gate.

The combinational logic of Stage1 was then augmented by adding a NAND2 gate to combine the outputs of the two NAND4 gates into a single signal, which was then fed into a single output register. This second single-stage design with two combinational logic levels is illustrated in Figure 4.11 (b), and is named *Stage12*. Finally, a 2-bit register was added between the NAND4 gates and the NAND2 gate to form a 2-stage pipelined design, shown above in Figure 4.11 (c), which is named *Stage2*.

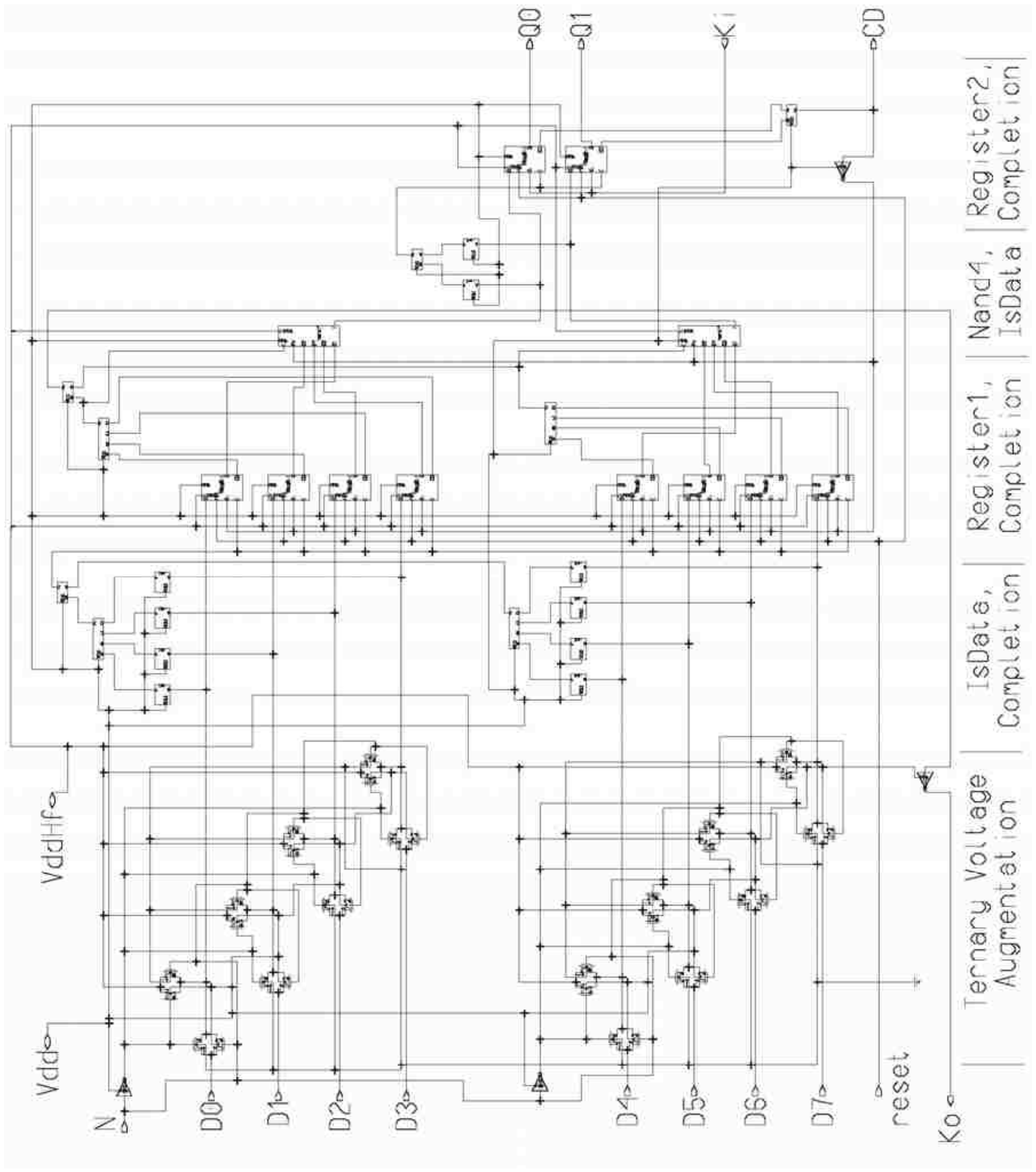


Figure 4.11 Schematic of DITL Systems. (a) Stage1.

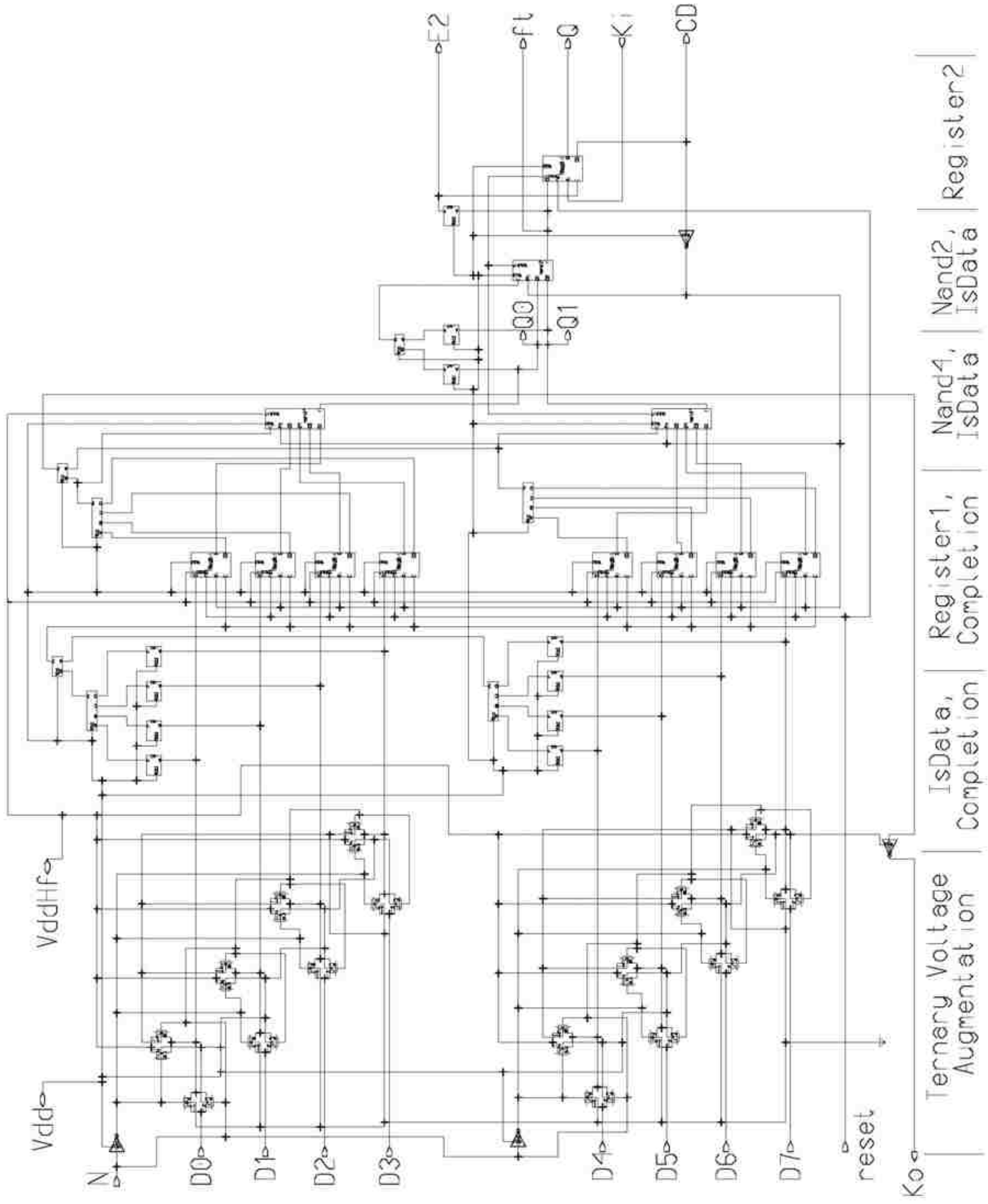


Figure 4.11 Schematic of DITL Systems (cont.). (b) Stage12.

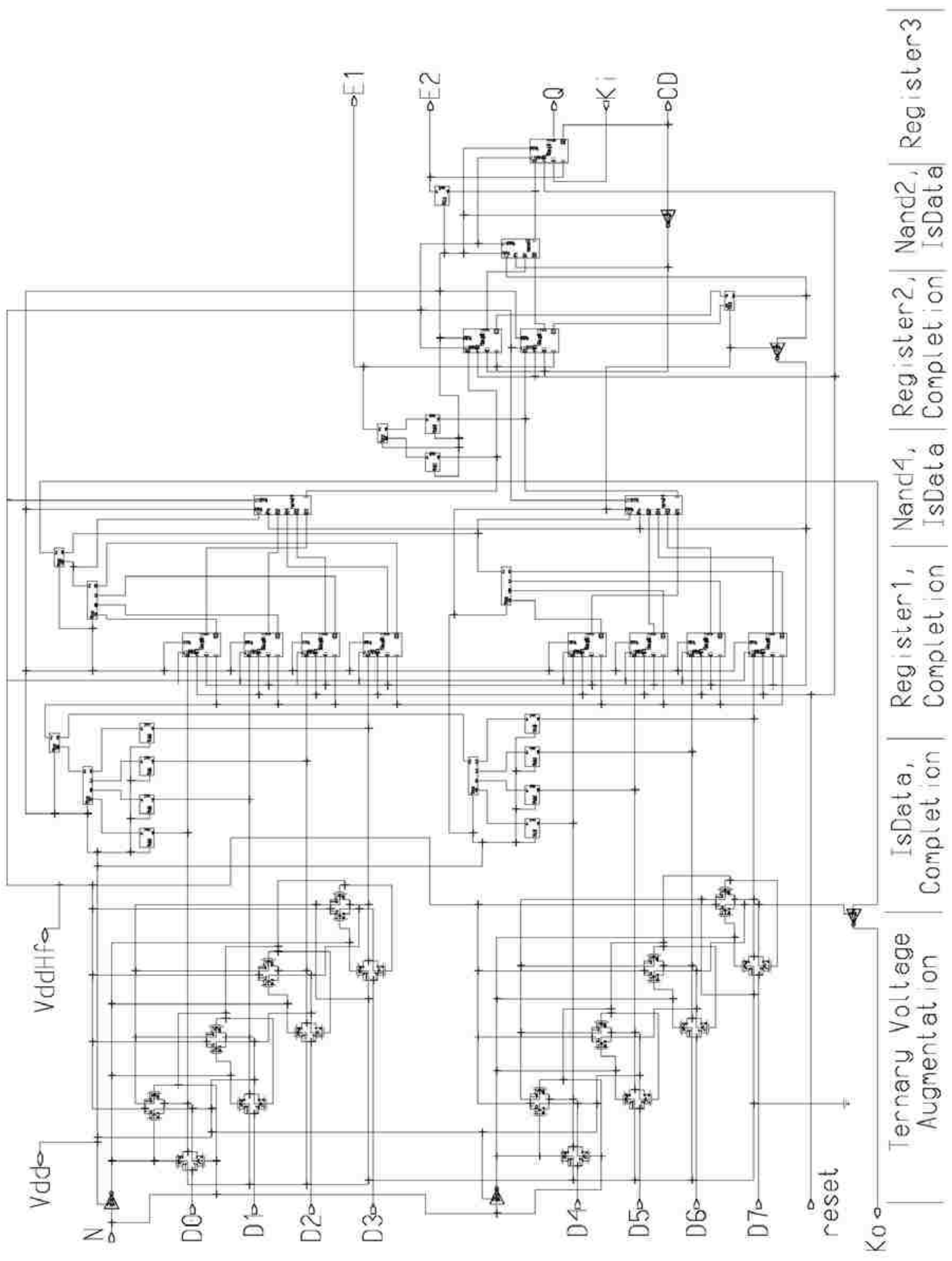


Figure 4.11 Schematic of DITL Systems (cont.). (c) Stage2.

4.3.2. NCL Equivalent Systems. For comparison purposes, the three DITL systems described above were redesigned using the standard dual-rail NCL paradigm [20], as shown in Figure 4.12. The designs use standard dual-rail NCL registers and completion components, shown in Figures 2.3 and 2.4, respectively. The NAND2 function consists of a TH22 and THand0 gate; and the NAND4 function requires sixteen TH44 gates, four TH14 gates, and one TH13 gate.

4.4. SIMULATION RESULTS

The three NCL and DITL systems were simulated using Mentor Graphics' ADvanced Mixed Signal simulator, ADMS, with inputs controlled by a VHDL testbench, as detailed in [21]. Asynchronous circuits require inputs to change based on changes in handshaking outputs; hence, a VHDL testbench can be used to monitor changes in the outputs and change the inputs accordingly, whereas a purely analog simulation does not provide this capability. Another advantage of using ADMS is that total power consumption can be automatically logged and used to calculate Energy per Operation. Five arbitrary DATA-NULL combinations were selected, and these five DATA/NULL wavefronts were input to each of the six circuits being simulated.

4.4.1. DITL Systems. The DITL Stage1 circuit simulation is shown in Figure 4.13. Outputs *Q0* and *Q1* clearly show three distinct voltage levels, and are the correct values corresponding to each of the five input vectors. *Q0* is produced by the NAND of the least significant 4 bits of *D* and *Q1* by the NAND of the most significant 4 bits of *D*. Signal *CD* is observed to determine when both *Q0* and *Q1* become DATA and when both become NULL. *Ki* is changed in the VHDL testbench based on the value of *CD*. Figure 4.13 (b) shows the same diagram with cursors and an additional waveform for Power.

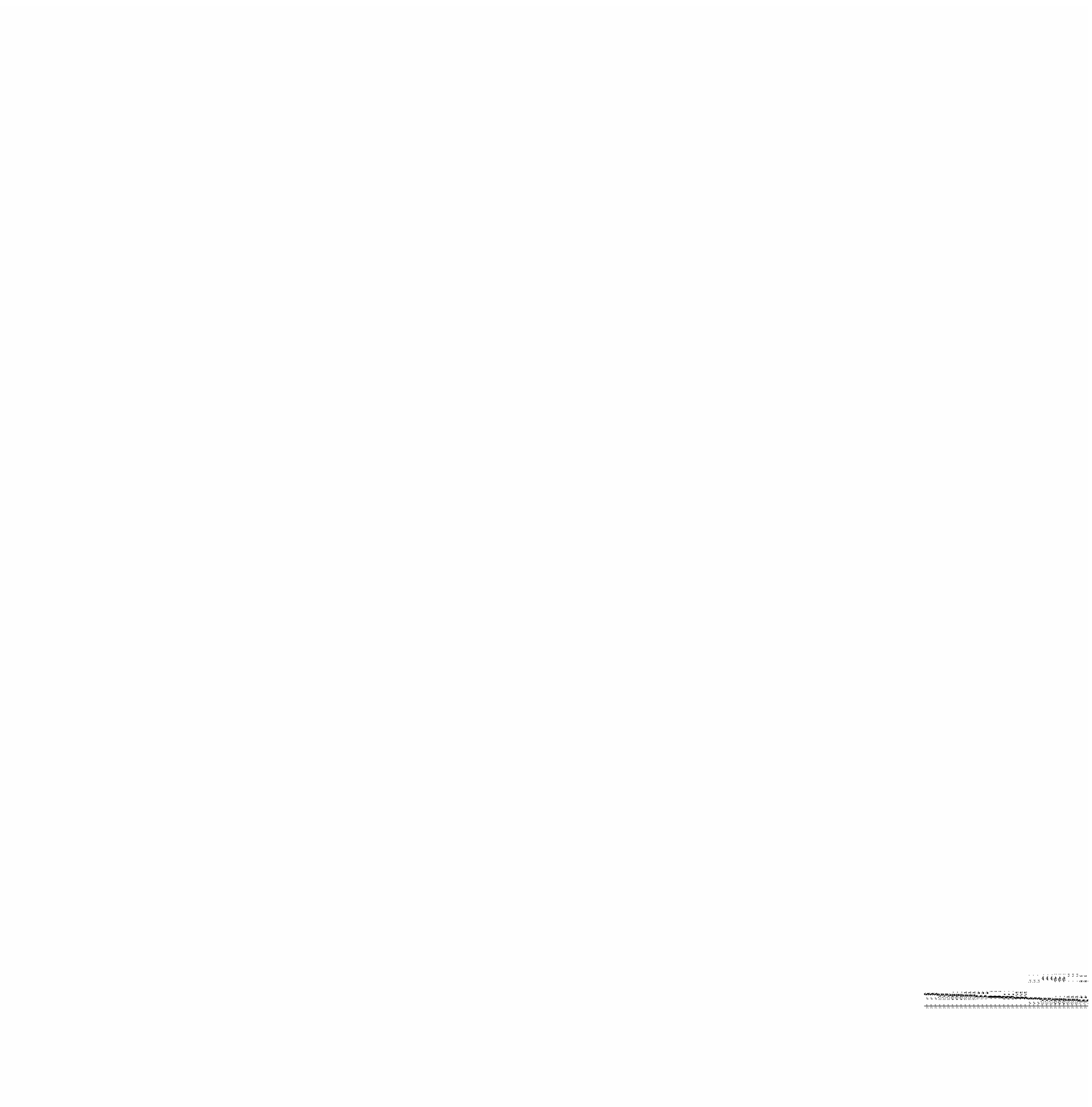


Figure 4.12 Schematic of NCL Equivalent Systems. (a) Stage1.

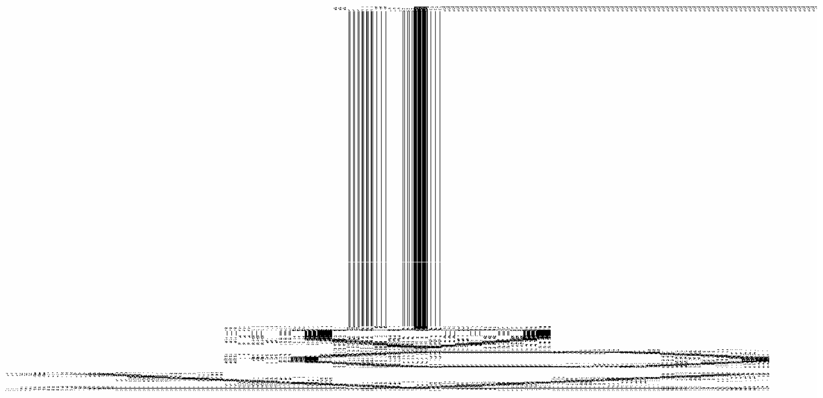


Figure 4.12 Schematic of NCL Equivalent Systems (cont.). (b) Stage12.

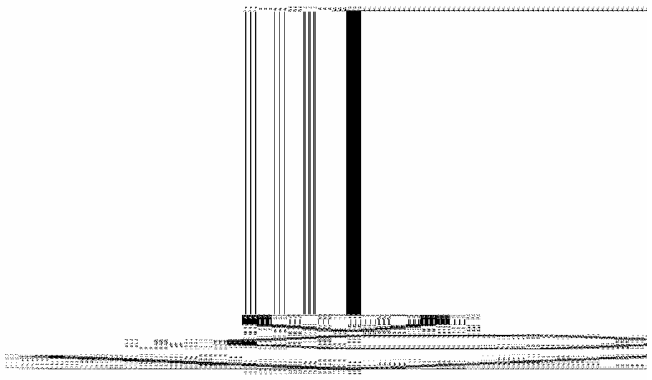


Figure 4.12 Schematic of NCL Equivalent Systems (cont.). (c) Stage2.

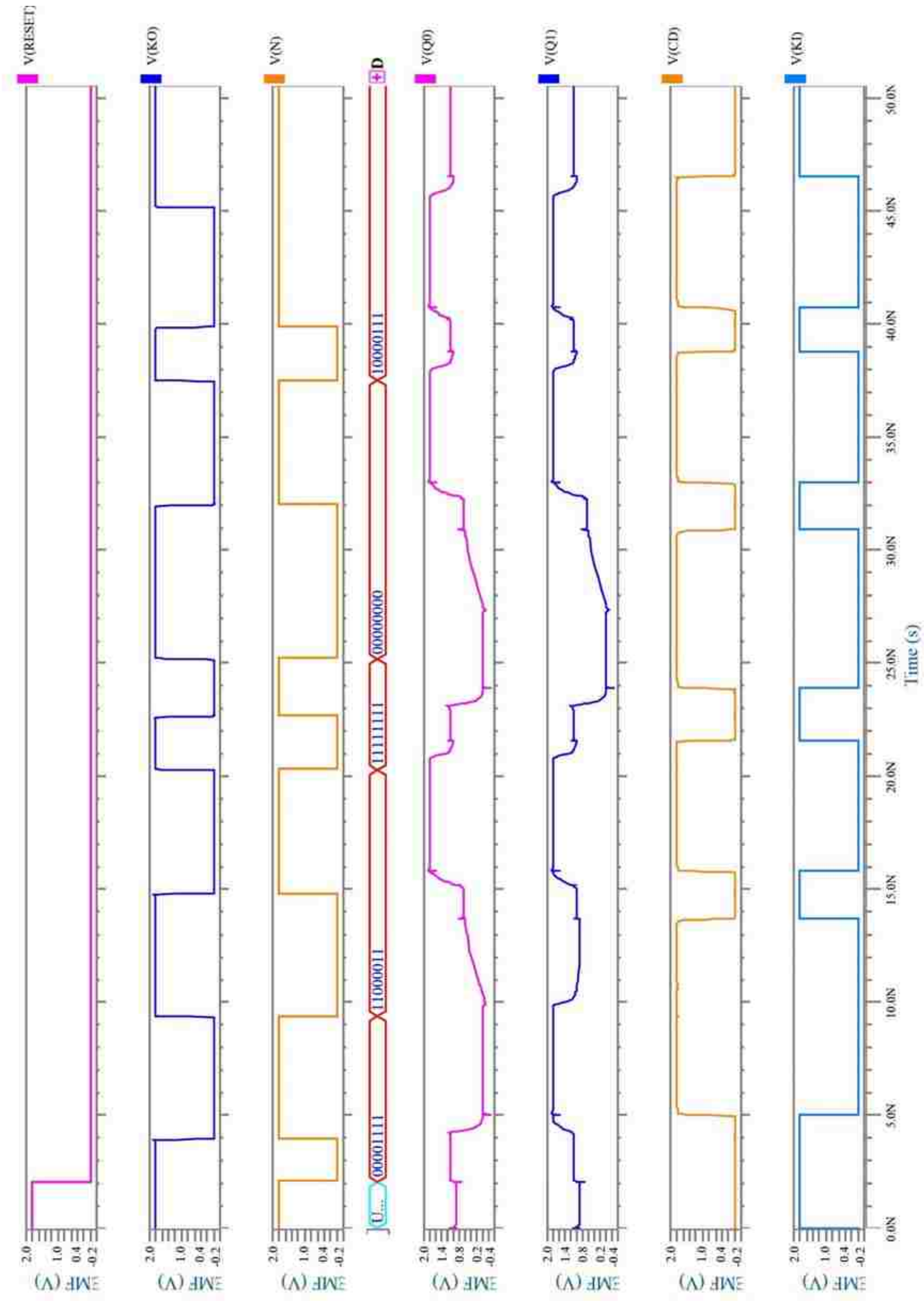


Figure 4.13 Simulation of DITL Stage1. (a) Overview.

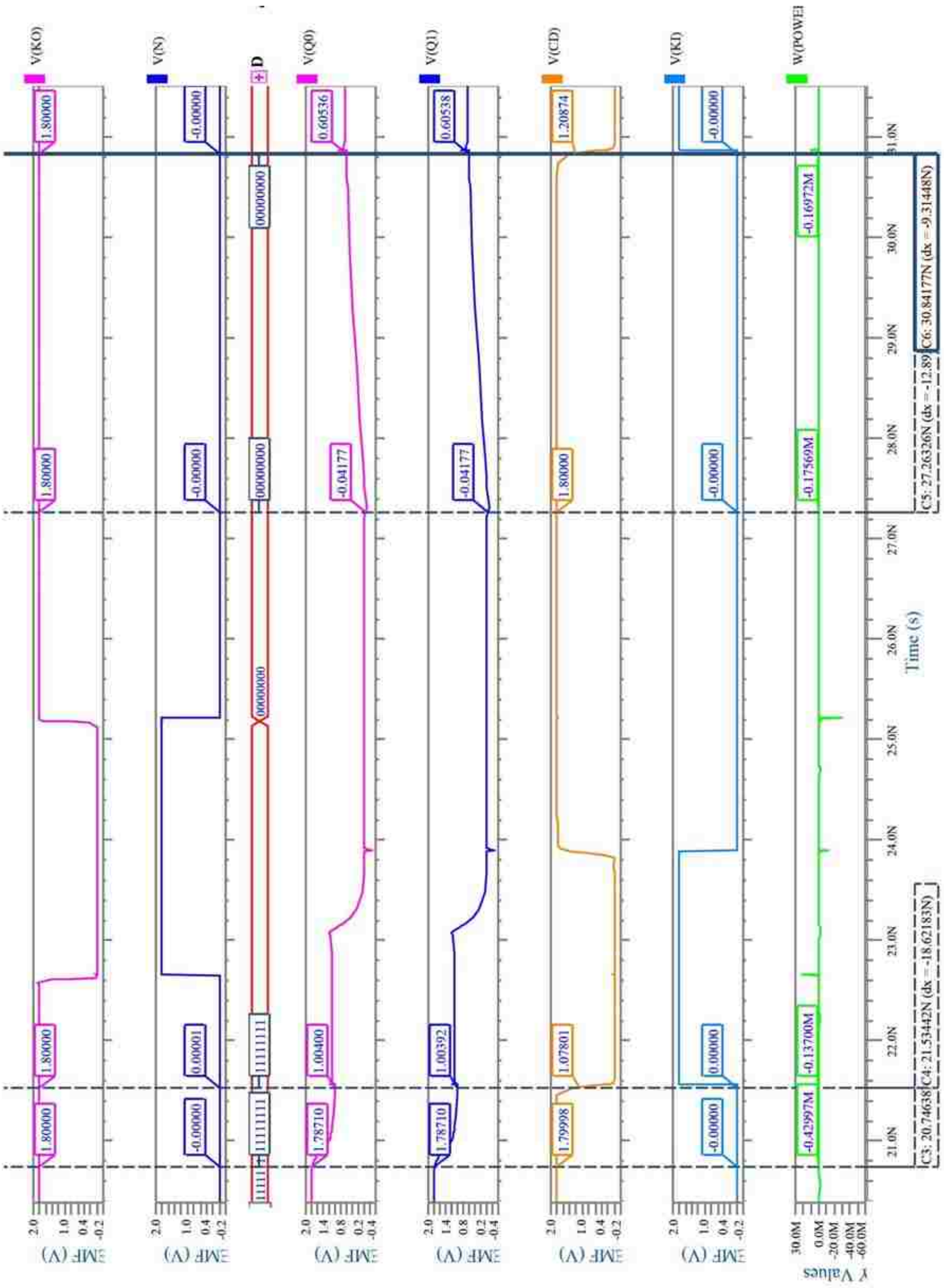


Figure 4.13 Simulation of DITL Stage1 (cont.). (b) Including power waveform and Cursors.

The area under this power curve is calculated and averaged over the 5 operations to obtain Energy per Operation. This figure also shows the range of voltages for the NULL state, from 0.6V to 1.1V; however, this is acceptable, resulting in a properly operating circuit. Charging time from ground to ½ Vdd is the major limiting factor to circuit performance, being approximately one order of magnitude slower than the Vdd to ½ Vdd transition (i.e., 3.8 ns vs. 0.4 ns). However, minimum sized transistors were used in all DITL system components (except for the one larger transistor in every register for resetting); hence, transistor sizing may be able to speedup the circuits and obtain a NULL value closer to the 0.9V optimal, the drawback being additional area and power.

Figure 4.14 shows the simulation waveforms for DITL systems, Stage12 and Stage2. The delay for 5 DATA/NULL wavefronts was measured from the simulations and divided by 5 to calculate average cycle time; and the total energy usage was automatically calculated from ADMS and divided by 5 to obtain average Energy per Operation. Table 4.1 shows the tabulated results of the DITL simulations. Note that the Stage2 design is slower than the Stage12 circuit, which is counterintuitive for a delay-insensitive paradigm, since adding additional registers normally does not slow down the system, it either speeds up or performance remains the same. However, the extra interaction between registers in DITL results in a pipelining condition similar to synchronous systems, where the number of combinational logic delays per stage can only be reduced so far in order to increase performance; and further pipelining can actually decrease performance.

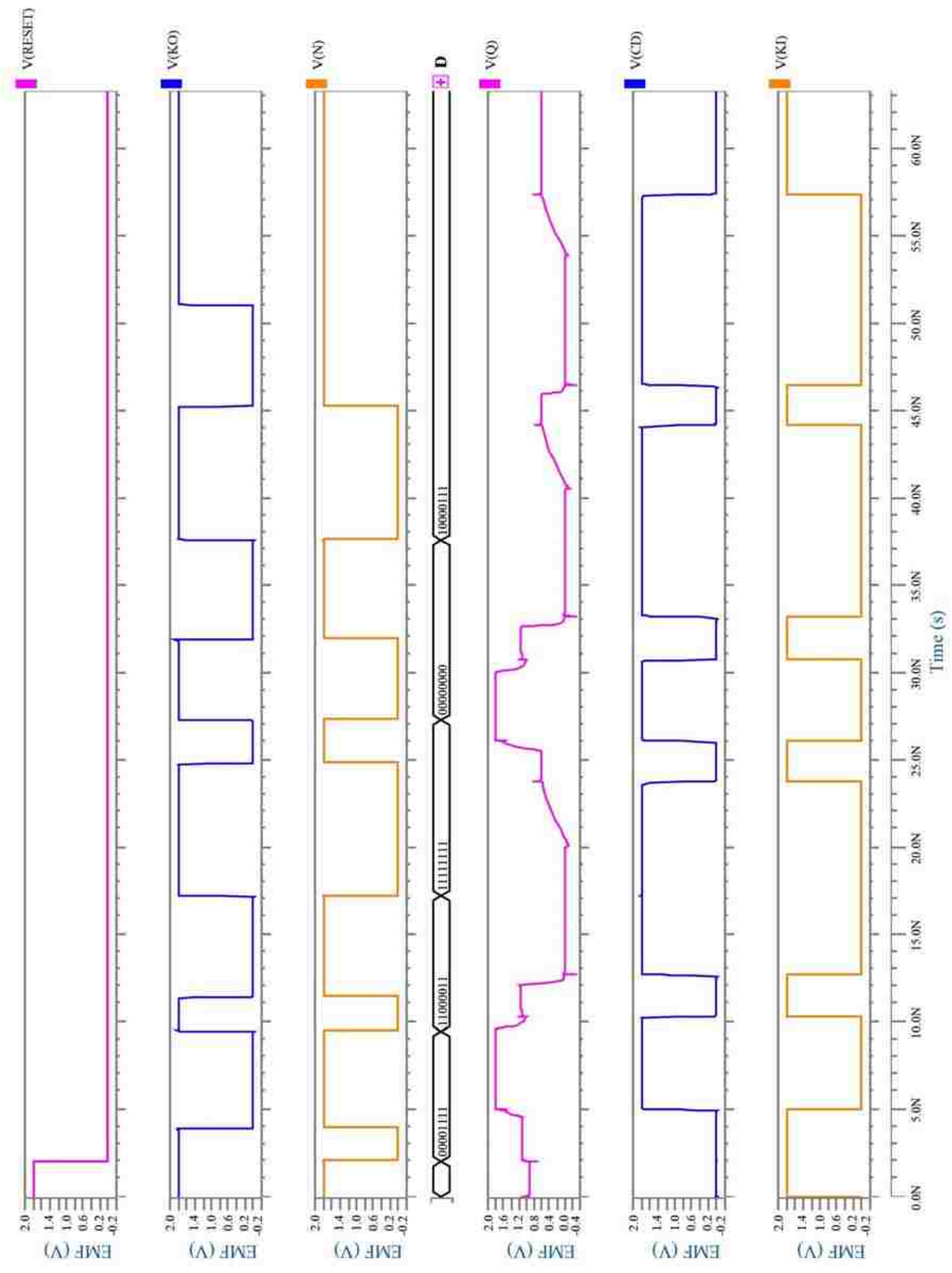


Figure 4.14 Simulation of other DITL Systems. (a) Stage12.

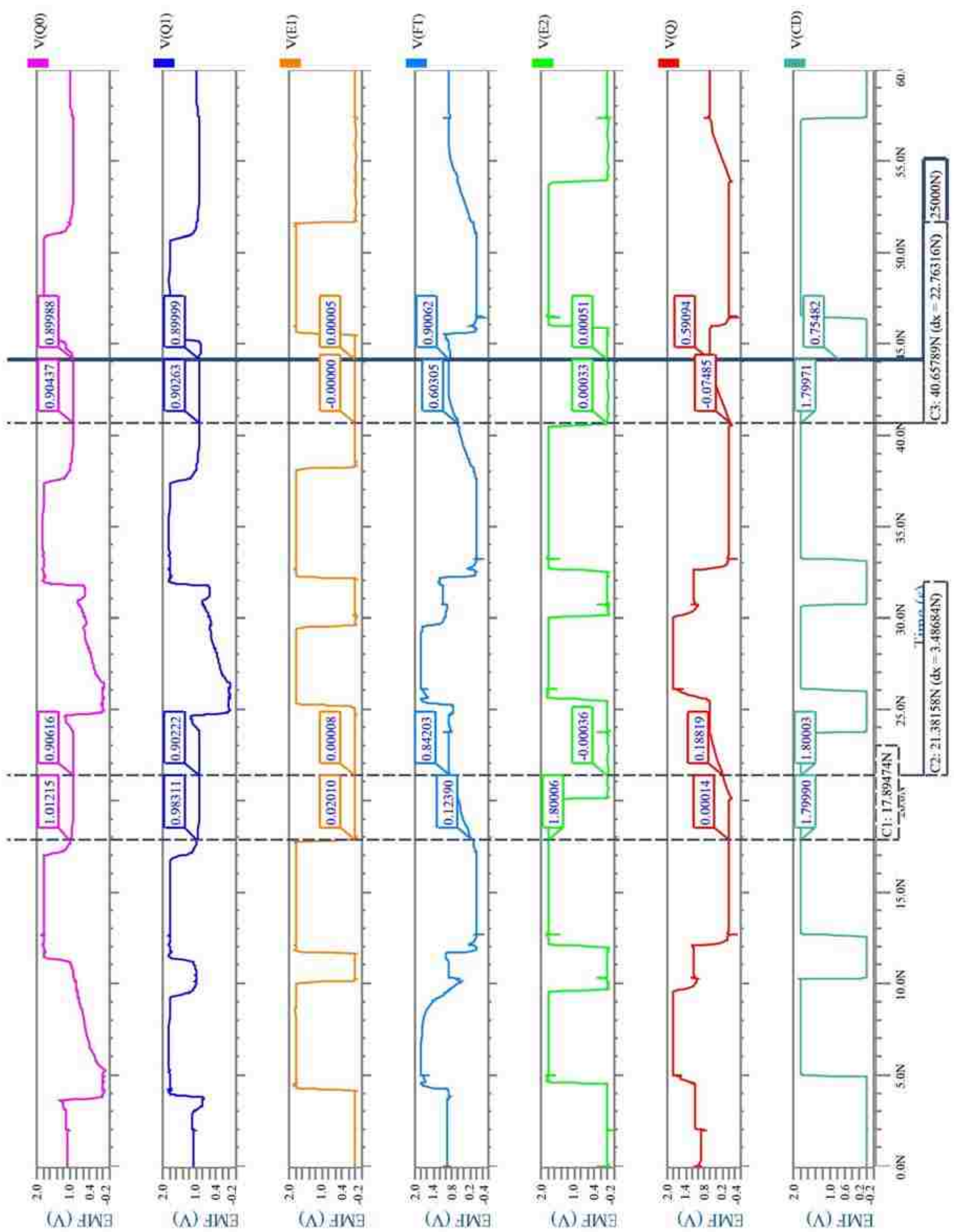


Figure 4.14 Simulation of other DITL Systems (cont.). (b) Stage12 with Internal Signals.

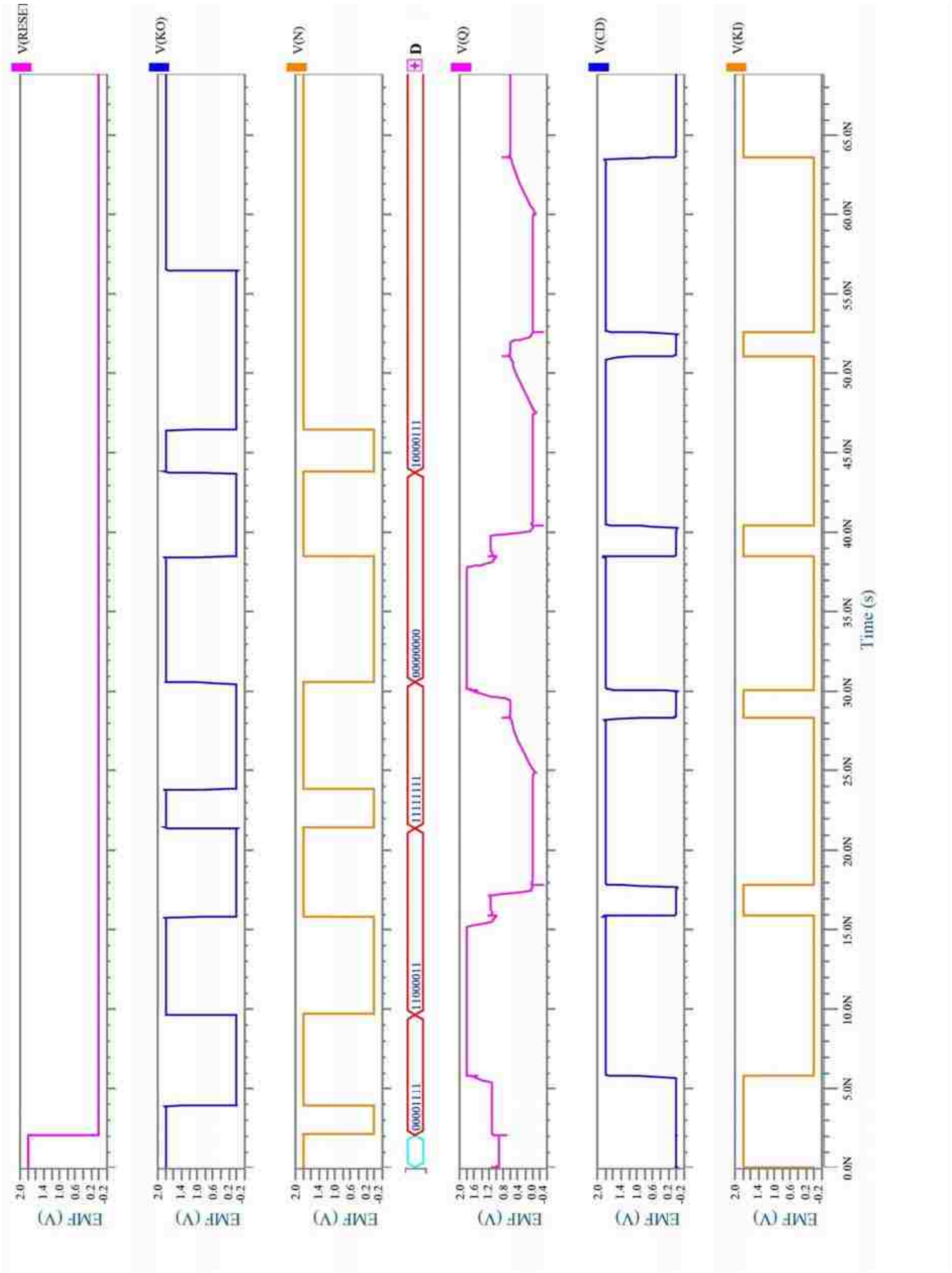


Figure 4.14 Simulation of other DITL Systems (cont.). (c) Stage2.

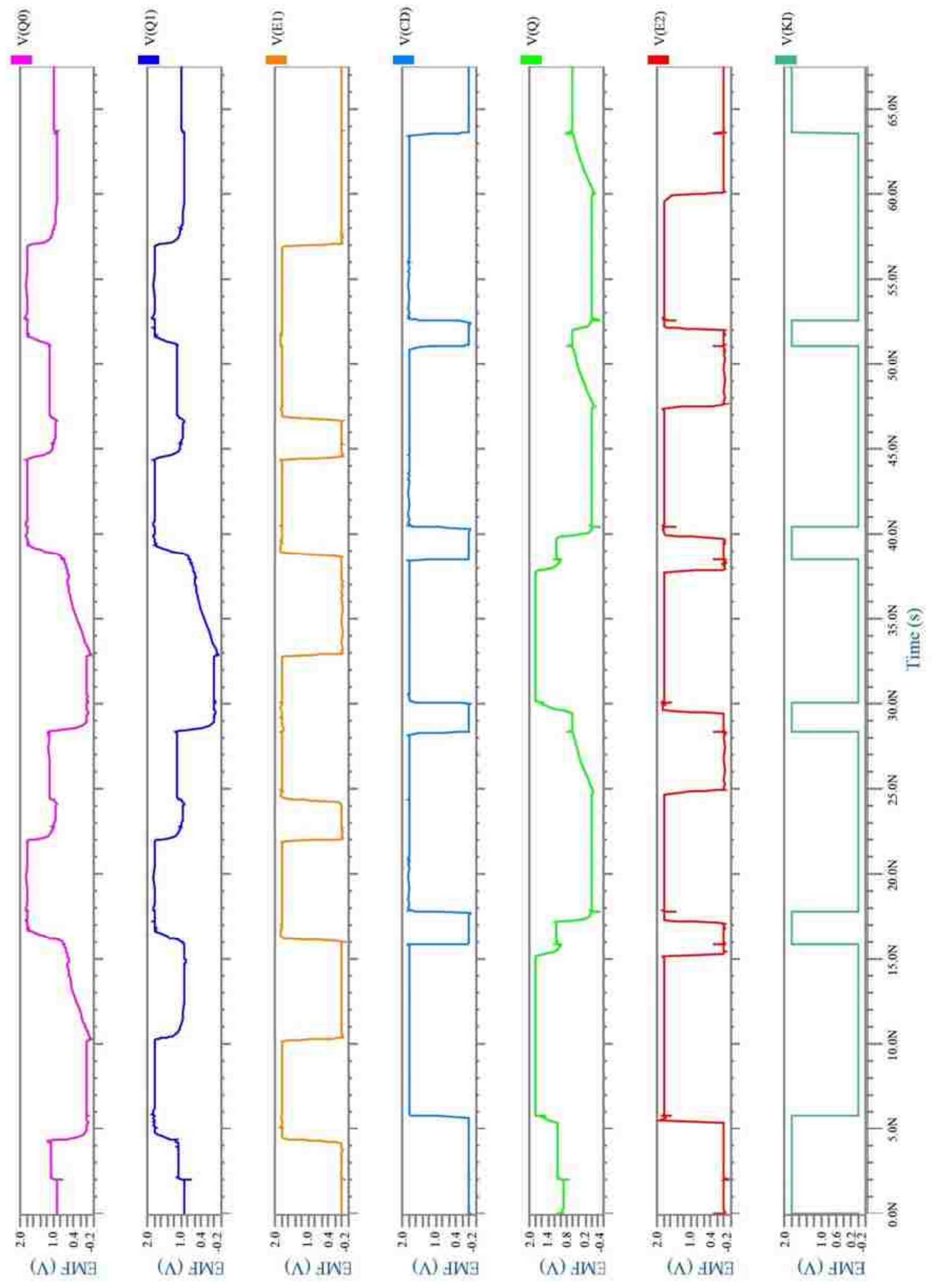


Figure 4.14 Simulation of other DITL Systems (cont.). (d) Stage2 with Internal Signals.

Table 4.1 DITL Simulation Results.

DITL	Stage1	Stage12	Stage2
Number of Transistors	**710 (10 big)**	**685 (9 big)**	**777 (11 big)**
Avg. energy/op (pJ)	**11.3109**	**10.7609**	**12.242**
Avg. Cycle Time (ns)	**8.6**	**11.1**	**12.3**

4.4.2. NCL Systems. The three NCL systems were simulated using the same inputs as the DITL systems, and their simulations are shown in Figure 4.15, and the results tabulated in Table 4.2. Note that only rail1 of the inputs are shown as a bus named *DATA* in order to reduce the diagram size. Also note that the Stage2 design is faster than the Stage12 circuit, as expected.

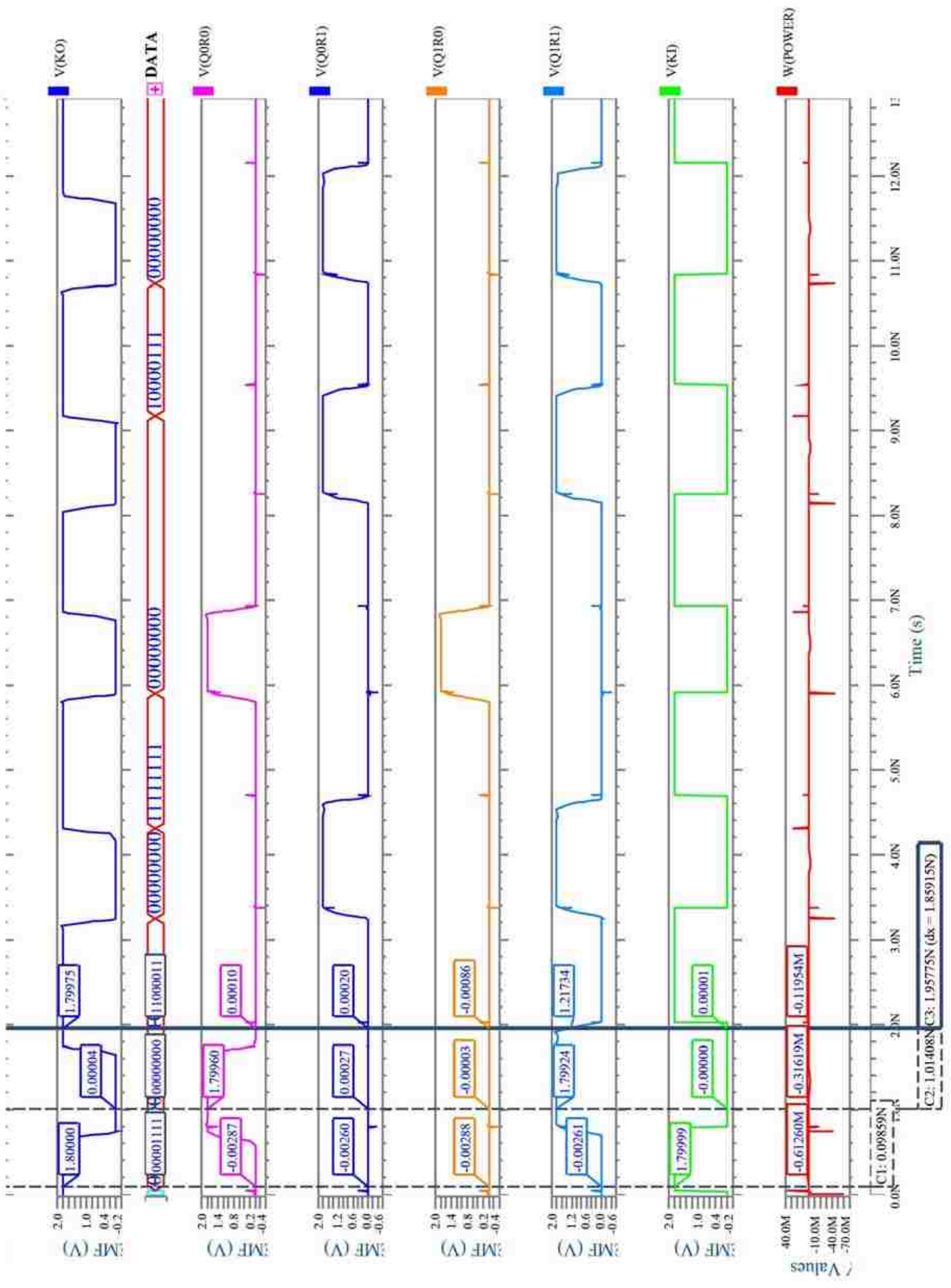


Figure 4.15 Simulation of NCL Equivalent Systems. (a) Stage1.

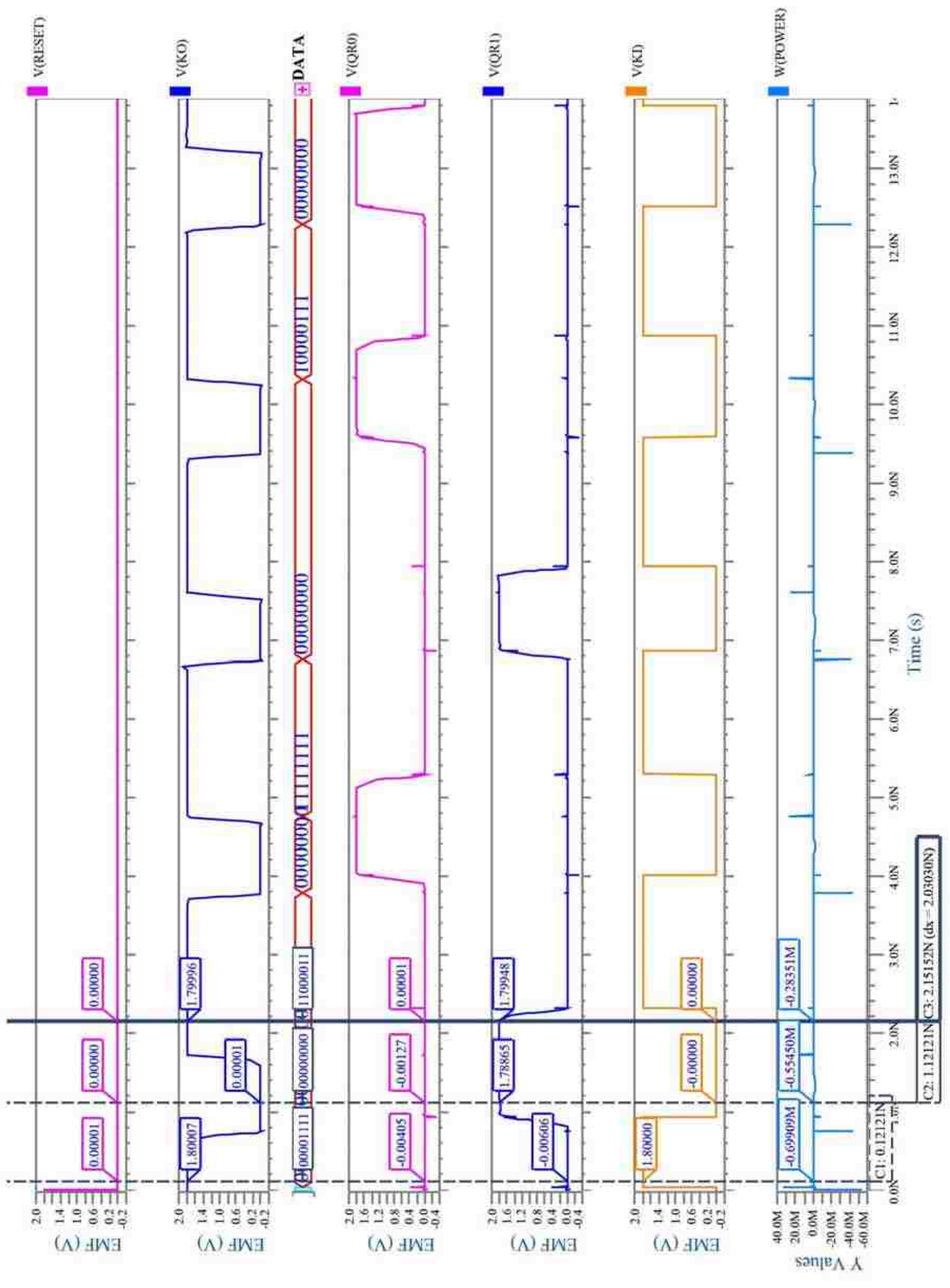


Figure 4.15 Simulation of NCL Equivalent Systems (cont.). (b) Stage12.

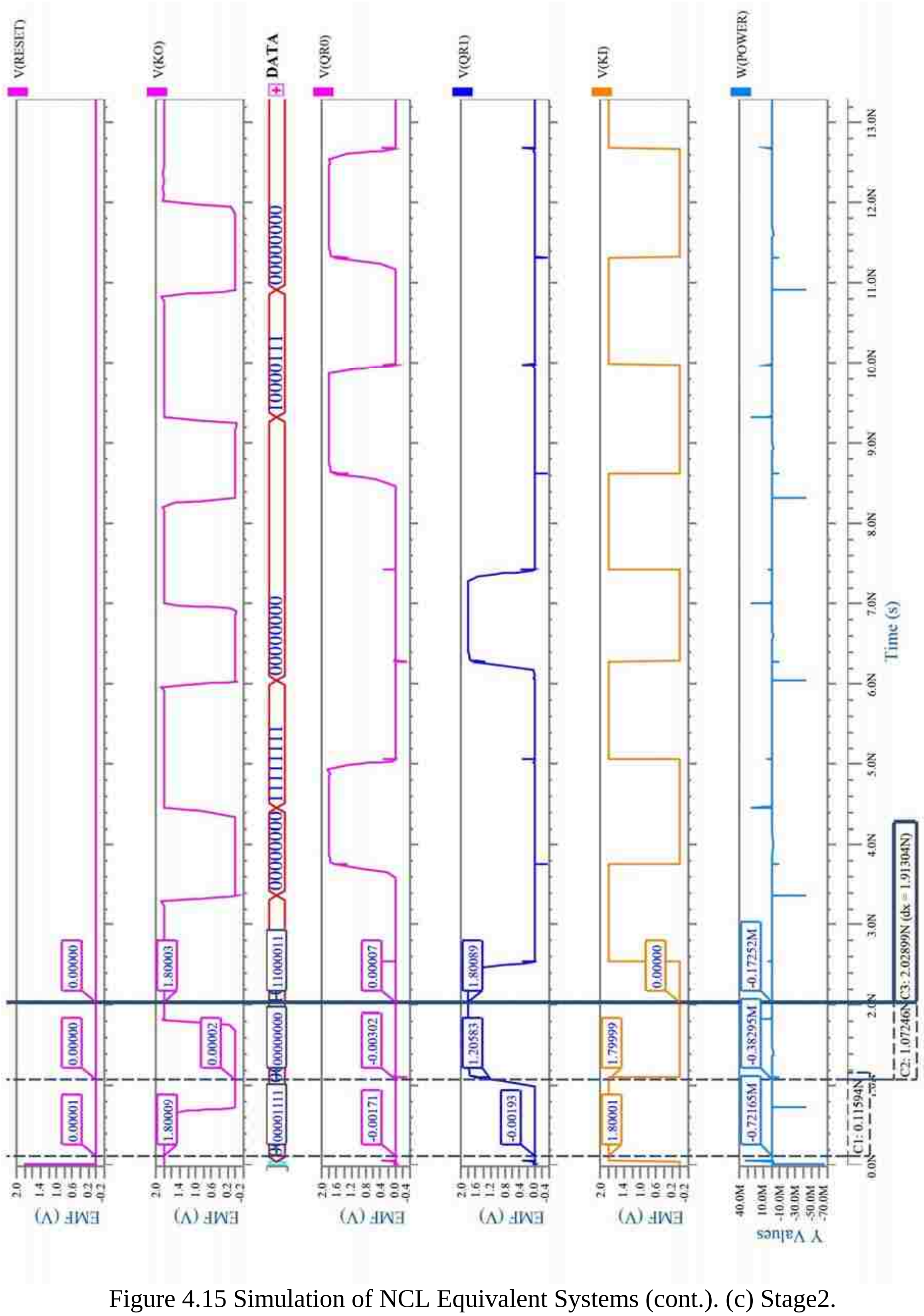


Figure 4.15 Simulation of NCL Equivalent Systems (cont.). (c) Stage2.

Table 4.2 NCL Simulation Results.

NCL	Stage1	Stage12	Stage2
Number of Transistors	**1160**	**1143**	**1227**
Avg. energy/op (pJ)	**6.5728**	**6.5046**	**7.1381**
Avg. Cycle Time (ns)	**2.4**	**2.7**	**2.5**

4.4.3. Comparison. Tables 4.1 and 4.2 show that the standard dual-rail NCL paradigm requires less energy per operation and is faster; however, DITL reduces the number of transistors by approximately 60%. DITL also requires far less interconnect area than NCL, further reducing area.

Even though less energy consumption was expected for DITL due to the theoretical decrease in dynamic power, as explained in Section 1.2, the increase in other power components, such as Static power (refer to Section 4.2.1), annulled the effect of a decrease in Dynamic power, yielding an overall increase in power consumption.

5. CONCLUSIONS AND FUTURE WORK

In this master's thesis, an alternative Single Rail Delay-Insensitive paradigm using ternary logic, which is based on NULL Convention Logic (NCL) and called Delay-Insensitive Ternary Logic (DITL), has been developed. The DITL paradigm has been shown to be fully delay-insensitive and to require substantially less area compared to NCL. However, NCL is better in terms of energy and performance.

Future work includes investigating alternative delay-insensitive paradigms, such as Pre-Charged Half Buffers [22], and redesigning them utilizing ternary logic to possibly reduce energy and increase performance compared to DITL. Additionally, transistor sizing needs to be looked at to see how this affects energy and performance; and the optimal number of combinational logic delays per stage for maximizing performance must be investigated.

BIBLIOGRAPHY

[1] http://www.itrs.net/Links/2003ITRS/Design2003.pdf (available August 2007).

[2] http://www.itrs.net/Links/2005ITRS/Design2005.pdf (available August 2007).

[3] Ivan E. Sutherland, "Micropipelines," *Communications of the ACM*, Vol. 32/6, pp. 720-738, 1989.

[4] 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.

[5] Hong-Yi Huang and Chung-Yu Wu, "Redundant algebra and integrated circuit implementation of ternary logic and their applications," *1993 IEEE International Symposium on Circuits and Systems, ISCAS '93*, pp.1905 – 1908, May 1993.

[6] T. Felicijan and S.B Furber, "An Asynchronous Ternary Logic Signaling system," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 11, Issue 6, pp. 1114 – 1119, Dec. 2003.

[7] C. L. Seitz, "System Timing," in *Introduction to VLSI Systems*, Addison-Wesley, pp. 218-262, 1980.

[8] C. H. (Kees) van Berkel, M. Rem, and R. Saeijs, "VLSI Programming," *1988 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 152-156, 1998.

[9] D. E. Muller, "Asynchronous Logics and Application to Information Processing," in *Switching Theory in Space Technology*, Stanford University Press, pp. 289-297, 1963.

[10] T. Verhoff, "Delay-Insensitive Codes – An Overview," *Distributed Computing*, Vol. 3, pp. 1-8, 1988.

[11] G. E. Sobelman and K. M. Fant, "CMOS Circuit Design of Threshold Gates with Hysteresis," *IEEE International Symposium on Circuits and Systems* (II), pp. 61- 65, 1998.

[12] A. Kondratyev, L. Neukom, O. Roig, A. Taubin, and K. Fant, "Checking Delay-insensitivity: 10^4 Gates and Beyond," *Eighth International Symposium on Asynchronous Circuits and Systems*, pp. 137-145, 2002.

[13] C.L. Connell and P.T. Balsara, “A new ternary MVL based completion detection method for the design of self-timed circuits using dynamic CMOS logic,” *Proceedings of the 2002 45th Midwest Symposium on Circuits and Systems MWSCAS-2002*, Volume 1, pp.I - 503-6 vol.1, Aug. 2002.

[14] C.L. Connell and P.T. Balsara, “A novel single-rail variable encoded completion detection scheme for self-timed circuit design using ternary multiple valued logic,” *Proceedings of the IEEE 2nd Dallas CAS Workshop on Low Power/Low Voltage Mixed-Signal Circuits and Systems, DCAS-01*, pp.P7 – 10, March 2001.

[15] Y. Nagata and M. Mukaidono, “Design of an asynchronous digital system with B-ternary logic,” *Proceedings of the 27th International Symposium on Multiple-Valued Logic,* pp.265 – 271, May 1997.

[16] Y. Nagata, D.M. Miller and M. Mukaidono, “B-ternary logic based asynchronous micropipeline,” *Proceedings of the 29th IEEE International Symposium on Multiple-Valued Logic*, pp.214 – 219, May 1999.

[17] R. Mariani, R. Roncella, R. Saletti and P. Terreni, “On the Realisation of Delay-Insensitive Asynchronous Circuits with CMOS Ternary logic,” *Third International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC '97)*, 1997.

[18] R. Mariani, R. Roncella, R. Saletti and P. Terreni,“A useful application of CMOS ternary logic to the realisation of asynchronous circuits,” *Proceedings of the 27th International Symposium on Multiple-Valued Logic*, pp. 203 – 208, May 1997.

[19] J. L. Huertas and J. M. Carmona, “Low-power Ternary CMOS Circuits,” *IEEE Proceedings of ISMVL*, pp. 170-174, 1979.

[20] S. C. Smith, “Integrating Asynchronous Digital Design into the Undergraduate Computer Engineering Curriculum,” *The 2006 ASEE Midwest Section Annual Conference*, September 2006.

[21] A. Singh and S. C. Smith, “Using a VHDL Testbench for Transistor-Level Simulation and Energy Calculation,” *The 2005 International Conference on Computer Design*, pp. 115-121, June 2005.

[22] A. J. Martin and M. Nystrom, “Asynchronous techniques for system-on-chip design**,”** *Proceedings of the IEEE*, pp. 1089 – 1120, Vol. 94, No. 6, June 2006.

VITA

Ravi Sankar Parameswaran Nair was born on June 15, 1983 in Trivandrum, India. He received the degree of Bachelor of Engineering in Electronics and Communication from S. C. T. College of Engineering (SCTCE), Kerala University, Trivandrum in June 2005. After obtaining his bachelor’s degree, he joined the Master of Science program in Computer Engineering at the University of Missouri-Rolla in January 2006 and graduated in December 2007. His research with Dr. Scott C. Smith in the University of Missouri-Rolla has concentrated in the area of Asynchronous Delay-Insensitive Digital Design using NULL Conventional Logic.