



**GATE-LEVEL COMMERCIAL MICROELECTRONICS VERIFICATION WITH
STANDARD CELL RECOGNITION**

THESIS

Leleia A. Hsia, Second Lieutenant, USAF

AFIT-ENG-MS-15-M-069

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the United States Government.

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENG-MS-15-M-069

GATE-LEVEL COMMERCIAL MICROELECTRONICS VERIFICATION WITH
STANDARD CELL RECOGNITION

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Electrical Engineering

Leleia A. Hsia, B.S.E.E.
Second Lieutenant, USAF

March 2015

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFIT-ENG-MS-15-M-069

GATE-LEVEL COMMERCIAL MICROELECTRONICS VERIFICATION WITH
STANDARD CELL RECOGNITION

THESIS

Leleia A. Hsia, B.S.E.E.
Second Lieutenant, USAF

Committee Membership:

Maj Derrick Langley, PhD
Chair

Mary Y. Lanzerotti, PhD
Member

Kenneth M. Hopkinson, PhD
Member

Richard K. Martin, PhD
Member

Abstract

Within the past two decades, the problem of counterfeit hardware has gained significant attention within the Department of Defense (DoD). Counterfeit electronics compromise national security systems as they may fail to meet durability requirements and/or contain malicious circuits [6, 16, 17]. This necessitates the development of methods to detect counterfeit electronics and prevent the counterfeit electronics from entering DoD systems. The DARPA TRUST program was established to address the need to verify integrated circuit (IC) electronics. This research describes the development of standard cell recognition (SCR) software intended to resolve conflicts in prior TRUST related applications of commercial software to verify IC designs. SCR software applications to circuits composed of up to 650 transistors are presented, and the resulting 90% SCR application success rate is discussed.

Acknowledgments

Sincere appreciation is given to Maj Derrick Langley for serving as advisor and committee chair, and to Dr. Mary Lanzerotti for continued guidance as an advisor. I also thank Dr. Ken Hopkinson and Dr. Rick Martin for serving as committee members on this research, and Dr. Brian Dupaix, Dr. Michael Myers, Mr. Todd James, Mr. Len Orlando and Mr. Bradley Paul at the AFRL MSDC for their research contributions. Additionally, I would like to acknowledge and thank Mr. Bradley Paul at the AFRL MSDC for sponsoring this research.

Leleia A. Hsia

Table of Contents

	Page
Abstract	iv
Acknowledgments	v
Table of Contents	vi
List of Figures	ix
List of Tables	xiv
List of Acronyms	xvi
 I. Introduction	 1
1.1 Motivation	1
1.2 Avoidance	3
1.3 Detection	3
1.4 Detection Methods	4
1.5 Proposed Methodology	4
1.6 Assumptions and Scope	5
1.7 Materials and Equipment	5
 II. Background	 6
2.1 Integrated Circuit Verification and Detection Methods	6
2.1.1 Functional Testing	6
2.1.2 Physically Unclonable Functions	9
2.1.3 Shortcomings of Functional Testing and Physically Unclonable Function Implementations	10
2.1.3.1 Transistor- and Gate-Level Testing	14
2.2 Summary of Recent Work	15
2.2.1 Standard Cell Recognition Efforts by Wonjong Kim and Hyunchul Shin of Hanyang University	20
2.2.1.1 Merge Series Transistors	20
2.2.1.2 Find Simple Gates	20
2.2.1.3 Match Hierarchical Subcircuits	21
2.2.1.4 Algorithm Insufficiencies as Applied to DARPA's Cir- cuit Verification Efforts	22

	Page
III. Methodology	23
3.1 Phase 1 Methodology - Gaining Familiarity with Software Tools	24
3.2 Phase 2 Methodology - Software Tool Application to Elementary Gates . . .	25
3.3 Phase 3 Methodology - Software Tool Application to Circuit D Equivalent	27
3.4 Phase 4 Methodology - SCR Code Analysis	29
IV. Results	31
4.1 SCR Research Results	31
4.1.1 Phase 1 Results - Gaining Familiarity with Software Tools	31
4.1.1.1 Cadence Virtuoso Implementation	31
4.1.1.2 Idle IDE Implementation	36
4.1.2 Phase 2 Results - Software Tool Application to Elementary Gates .	38
4.1.2.1 NAND2 Gate	38
4.1.2.2 NOR2 Gate	41
4.1.2.3 AND2 Gate	44
4.1.2.4 OR2 Gate	46
4.1.2.5 NAND2b0 Gate	48
4.1.2.6 OAI21 Gate	50
4.1.2.7 OAI21b1 Gate	52
4.1.2.8 OAI21b0b1 Gate	54
4.1.3 Phase 3 Results - Software Tool Application to Circuit D Equivalent	56
4.1.4 Phase 4 Results - SCR Code Analysis	58
4.1.4.1 Level of Maturity	58
4.2 Discussion of SCR Algorithm and Code	91
4.2.1 Explication of SCR Algorithm	91
4.2.1.1 Class Definitions	91
4.2.1.2 Functions	93
4.2.2 Algorithm Attributes	118
4.3 Advantages of Transistor-level Verification with SCR over Functional Testing	121
4.3.1 Malicious Change in Gate Composition	121
4.3.2 Malicious Switch in Gate Input Signals	132
V. Conclusion and Future Work	136
5.1 Summary	136
5.2 Future Work	136
5.3 Conclusion	139
Appendix A: NCSU Digital Parts Standard Cell Library	140

	Page
Appendix B: Cells included in TRUST Test Articles	143
Appendix C: SCR Code	170
Appendix D: Netlists	257
Bibliography	326

List of Figures

Figure	Page
1 MDA data analysis of performance grade of suspect parts (Adapted from [7])	2
2 Functional test of the 64-bit adder [9]	7
3 A standard IC with malicious circuits (Circuits 1 and 2) added [10]	8
4 Physically unclonable functions [10]	9
5 Full adder circuit with PUF and malicious extraneous component	11
6 Simulated inputs and outputs of the full adder circuit with PUF and malicious extraneous component	12
7 Full adder circuit with PUF only	12
8 Simulated inputs and outputs of the full adder circuit with PUF only	13
9 Forward and reverse IC design flows [19]	15
10 Limited scope forward and reverse IC design flows	16
11 Circuit A conceptual process [19]	17
12 Part of Circuit A schematic before corrections [19]	18
13 Part of Circuit A schematic after corrections [19]	19
14 Abstraction levels of various gates	26
15 A schematic of the complex Circuit D [Adapted from [19]].	27
16 A schematic of the complex Circuit D, mid-level abstraction (Image courtesy of M. Seery).	28
17 A schematic of the complex Circuit D, low-level abstraction.	28
18 A flat schematic of the complex Circuit D.	29
19 Flat inverter circuit schematic designed with Cadence Virtuoso	32
20 Inverter symbol designed with Cadence Virtuoso	33
21 Gate-level inverter circuit designed with Cadence Virtuoso	33

Figure	Page
22 Flat inverter circuit netlist generated by Cadence Virtuoso	34
23 Gate-level inverter circuit netlist generated by Cadence Virtuoso	35
24 Transistor-level schematic of a NAND2 gate	39
25 Gate-level schematic of a NAND2 gate	40
26 Transistor-level schematic of a NOR2 gate	42
27 Gate-level schematic of a NOR2 gate	43
28 Transistor-level schematic of a AND2 gate	44
29 Gate-level schematic of a AND2 gate	45
30 Transistor-level schematic of an OR2 gate	46
31 Gate-level schematic of a OR2 gate	47
32 Transistor-level schematic of a NAND2b0 gate	48
33 Gate-level schematic of a NAND2b0 gate	49
34 Transistor-level schematic of an OAI21 gate	50
35 Gate-level schematic of an OAI21 gate	51
36 Transistor-level schematic of an OAI21b1 gate	53
37 Gate-level schematic of an OAI21b1 gate	53
38 Transistor-level schematic of an OAI21b0b1 gate	55
39 Gate-level schematic of an OAI21b0b1 gate	55
40 Transistor-level schematic of the XOR circuit tested	59
41 Gate-level representation of the XOR circuit tested	59
42 Transistor-level schematic of the Master/Slave DFF circuit tested	61
43 Gate-level representation of the Master/Slave DFF circuit tested	61
44 Transistor-level schematic of the 2-to-1 MUX circuit tested	62
45 Gate-level representation of the 2-to-1 MUX circuit tested	63
46 Transistor-level schematic of the Digital Comparator circuit tested	64

Figure	Page
47 Gate-level representation of the Digital Comparator circuit tested	65
48 Transistor-level schematic of the 4-bit ripple carry adder circuit tested	67
49 Gate-level representation of the 4-bit ripple carry adder adder circuit tested . . .	68
50 Transistor-level schematic of the Test1 circuit tested	71
51 Gate-level representation of the Test1 circuit tested	71
52 Transistor-level schematic of the Test2 circuit tested	73
53 Gate-level representation of the Test2 circuit tested	74
54 Transistor-level schematic of the Test3 circuit tested	76
55 Gate-level representation of the Test3 circuit tested	77
56 Transistor-level schematic of the Test4 circuit tested	79
57 Gate-level representation of the Test4 circuit tested	80
58 Percent comprehensiveness as a function of number of cells in circuit	90
59 Main components of SCR code	91
60 Organization of objects created by class definitions in the SCR code	93
61 The seven steps of the SCR algorithm	94
62 Flowchart of the SCR function in the top level of hierarchy	95
63 Flowchart of the Find_tx function in the second level of hierarchy	98
64 Flowchart of the Find_cells function in the second level of hierarchy	99
65 Flowchart of the Find_cell_types function in the second level of hierarchy . . .	100
66 Process flow of the replace_cells function in the second level of hierarchy . . .	101
67 Flowchart of the Create_NMOS_objects function in the third level of hierarchy .	103
68 Flowchart of the Create_PMOS_objects function in the third level of hierarchy .	104
69 Flowchart of the Find_OR2s function in the third level of hierarchy	105
70 Flowchart for the Find_OR2_types function	107
71 Flowchart for the Find/assign OR2 types subroutine used in Figure 70	108

Figure	Page
72	Flowchart for the function Remove_cell_transistors in the third level of hierarchy 112
73	Flowchart for the function Add_cells_to_components in the third level of hierarchy 113
74	Flowchart for the function Write_cells_to_netlist in the third level of hierarchy . 114
75	Flowchart for the function Remove_OR2_transistors_from_components in the fourth level of hierarchy 115
76	Flowchart for the function Add_OR2s_to_components in the fourth level of hierarchy 116
77	Flowchart for the function Write_cells_to_netlist in the fourth level of hierarchy 117
78	Full adder cell. Compositions of gates at the fourth level of abstraction are shown in boxes 120
79	Gate-level representation of the unmodified 4-bit ripple carry adder circuit . . . 122
80	Transistor-level schematic of the unmodified 4-bit ripple carry adder circuit . . 123
81	Gate-level representation of the altered (changed OAI21 gate composition in Full Adder Cell 0) 4-bit ripple carry adder circuit 124
82	Transistor-level schematic of the altered (changed OAI21 gate composition in Full Adder Cell 0) 4-bit ripple carry adder circuit 125
83	Altered full adder cell in the 4-bit ripple carry adder circuit. Composition of the maliciously altered OAI21 gate at the third level of abstraction is shown in the red box 125
84	Unaltered full adder cell in the 4-bit ripple carry adder circuit 128
85	Altered full adder cell in the 4-bit ripple carry adder circuit. Composition of the maliciously altered OAI21 gate at the third level of abstraction is shown in the red box 129
86	Simulation results of the unmodified 4-bit ripple carry adder 130

Figure	Page
87	Simulation results of the modified (changed OAI21 gate composition in Full Adder Cell 0) 4-bit ripple carry adder 131
88	Gate-level representation of the unmodified 4-bit ripple carry adder circuit . . . 132
89	Gate-level representation of the maliciously modified (switched OAI21 gate inputs in Full Adder Cell 0) 4-bit ripple carry adder circuit 133
90	Maliciously modified (switched OAI21 gate inputs in Full Adder Cell 0) 4-bit ripple carry adder circuit. Compositions of gates at the fourth level of abstraction are shown in boxes 133
91	Transistor-level schematic of the maliciously modified (switched OAI21 gate inputs in Full Adder Cell 0) 4-bit ripple carry adder circuit 134

List of Tables

Table	Page
1 Negative Effects of Counterfeit Electronics on the Government [Adapted from [6]]	2
2 Gates to be identified by SCR algorithm	26
3 Name, Description, and Passed Objects of Stage I Python Standard Cell Recognition (SCR) Functions	37
4 Full adder test results	57
5 XOR gate test results	60
6 Master/Slave DFF test results	60
7 2-to-1 MUX gate test results	63
8 Digital comparator test results	65
9 4-bit ripple carry adder test results	69
10 Test1 test results	72
11 Test2 test results	75
12 Test3 test results	78
13 Test4 test results	80
14 Inverter results	82
15 NAND2 results	83
16 NOR2 results	83
17 AND2 results	84
18 OR2 results	84
19 NAND2b0 results	85
20 OAI21 results	86
21 OAI21b1 results	86

Table	Page
22 OAI21b0b1 results	87
23 Algorithm comprehensiveness for TRUST test circuits	89
24 Truth table of the unmodified Full Adder Cell 0	126
25 Truth table of the maliciously modified Full Adder Cell 0	127
26 Cells included in the first TRUST test article	143
27 Cells included in the second TRUST test article	151
28 Cells included in the third TRUST test article	158
29 Cells included in the fourth TRUST test article	162
30 Cells included in the fifth TRUST test article	169

List of Acronyms

Acronym	Definition
AES	Advanced Encryption Standard
AFRL	Air Force Research Laboratory
AFIT	Air Force Institute of Technology
AMI	American Microsystems, Inc.
DARPA	Defense Advanced Research Projects Agency
DC	Direct Current
DoD	Department of Defense
FPR	False Positive Rate
I2C	Inter-Integrated Circuit
IC	Integrated Circuit
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
IRIS	Integrity and Reliability of Integrated Circuits
MDA	Missile Defense Agency
MEMS	Micro-Electro-Mechanical Systems
MOSIS	Metal Oxide Semiconductor Implementation Service
MSDC	Mixed Signal Design Center
NCSU	North Carolina State University
PUF	Physically Unclonable Function
RTL	Register Transfer Language
SCR	Standard Cell Recognition
SETA	Systems Engineering and Technical Assistance

Acronym	Definition
SRI	Stanford Research Institute
TPR	True Positive Rate
TRUST	Trusted Integrated Circuits
VHDL	Very-High-Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integration
WPAFB	Wright Patterson Air Force Base

GATE-LEVEL COMMERCIAL MICROELECTRONICS VERIFICATION WITH STANDARD CELL RECOGNITION

I. Introduction

1.1 Motivation

WITHIN the past two decades, the problem of counterfeit software and hardware has gained significant attention commercially and within the Department of Defense (DoD). The attention was initially drawn to the prevention of counterfeit software, and has expanded to address the growing problem of counterfeit hardware electronics and integrated circuits (ICs) [5, 13]. In 2006, an Institute of Electrical and Electronics Engineers (IEEE) *Spectrum* article was published to discuss the rise in counterfeit electronics, and in 2008, a *Businessweek* article addressed how counterfeit electronics have compromised military systems and generated unrest within the Pentagon [11, 16]. Specifically, the *Businessweek* article states:

“The American military faces a growing threat of potentially fatal equipment failure and even foreign espionage because of counterfeit computer components used in warplanes, ships and communication networks.” [11]

The dangers mentioned in this article excerpt, however, are not an exhaustive list. Other effects caused by the threat of counterfeit electronics to the government are summarized in Table 1.

Referencing the third effect from Table 1, counterfeit electronics in the supply chains for national security systems pose a problem not only because these circuits may be of lower quality, causing them to fail the durability requirements for military use, but also because counterfeit electronics may contain malicious circuit insertions. This risk is

especially troubling, given that according to a study conducted by the Missile Defense Agency (MDA), twenty percent of suspect parts are military grade, as shown in Figure 1.

Table 1: Negative Effects of Counterfeit Electronics on the Government [Adapted from [6]]

Stakeholder	Negative Effects
Government	- Lost tax revenue due to illegal sales of counterfeit goods - Cost of Intellectual Property (IP) enforcement - Risk of counterfeits entering supply chains with national security or civilian safety implications

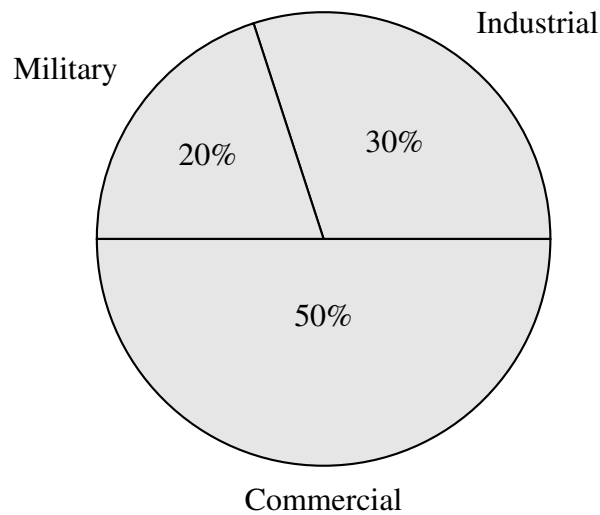


Figure 1: MDA data analysis of performance grade of suspect parts (Adapted from [7])

The government has responded to this growing threat by barring certain vendors from providing government equipment and by creating programs to identify counterfeit components [4, 18]. On 31 December 2011, President Obama signed the 2012 National Defense Authorization Act (Bill S.1867), which included the amendment S.Amdt.1092,

written to ensure that government personnel and contractors “detect and avoid counterfeit electronic parts and suspect counterfeit parts” [14, 18]. Avoidance can be accomplished by barring suspect vendors from providing products for use in U.S. government systems, and detection can be accomplished by continuing research begun by government programs.

1.2 Avoidance

Responding to the avoidance measure, on 8 October 2012, Chairman Mike Rogers and Ranking Member C.A. Dutch Ruppersberger of the Permanent Select Committee on Intelligence published a report in which they asserted that the risks associated with Huawei’s and ZTE’s provision of equipment to U.S. critical infrastructure could undermine core U.S. national-security interests. In a press conference, Chairman Mike Rogers claimed that equipment produced by Huawei exhibited unexpected behavior and even sent data to China [22]. As a result, he recommended that “U.S. government systems, particularly sensitive systems, should not include Huawei or ZTE equipment, including component parts” [15]. However, despite the efforts of the government to prevent counterfeit electronics from infiltrating government systems, the problem persists. As recently as September 2013, the IEEE *Spectrum* magazine featured an article explaining how counterfeit components are continuing to appear in and compromise military systems, such as the P-8A Poseidon aircraft [20].

1.3 Detection

Regarding detection through the continuation of research begun by government programs, one such program in place to help detect counterfeit electronics is the Defense Advanced Research Projects Agency (DARPA) Trusted Integrated Circuits (TRUST) program. Originally founded in 2007, the purpose of the DARPA TRUST program is to ensure the trust of ICs used in military systems, but designed and fabricated under untrusted conditions. Specifically, it aims to create a method of quantifying the amount of

trust placed in an integrated circuit based upon the probability of positively identifying an integrated circuit that was maliciously attacked, where a malicious attack is denoted by any change in the integrated circuit [4].

1.4 Detection Methods

The methods of detecting counterfeit electronics have taken on many different forms with the evolving sophistication of counterfeiting techniques. Beginning with visual inspection of counterfeit parts, detection methods have included (but are not limited to) rudimentary Direct Current (DC) bias testing and failure analysis, and more complex functional testing, physically unclonable function (PUF) implementations, and gate- and transistor-level testing [8–10]. Although functional testing and Physically Unclonable Function (PUF) implementations are advanced methods of detection, they both have shortcomings that render them insufficient for ensuring the trust of Integrated Circuits (ICs). Thus, of the detection methods listed, the method that shows the most promise of effectively detecting sophisticated counterfeit electronics is gate- and transistor-level testing.

1.5 Proposed Methodology

This research continues work in gate- and transistor-level testing that builds on the DARPA TRUST program. Prior work explored various methods of circuit comparison and matching in order to evaluate the capabilities of each method to identify whether or not an integrated circuit had been maliciously attacked. This research aims to resolve the issue of abstraction-level incongruence that prevented netlist matching between netlists created at the transistor level and netlists created at the gate level. In this paper, the term “netlist” describes a text-based representation of circuit components and connections. Such resolution will occur by applying standard cell recognition (SCR) to the transistor-level netlists in order to transform the netlists to the gate level.

1.6 Assumptions and Scope

Successful trials will apply SCR to a circuit's transistor-level netlist and match it to the same circuit's gate-level netlist. The following initial scope limitations will be placed on the research:

1. Circuit schematics will be assumed to be free of parasitic capacitances.
2. The netlist language used is Spectre™.
3. Circuits tested will be limited in complexity, containing <10 types of gates.
4. The technology for circuit designs will be limited to American Microsystems, Inc. (AMI) 0.6 um technology for fabrication through Metal Oxide Semiconductor Implementation Service (MOSIS).
5. The accuracy of the SCR methods will be evaluated based upon the ability to correctly match transistor patterns to their corresponding standard cells without false matchings.

1.7 Materials and Equipment

The research presented in this document will be performed in the Air Force Institute of Technology (AFIT) Micro-Electro-Mechanical Systems (MEMS)/Very Large Scale Integration (VLSI) Laboratory and the Air Force Research Laboratory (AFRL) Mixed Signal Design Center (MSDC), located on Wright Patterson Air Force Base (WPAFB). The materials and equipment needed include a Linux workstation for Cadence software and a Windows workstation for Idle, an Integrated Development Environment (IDE) for Python. The Cadence software (specifically, Cadence Virtuoso) will be used to create the circuit schematics and generate the corresponding netlists.

II. Background

The background information provided in this chapter contains two sections.

The first section describes various circuit verification methods and explores the shortcomings and/or advantages of each method. The second section describes the methodology of prior research in transistor-level verification and explains how the results prompted the research presented in Chapters 3 and 4.

2.1 Integrated Circuit Verification and Detection Methods

In order to determine the effectiveness of a detection or verification method, the probability of accurately verifying an integrated circuit as safe or malicious must be evaluated. This probability is defined by two metrics: the probability of detecting a malicious component when it exists (P_D) and the probability of a false alarm (detecting a malicious component when it does not exist) (P_{FA}) [9]. Naturally, the goal is to create methods of verifying chips that will maximize P_D and minimize P_{FA} . Two past verification methods created and evaluated were to conduct functional testing and to implement PUFs in the circuitry.

2.1.1 Functional Testing.

DARPA's Microsystems Technology Office (MTO) has evaluated functional testing within the scope of a 64-bit adder with two malicious transistors to cause errors in the 61st bit of the adder. In performing the functional test of the adder, the probability of detecting that a malicious component exists is 100 percent due to the erroneous output, but P_{FA} is unacceptably high due to the fact that functional testing does not provide the ability to determine which specific transistor(s) or component(s) within the adder is (are) malicious [9]. A visual representation of this problem is represented in Figure 2. The red color indicates malicious transistors, and the green color indicates benign transistors.

Functional testing is capable of identifying the whole adder as malicious, but is not capable of distinguishing the red from the green transistors.

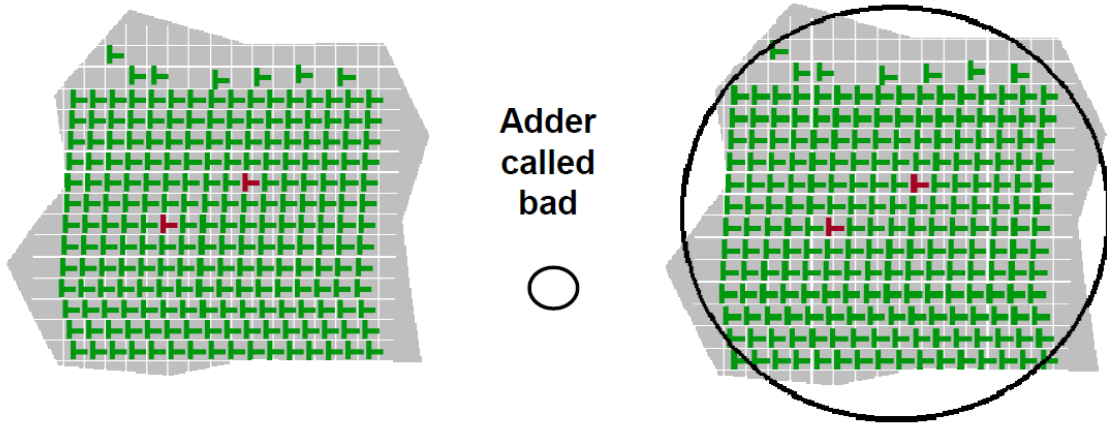


Figure 2: Functional test of the 64-bit adder [9]

Although functional testing presented $P_D = 1$ in the example with the adder, there are other cases in which functional testing can fail completely by producing $P_D = 0$. One such case is explicitly mentioned in a DARPA area of interest in which it is necessary to determine if an IC corresponding to a known design performs extra functions in addition to its specified function [9]. That is, the IC produces the specified outputs, but also produces outputs that are extraneous to what was originally intended in the initial design. These extra functions are due to the insertion of malicious circuitry into the original design. A visual representation of the malicious circuits can be seen by the two red blocks labeled “Circuit 1” and “Circuit 2” in Figure 3.

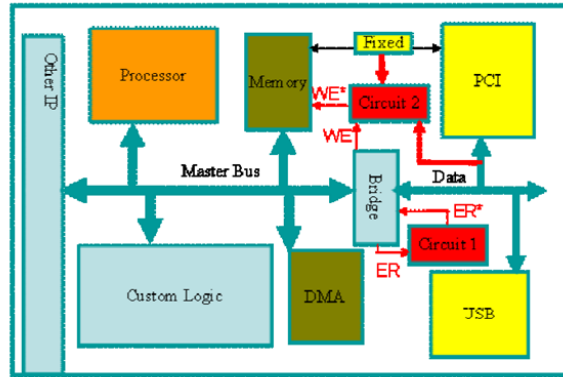


Figure 3: A standard IC with malicious circuits (Circuits 1 and 2) added [10]

2.1.2 Physically Unclonable Functions.

In the method of physically unclonable functions (PUFs), information regarding the expected timing and delay of a circuit is leveraged as two or more challenge bits are implemented at the same time through a set of multiplexers, and the value of the output depends on which signal traveled the fastest through the circuit. This concept is represented in Figure 4.

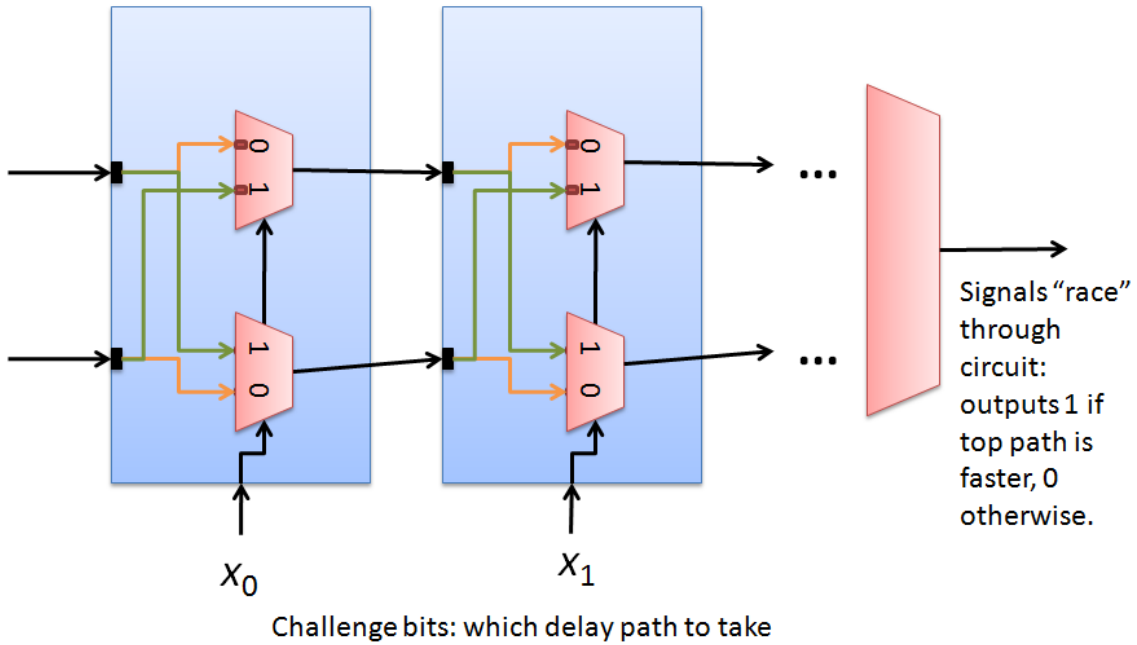


Figure 4: Physically unclonable functions [10]

However, two problems exist when utilizing PUFs. First, for the same reason as identified with conducting functional testing on the adder, P_{FA} is unacceptably high, and further potential for a high P_{FA} exists when taking into account IC wearout at the device level. For example, gate oxide wearout, electromigration, and self-heating of interconnections can adversely affect the timing of a circuit [21]. Gate oxide wearout causes a shift in threshold voltage and an increase in gate leakage, causing a transistor to

perform too slowly. Electromigration can cause a void in the interconnections, terminating a signal prematurely. Self-heating of interconnections increases the temperature of the wires, causing them to have a higher resistance and delay. As the outputs of the PUFs are directly affected by the timing, device wearout can cause the output to be inconsistent with what is expected given the initial timing information. As a result, the PUF implementation method would indicate that the circuit has been maliciously attacked when in reality, the device is simply past its useful operating life.

The second problem is that an attacker could easily model the PUFs or modify the remainder of the chip without affecting the performance of the PUFs [10]. While this issue does not affect P_{FA} , a greater sophistication of the attacker's design lowers the probability of detection.

2.1.3 Shortcomings of Functional Testing and Physically Unclonable Function Implementations.

The potential for functional testing and PUF implementations to produce a probability of detection of zero is modeled in Figure 5. In this full adder circuit, it is specified that neither the full adder cell nor the PUF implementation have been maliciously modified in any way. The full adder cell produces the proper expected outputs of S and C_{out} , and likewise, the PUF produces the outputs that the designer would expect. Thus, both methods of functional testing and the implementation of PUFs would determine that the full adder circuit has not been maliciously modified. However, it is clear that this is an incorrect determination, given that the malicious extraneous component exists. The malicious extraneous component taps into the outputs of the full adder cell and sends the signals through a NOR gate, which produces an extraneous output unbeknownst to the designer and tester.

Figure 6 illustrates the simulated inputs and outputs of the circuit depicted in Figure 5. The circuit inputs include the full adder cell inputs ($A0$, $A1$, and C_{in}) and the

PUF inputs ($B0$, $B1$, $B2$, and $B3$). The circuit outputs include the expected outputs from the full adder cell (S and C_{out}), the expected output from the PUF (PUF_{out1}), and the malicious extraneous output (X_{out}).

Figure 7 depicts the full adder circuit with the full adder cell and PUF, but without the malicious extraneous circuit. Figure 8 illustrates the simulated inputs and outputs of the circuit presented in Figure 7. The circuit inputs include the full adder cell inputs ($A0$, $A1$, and C_{in}) and the PUF inputs ($B0$, $B1$, $B2$, and $B3$). The circuit outputs include the expected outputs from the full adder cell (S and C_{out}) and the expected output from the PUF (PUF_{out1}).

By comparing Figure 6 with Figure 8, it can be seen that the addition of the malicious extraneous NOR gate produces an extraneous output without affecting the expected outputs. Hence, it is possible to maliciously alter a circuit in such a way that neither a PUF implementation nor functional testing could detect the modification.

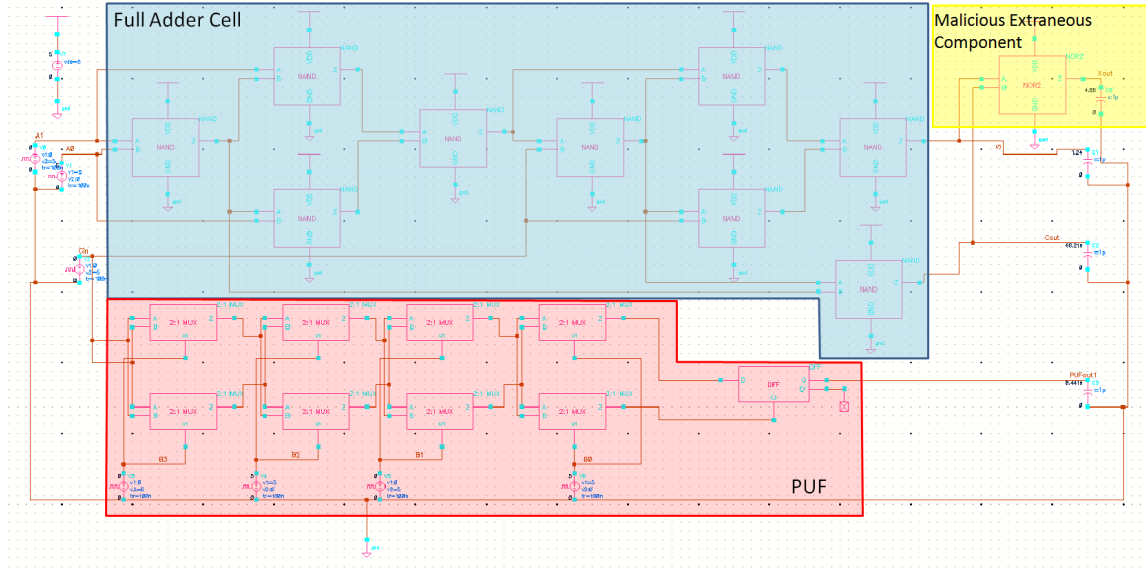


Figure 5: Full adder circuit with PUF and malicious extraneous component

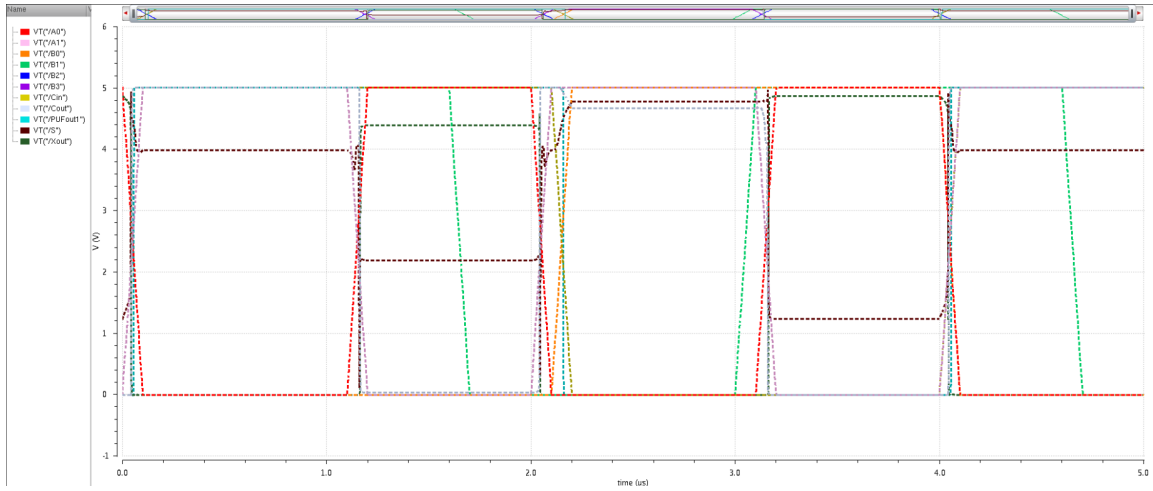


Figure 6: Simulated inputs and outputs of the full adder circuit with PUF and malicious extraneous component

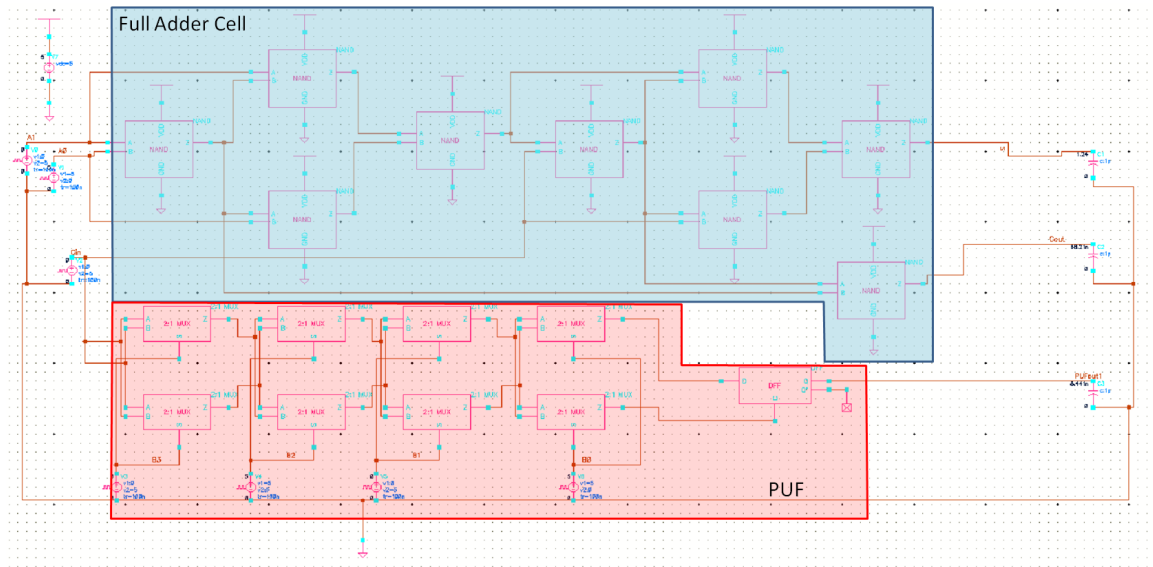


Figure 7: Full adder circuit with PUF only

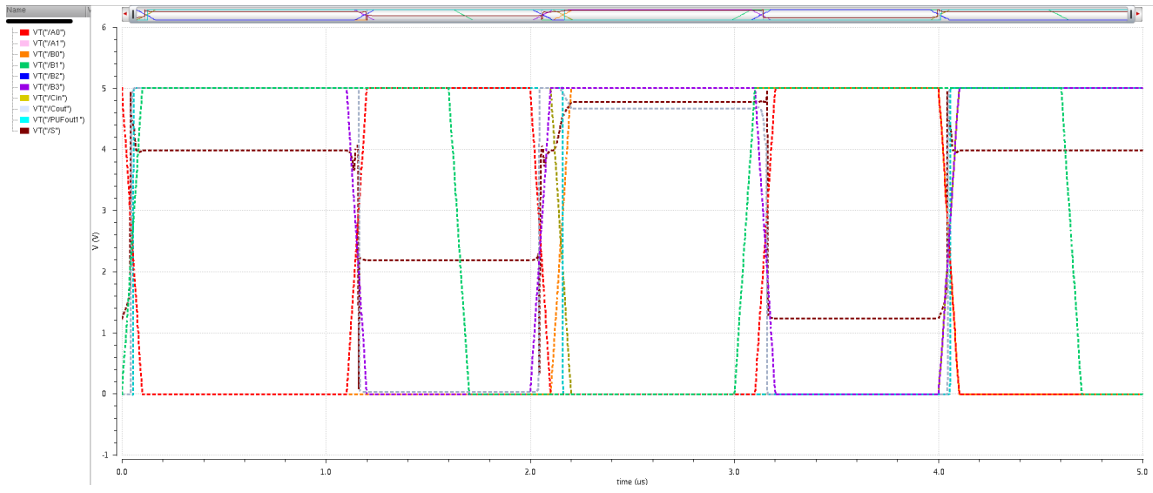


Figure 8: Simulated inputs and outputs of the full adder circuit with PUF only

2.1.3.1 Transistor- and Gate-Level Testing.

Due to the inability to consistently achieve both a high P_D and a low P_{FA} with these two methods, current research has shifted focus to transistor-level and gate-level testing. While a lower P_D must be tolerated until more precise methods of transistor-level testing have been achieved, P_{FA} is desirably low such that the individual malicious transistors or components can be uniquely identified. Ideally, the DARPA TRUST program aims for target metrics of $P_D = .99$ and $P_{FA} = 1\text{E-}6$ for a problem size of 50 million transistors and a detection time of 120 hours [19].

The process of verifying a circuit at the transistor or gate level includes ensuring that the circuit designed in the forward design flow matches the circuit fabricated in the reverse design flow. The forward and reverse design flows are represented in Figure 9. The forward design flow involves the following sequence of steps:

1. Design in Register Transfer Language (RTL) and inclusion of necessary IP cores (Window 1)
2. Device synthesis and optimization, and test insertion (Window 2)
3. Clock insertion and Place and Route (Window 3)
4. Mask generation, fabrication, functional testing, and system integration (Window 4 and beyond)

Referencing the figure, it is important to notice that a forward netlist is generated between test insertion (Window 2) and clock insertion (Window 3). The reverse design flow involves de-layering a fabricated chip, capturing the metallization and connectivity of the chip, and generating a netlist to show the electronic equivalent representation.

Under the DARPA TRUST program, software tools to aid in verification were developed by various contractors, one of which is Raytheon. Prior to 2012, Raytheon

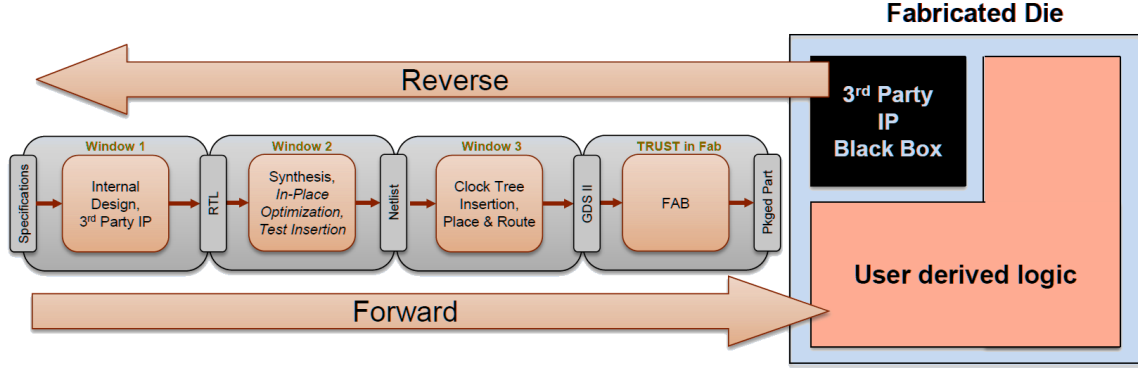


Figure 9: Forward and reverse IC design flows [19]

applied these tools applied to TRUST test cases, but not to real-world circuits. However, in 2012, they transitioned their tools to the AFRL MSDC. These tools were used in combination with the Cadence Design software suite in order to explore the ability to achieve a high P_D and a low P_{FA} with real world circuits [19].

2.2 Summary of Recent Work

As of March 2014, prior research had successfully achieved the implementation of these tools within a limited scope. The first scope limitation is to focus solely on achieving a low P_{FA} . Given that all the test articles are known to be free of malicious insertions, P_D is irrelevant. The second scope limitation is on the design flow; the scope of the research is limited to Windows 1 through 3 in the forward direction and Windows 3 through 1 in the reverse direction. The design flow for the research conducted at the AFRL MSDC is represented in Figure 10, in which the windows are called “phases,” as the phases in the AFRL MSDC design flow do not exactly match the windows in the general design flow.

Successful verification is identified as a perfect matching between the golden and the revised netlists, where the golden netlist is defined as the netlist generated during Phase 2

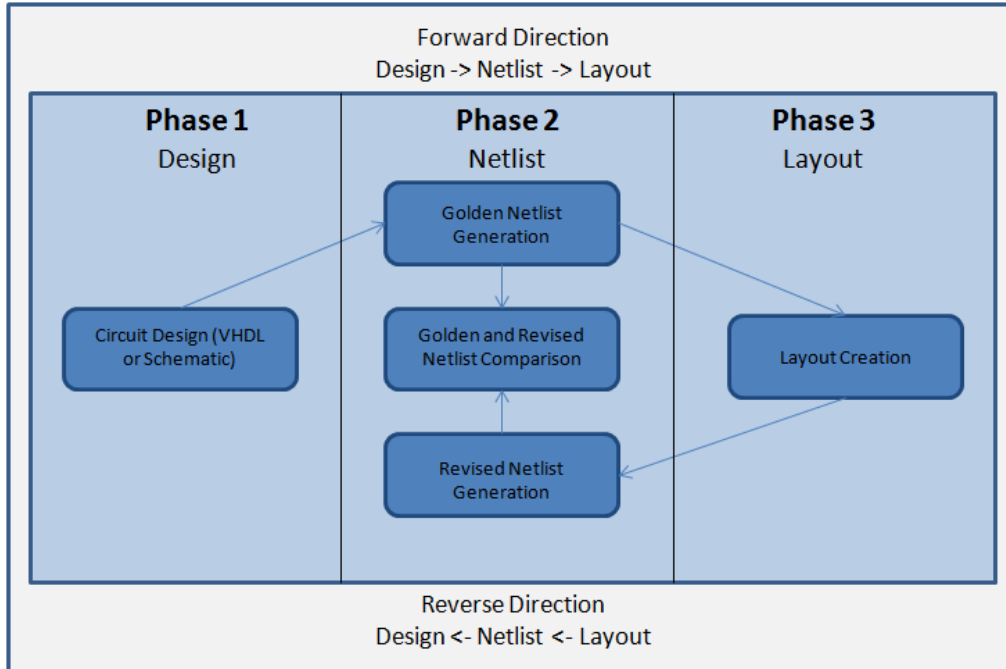


Figure 10: Limited scope forward and reverse IC design flows

in the forward direction, and the revised netlist is defined as the netlist generated during Phase 2 in the reverse direction.

The tests conducted for circuit verification include a) transistor-level testing of a single bit full-adder cell of a transistor-based architecture (Circuit A) and b) gate-level testing of a clocked inverter (Circuit B), an Inter-Integrated Circuit (I2C) bus communication core (Circuit C), a full adder of a gate-based architecture (Circuit D), and an Advanced Encryption Standard (AES) cryptography core (Circuit E). With Circuit A, as seen in Figure 11, the golden netlist is derived from a schematic in Cadence Virtuoso software and generated with NC-Verilog, and the revised netlist is extracted with Cadence Virtuoso from the custom layout and generated with NC-Verilog. The golden and revised netlists are then compared in Cadence Conformal.

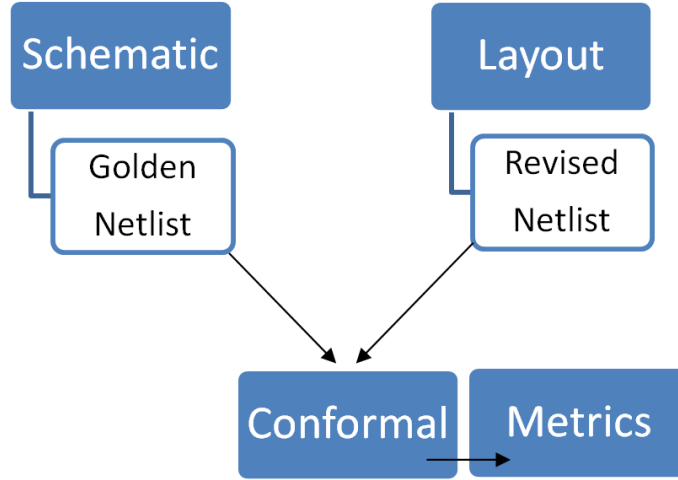


Figure 11: Circuit A conceptual process [19]

After some human interactions (see Figures 12 and 13), including converting global signals to cell-level I/O pins, hand-mapping points that are excluded from the mapping process but not identified as being un-mapped, adjusting the ordering of components in series, and correcting transistor directionality, 100% transistor-level matching between the two netlists is achieved.

Since all points are now successfully mapped, it can be concluded that no false alarms occurred, meaning that, for this instance in Circuit A, P_{FA} was minimized to zero [19].

For Circuit B, the golden netlist is derived from Very-High-Speed Integrated Circuit Hardware Description Language (VHDL) code and compiled into Verilog with Cadence RTL Compiler. The revised netlist is derived by generating a floorplan layout from the golden Verilog netlist with Cadence Encounter, and then generating another netlist from the layout. With this process, 100% gate-level matching is achieved in Cadence Conformal, and P_{FA} is minimized. Netlists for Circuits C are generated in the same manner as those from Circuit B, and perfect matching is also achieved, which is expected

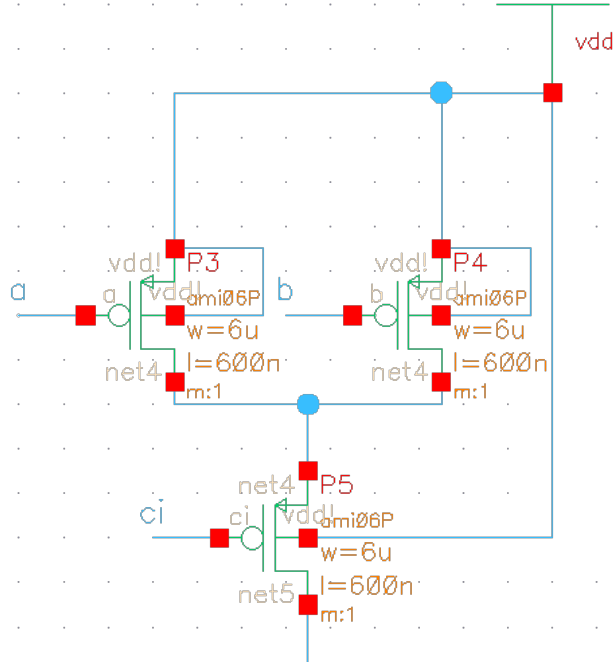


Figure 12: Part of Circuit A schematic before corrections [19]

given that Encounter's backend is the same as the RTL Compiler frontend. The same method is applied to Circuit D, and, naturally, perfect matching is achieved in Conformal. Unfortunately, Circuit E is not verified as the toolset does not have the capabilities to complete the production of the golden netlist [19].

Regarding Circuit D, the verification process does not end with netlist matching in Conformal; a different avenue of verification is pursued for the sake of surveying the capabilities of other tools. The floorplanned design is imported into Cadence Virtuoso in order to produce a transistor-level representation of the design. A netlist is then generated from the transistor-level layout, which produces a transistor-level netlist format. This results in a problem which prevents netlist matching, as there is an incongruence in abstraction levels between the gate-level golden netlist and the transistor-level revised netlist [19]. Hence, it is necessary to perform SCR on the transistor-level revised netlist to transform it into a gate-level revised netlist. One such method of SCR has been performed

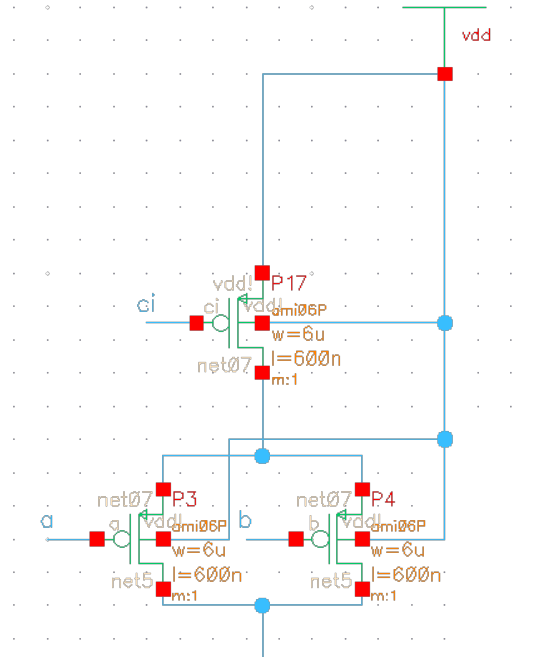


Figure 13: Part of Circuit A schematic after corrections [19]

independently of DARPA's efforts by Wonjong Kim and Hyunchul Shin of Hanyang University [12].

2.2.1 Standard Cell Recognition Efforts by Wonjong Kim and Hyunchul Shin of Hanyang University.

Wonjong Kim and Hyunchul Shin have explored a method of hierarchical netlist extraction using an algorithm that performs three main functions: merge series transistors, find simple gates, and match hierarchical subcircuits [12]. In the context of the research conducted by Wonjong Kim and Hyunchul Shin, their concept of levels of hierarchy is equivalent to the concept of levels of abstraction presented in this research.

2.2.1.1 Merge Series Transistors.

The first part of this algorithm is to merge series transistors into a multi-gate device by searching for nets that, for the same type of transistors (either p-type or n-type), connect only two source/drain terminals. Excluded from the merger, though, are nets that connect to an external terminal in a larger circuit [12].

2.2.1.2 Find Simple Gates.

The second part of the algorithm is to use the merged series transistors to find simple gates, including inverters, NOR gates, and NAND gates.

For inverters, the algorithm takes into account the fact that inverters are composed of a p-type transistor and an n-type transistor that share a common source/drain signal (the source and drain of a transistor are interchangeable) and gate signal. The other source/drain signal is connected to V_{dd} for the p-type transistor and ground (gnd) for the n-type transistor.

NOR gates are partly composed of p-type transistors in series with n-type transistors in parallel. Hence, when it is identified that there are merged series p-type transistors that have one source/drain connection with V_{dd} , the algorithm searches for n-type transistors that a) share a source/drain connection with the merged series p-type transistors and b) share gate connections with the merged series p-type transistors. If the algorithm finds the

n-type transistors that meet the two criteria, the n-type transistors and corresponding p-type transistors are replaced with a NOR gate.

Similarly, NAND gates are partly composed of p-type transistors in parallel and n-type transistors in series. When it is identified that there are merged series n-type transistors and one source/drain connection is gnd, the algorithm searches for p-type transistors that share both a source/drain connection and gate connections with the merged series n-type transistors. If the p-type transistors are found, both the p-type transistors and corresponding n-type transistors are replaced with a NAND gate [12].

2.2.1.3 Match Hierarchical Subcircuits.

The third and chief part of the algorithm is to find subcircuits from the netlist. This is done recursively from the lowest hierarchical level to the highest level since a subcircuit can be identified only after the child subcircuits which compose the parent have been identified. To actually find each subcircuit, a modified version of the SubGemini algorithm is used first to identify all possible matchable locations of the subcircuit in the layout netlist and second to determine if a subcircuit actually exists at each of the possible locations [12]. To complete the first task, the algorithm sets a key node in the schematic netlist and searches for a candidate vector, which is a set of nodes that potentially match the key node. To complete the second task, each node in the candidate vector is examined to determine which of the nodes in the schematic's subcircuit graph map to the nodes in the layout graph, such that the nodes from the candidate vector match the key node. To accomplish this, a match between the key node and a node in the candidate vector is assumed and the two nodes are uniquely labeled. Using the two nodes as a starting point, the subcircuit and layout netlists are simultaneously given matching labels only if a valid mapping between the two graphs exists. A subcircuit is positively identified when, for all the subcircuit nodes from the schematic netlist, there are labels that have a perfect match in the layout netlist.

However, there are instances in which a subcircuit can be falsely identified. The algorithm identifies this type of error by comparing the number of candidates in the layout netlist for a subcircuit with the number of subcircuits used in the schematic netlist. If the numbers are not the same, then a subcircuit is declared to be falsely identified and is expanded in the next iteration of the algorithm (the next level of hierarchy) [12].

2.2.1.4 Algorithm Insufficiencies as Applied to DARPA's Circuit Verification Efforts.

Unfortunately, the algorithm in [12] is insufficient for application to DARPA's circuit verification methods. For all intents and purposes, the layout netlist in this algorithm can be considered a revised netlist, and the schematic netlist can be considered the golden netlist. In the previous section, it was mentioned that the correctness of identifying subcircuits is based upon a comparison between the layout (revised) netlist and the schematic (golden) netlist. This method inherently bases its operation on the assumption that the circuit represented by the revised netlist is perfectly equivalent to the circuit represented by the golden netlist. Given that a goal of DARPA's efforts is to identify circuit layouts that have been modified from the original schematic, an algorithm suitable to achieve this goal must conduct SCR solely on the revised netlist and independently of the golden netlist. The research presented in this paper aims to create a novel algorithm that conducts SCR without referencing a golden netlist so that it is suitable for application to DARPA circuit verification. The next section will discuss the methodology of producing this novel algorithm.

III. Methodology

THIS research seeks to conduct SCR on a transistor-level netlist of a circuit equivalent to Circuit D, referenced in the preceding chapter. The goal is that SCR will create a gate-level format from the transistor-level format by identifying transistor patterns and matching them to standard cells. Two avenues of developing SCR technology exist. The first avenue is with software developed by Stanford Research Institute (SRI), and the second is by writing original Python code. With regard to software developed by SRI, Mr. Saverio Fazzari, a DARPA Systems Engineering and Technical Assistance (SETA) contractor, has contacted the AFRL MSDC to continue work in developing software that originated in the DARPA Integrity and Reliability of Integrated Circuits (IRIS) program. Pursuing research with SRI software would involve the following four components:

1. Investigating the portability of the software.
2. Building a technology base on how to implement the tool.
3. Reproducing the results achieved in the DARPA IRIS program.
4. Applying the tool to test articles that were previously unexplored with regard to the SRI software.

The completion of parts one and two would need to be accomplished by the AFRL MSDC before this research could continue with parts three and four.

Presently, none of the software available at AFIT or the AFRL MSDC are capable of conducting SCR at even the elementary level, so it is necessary to create a new, original program to conduct SCR. For this reason, four phases of research using Cadence software and Python code are proposed. The four phases are:

1. Gain familiarity with software tools.

2. Apply software tools to elementary gates, gradually increasing in complexity.
3. Apply software tools to conduct SCR on a netlist of a circuit equivalent to Circuit D.
4. Conduct analysis of SCR code to determine level of maturity.

These phases are explained in detail in the sections below.

3.1 Phase 1 Methodology - Gaining Familiarity with Software Tools

Gaining familiarity with the software tools involves various activities, such as working in the software environment of Cadence Virtuoso and Idle (the default integrated development environment (IDE) bundled with Python software) and completing tutorials. The necessary skills to gain with Cadence Virtuoso in order to complete this research include:

1. Creating a flat (transistor-level) circuit schematic;
2. Creating gate-level cells;
3. Creating a gate-level schematic;
4. Generating circuit netlists.

The necessary skills to gain with Python in order to complete this research include:

1. Creating class instances;
2. Creating functions to perform SCR operations;
3. Passing objects between functions;
4. Reading/writing netlists to/from files.

3.2 Phase 2 Methodology - Software Tool Application to Elementary Gates

Given that this SCR effort does not build on any pre-existing programs for conducting SCR, the SCR algorithm must be written as simply as possible. Thus, two major process simplifications are made to reduce the algorithm complexity. First, transistor-level netlists are generated only in the Spectre netlist language. Second, the transistor-level netlists are generated from schematics rather than layouts in order to avoid complexities encountered in layouts. Such complexities include parasitic capacitances and transistors in parallel appearing as one transistor with the width equal to the sum of the separate transistors widths.

Additionally, the algorithm must be created initially to conduct SCR at an elementary level and then incrementally scaled in sophistication to detect increasingly complex gates. Hence, research in this phase will focus first on writing the portion of the SCR code to identify simple transistors. Then, it will focus on identifying gates of increasing complexity and abstraction levels, as seen in Table 2.

The abstraction level of each type of gate is shown in Figure 14.

Table 2: Gates to be identified by SCR algorithm

Gate	# Transistors	# Input signals	# Output signals	Abstraction level
INV	2	1	1	2
NAND2	4	2	1	2
NOR2	4	2	1	2
AND2	6	2	1	3
OR2	6	2	1	3
NAND2b0	8	2	1	4
OAI21	10	3	1	4
OAI21b1	12	3	1	4
OAI21b0b1	14	3	1	4

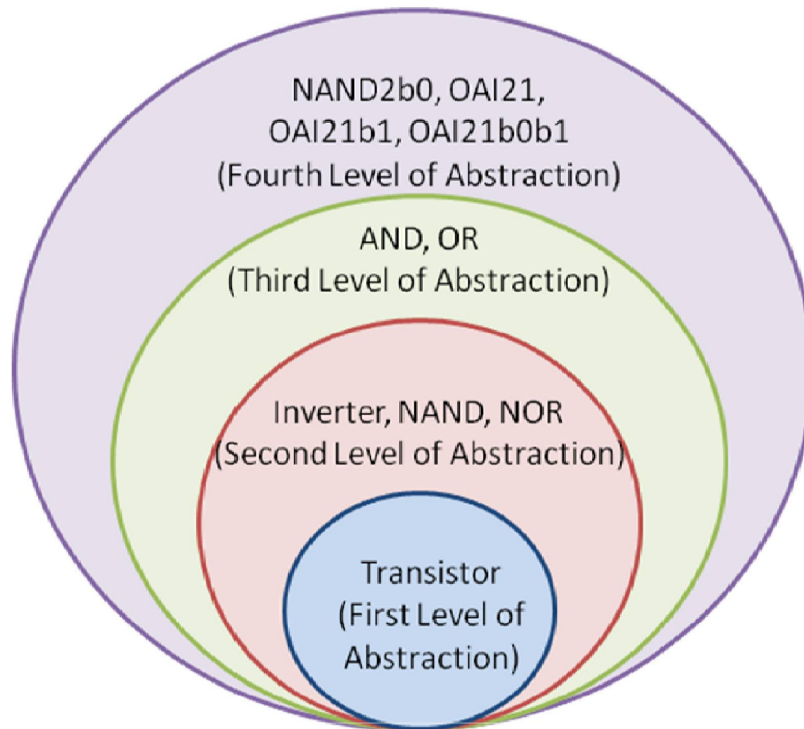


Figure 14: Abstraction levels of various gates

3.3 Phase 3 Methodology - Software Tool Application to Circuit D Equivalent

Revisiting Circuit D, a gate-level schematic of the circuit is presented in Figure 15. It can be seen in Figure 15 that the adder cell is composed of nine standard cells. Figure 16 presents the same adder cell at a lower level of abstraction, constituting 15 gates. Figure 17 presents the adder cell at the lowest level of abstraction, constituting 20 gates. From Figure 17, a transistor-level representation of the cell is generated and represented in the schematic in Figure 18. This phase of research is intended to conduct SCR on the transistor-level netlist of the cell represented in Figure 18 in order to reproduce a gate-level netlist equivalent to the netlist generated from the schematic in Figure 15.

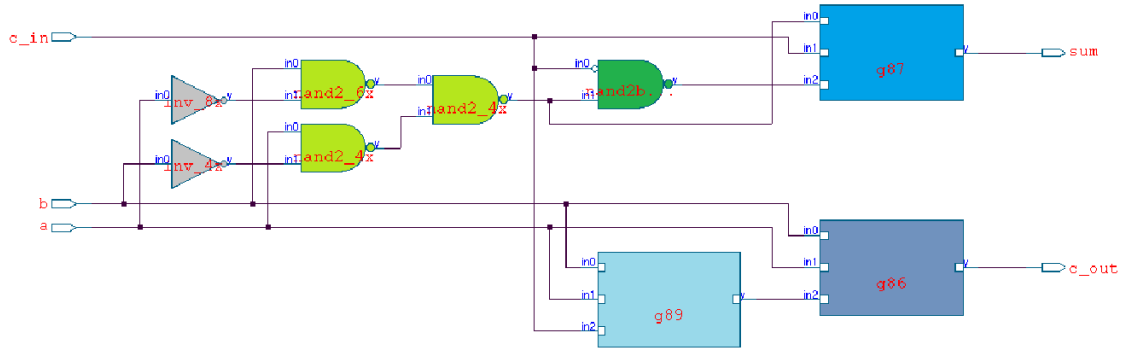


Figure 15: A schematic of the complex Circuit D [Adapted from [19]].

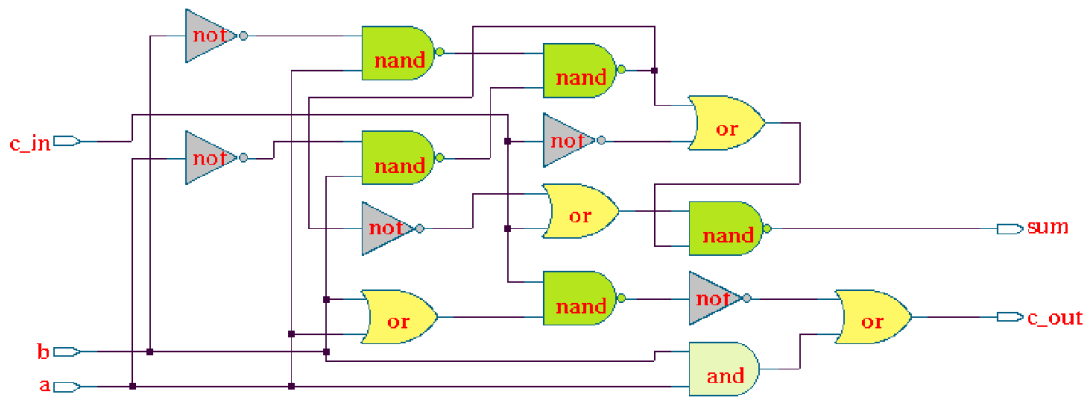


Figure 16: A schematic of the complex Circuit D, mid-level abstraction (Image courtesy of M. Seery).

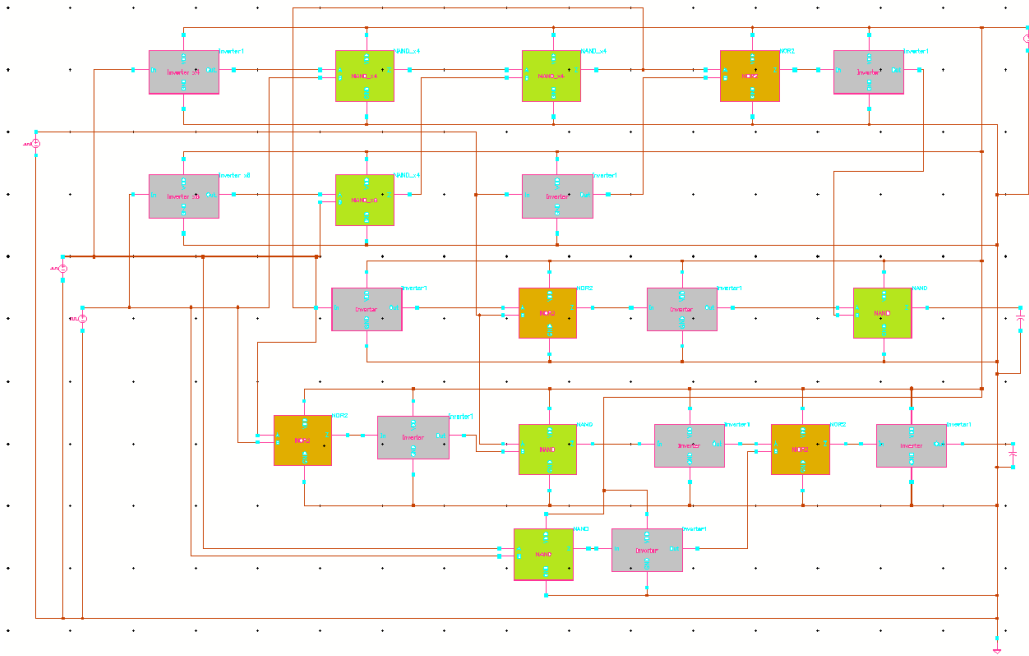


Figure 17: A schematic of the complex Circuit D, low-level abstraction.

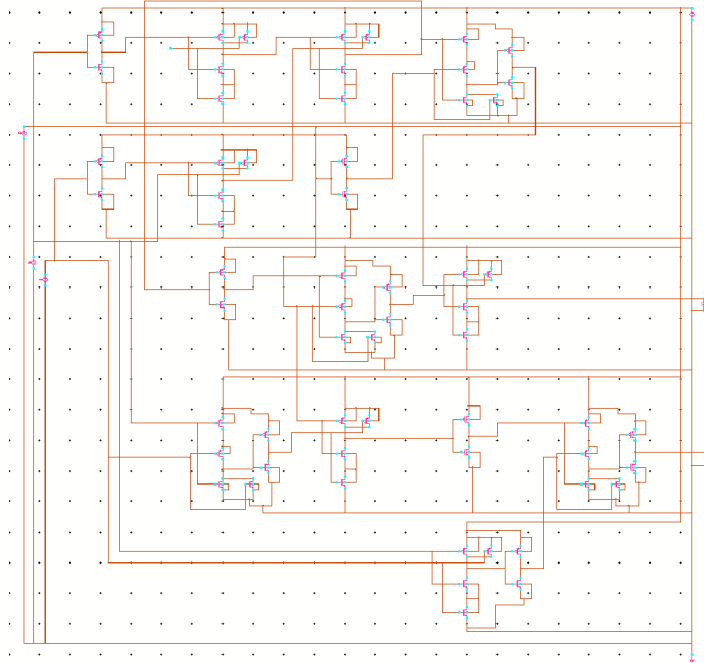


Figure 18: A flat schematic of the complex Circuit D.

3.4 Phase 4 Methodology - SCR Code Analysis

Once SCR code has been written to successfully conduct SCR on a circuit equivalent to Circuit D, the next step is to conduct an analysis to determine the SCR code's level of maturity. The level of maturity will be evaluated based upon recognition accuracy and comprehensiveness. Recognition accuracy will be tested by applying the SCR code to nine unique circuits. The circuits will be constructed using only the set (or subset) of gates listed in Section 3.2. Variety will be introduced into the set of circuits by:

1. Including circuits designed by individuals without visibility into the development of the code.
2. Varying the gate configurations.
3. Varying the transistor and gate counts of the circuits.

Comprehensiveness will be evaluated based upon the percentage of cells contained in the North Carolina State University (NCSU) Digital Parts standard cell library that the code could currently recognize. Additionally, comprehensiveness will be evaluated based upon the percentage of cells that the code is capable of identifying in five different TRUST test circuits.

IV. Results

THIS chapter contains three sections. The first section, Section 4.1, presents and evaluates each phase's research results achieved by applying the methodology described in Chapter 3 to test articles. The second section, Section 4.2, explores the SCR code developed in the first three phases of the research, specifically discussing the resulting SCR code structure and the implementation of the algorithm used to conduct SCR. The third section, Section 4.3, explores the application of the the advantages of transistor-level testing with SCR over functional testing as a circuit verification method.

4.1 SCR Research Results

4.1.1 Phase 1 Results - Gaining Familiarity with Software Tools.

4.1.1.1 Cadence Virtuoso Implementation.

Phase 1 of the research involved exploring the capabilities of Cadence Virtuoso and Idle. The capabilities of Cadence Virtuoso identified as necessary for the research in 3.1 are:

1. Creating a flat (transistor-level) circuit schematic.
2. Creating gate-level cells.
3. Creating a gate-level schematic.
4. Generating circuit netlists.

Figures 19, 20, 21, 22, and 23 represent examples of the results attained by realizing each skill in Cadence Virtuoso. Figure 19 shows a custom flat inverter circuit schematic designed with this tool, which demonstrates the realization of the first required capability of creating a flat schematic. Figure 20 shows the corresponding custom-designed inverter symbol, a gate-level cell utilized in the construction of gate-level schematics. The creation

of the inverter symbol denotes the attainment of the second required capability of creating gate-level cells. Figure 21 shows a gate-level inverter circuit schematic, a fulfillment of the third required capability of creating a gate-level schematic. Figures 22 and 23 show the netlists generated from the flat and gate-level schematics, respectively, which demonstrates the achievement of the fourth required capability of generating circuit netlists.

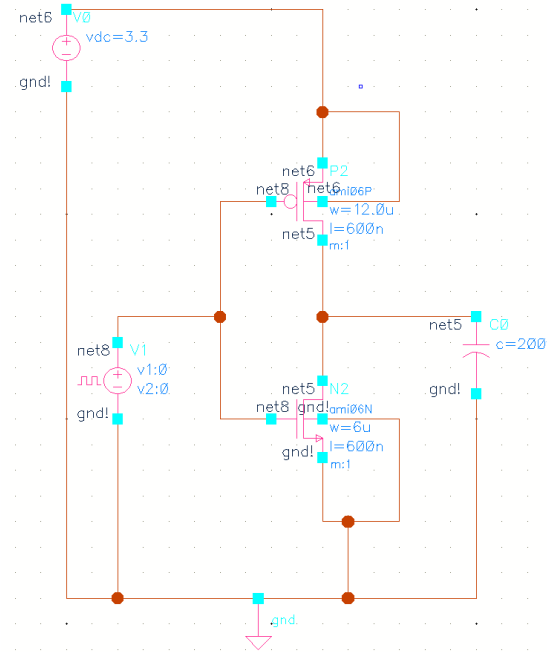


Figure 19: Flat inverter circuit schematic designed with Cadence Virtuoso

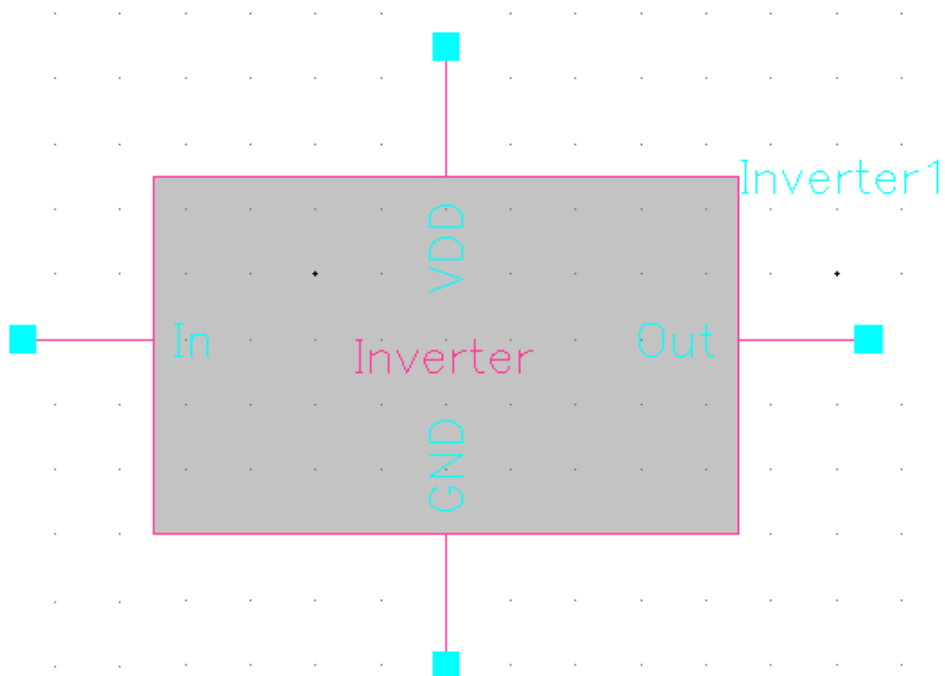


Figure 20: Inverter symbol designed with Cadence Virtuoso

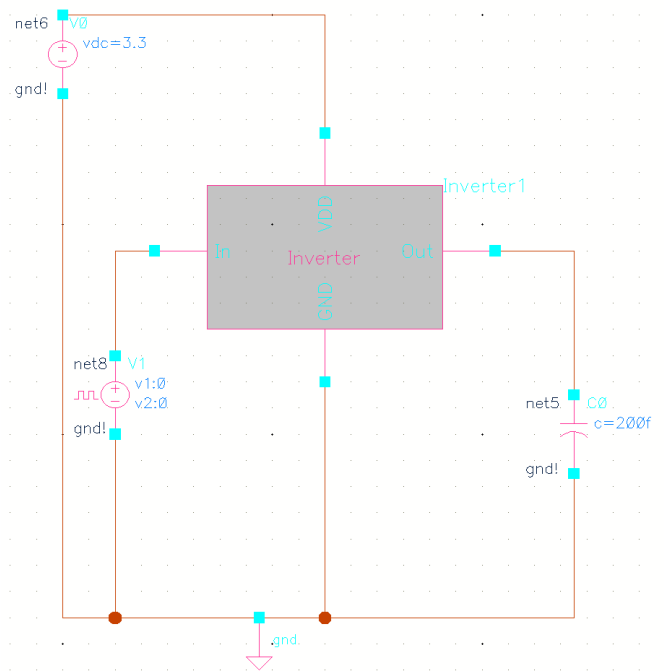


Figure 21: Gate-level inverter circuit designed with Cadence Virtuoso

```

// Generated for: spectre
// Generated on: Oct 7 11:31:49 2014
// Design library name: Thesis
// Design cell name: inverter
// Design view name: schematic
simulator lang=spectre
global 0

// Library name: Thesis
// Cell name: inverter
// View name: schematic
N2 (net5 net8 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u pd=15.0u \
    m=1 region=sat
P2 (net5 net8 net6 net6) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
    ps=27.0u pd=27.0u m=1 region=sat
C0 (net5 0) capacitor c=200f m=1
V0 (net6 0) vsource type=dc dc=3.3
V1 (net8 0) vsource type=pulse val0=0 val1=0
simulatorOptions options reltol=1e-3 vabstol=1e-6 iabstol=1e-12 temp=27 \
    tnom=27 scalem=1.0 scale=1.0 gmin=1e-12 rforce=1 maxnotes=5 maxwarns=5 \
    digits=5 cols=80 pivrel=1e-3 sensfile="../../psf/sens.output" \
    checklimitdest=psf
modelParameter info what=models where=rawfile
element info what=inst where=rawfile
outputParameter info what=output where=rawfile
designParamVals info what=parameters where=rawfile
primitives info what=primitives where=rawfile
subckts info what=subckts where=rawfile
saveOptions options save=allpub

```

Figure 22: Flat inverter circuit netlist generated by Cadence Virtuoso

```

// Generated for: spectre
// Generated on: Oct 13 11:50:05 2014
// Design library name: Thesis
// Design cell name: inverter_circuit_hier
// Design view name: schematic
simulator lang=spectre
global 0

// Library name: Thesis
// Cell name: inverter_cell
// View name: schematic
subckt inverter_cell GND In Out VDD
  N2 (Out In GND GND) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
    pd=15.0u m=1 region=sat
  P2 (Out In VDD VDD) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
    ps=27.0u pd=27.0u m=1 region=sat
ends inverter_cell
// End of subcircuit definition.

// Library name: Thesis
// Cell name: inverter_circuit_hier
// View name: schematic
I2 (0 net8 net5 net6) inverter_cell
C0 (net5 0) capacitor c=200f m=1
V0 (net6 0) vsource type=dc dc=3.3
V1 (net8 0) vsource type=pulse val0=0 val1=0
simulatorOptions options reltol=1e-3 vabstol=1e-6 iabstol=1e-12 temp=27 \
  tnom=27 scalem=1.0 scale=1.0 gmin=1e-12 rforce=1 maxnotes=5 maxwarns=5 \
  digits=5 cols=80 pivrel=1e-3 sensfile="../psf/sens.output" \
  checklimitdest=psf
modelParameter info what=models where=rawfile
element info what=inst where=rawfile
outputParameter info what=output where=rawfile
designParamVals info what=parameters where=rawfile
primitives info what=primitives where=rawfile
subckts info what=subckts where=rawfile
saveOptions options save=allpub

```

Figure 23: Gate-level inverter circuit netlist generated by Cadence Virtuoso

4.1.1.2 Idle IDE Implementation.

The necessary Idle capabilities identified in 3.1 are using Python to:

1. Create class instances. This capability is necessary for the ability to create gate objects (inverter, NAND2, NOR2, etc.) that have attributes as described in Section 4.2.1.1.
2. Create functions to perform SCR operations. This capability enables the execution of the functions described in Section 4.2.1.2.
3. Pass objects between functions. This capability contributes to the successful execution of the functions that accept or return objects as described in Section 4.2.1.2.
4. Read/write netlists to/from files. This capability is required in order for the SCR algorithm to accept an input netlist and produce an output netlist.

The capabilities were demonstrated by writing and testing Python SCR code to perform SCR on a simple inverter. Regarding the creation of class instances, the classes defined were: NMOS, PMOS, inverter, and inverter_type. Instances of each class are stored and tracked in the arrays “NMOSx,” “PMOSx,” “inverters,” and “inverter_types,” such that NMOS class instances are stored in NMOSx, PMOS class instances are stored in PMOSx, etc. The functions designed to read/write netlists to/from files and perform SCR operations are described in Table 3.

By implementing the Python SCR code described, the ability to create class instances and SCR functions, pass objects between functions, and read/write netlists to/from files was demonstrated.

Table 3: Name, Description, and Passed Objects of Stage I Python SCR Functions

Function Name	Description	Objects Passed
find_tx()	Reads flat netlist and finds the lines correlating to transistors in the netlist	Netlist file name
create_NMOS_objects()	Creates NMOS class instances of the NMOS transistors found and stores them in NMOS _{tx} array	Netlist file name
create_PMOS_objects()	Creates PMOS class instances of the PMOS transistors found and stores them in PMOS _{tx} array	Netlist file name
find_inverter()	Examines the transistor connections, identifies inverter patterns, and stores the inverters in the inverters array	Current inverter count
replace_inverter()	Removes the transistors belonging to inverters from the netlist and replaces them with the inverter instances	Netlist file name

4.1.2 Phase 2 Results - Software Tool Application to Elementary Gates.

Building on the methodology discussed in Section 3.2, this phase of research focused on writing SCR code to implement an algorithm that first identifies simple transistors and then identifies gates of increasing complexity and abstraction level. The algorithm identifies gates exactly as described in the subsequent sections. That is, the algorithm recognizes gates solely based on transistor and/or sub-gate connections. Feature sizes, technology sizes, etc. do not affect gate recognition.

The portion of this phase intended to identify instances of transistors was accomplished in Phase 1 with the testing of Idle's abilities. Likewise, developing SCR code to identify inverters was also accomplished in Phase 1. Inverters, which are gates in the second abstraction level, are identified by finding an NMOS/PMOS pair of transistors that share a common drain connection and gate connection, but do not share a common source connection. The transistor-level circuit schematic and equivalent gate-level circuit schematic for the inverter are shown above in Figure 19 and Figure 21, respectively.

4.1.2.1 NAND2 Gate.

Unique to Phase 2 of the research was the development of SCR code to recognize the NAND2 gate, a gate in the second abstraction level. As inverters are not identified based upon source connections to VDD or GND, NAND2 gates are identified by examining transistors connected to the inverters. Specifically, NAND2 gates are identified by finding an NMOS transistor (N1) whose drain is connected to the source of the inverter's NMOS transistor (N0) and a PMOS transistor (P1) that shares a common gate connection with N1, a common source connection with the PMOS transistor of the inverter (P0), and a common drain connection with P0. This results in finding a gate with two NMOS transistors in series and two PMOS transistor in parallel. Figure 24 depicts the transistor-level schematic of the NAND2 gate, and Figure 25 shows the equivalent gate-level schematic.

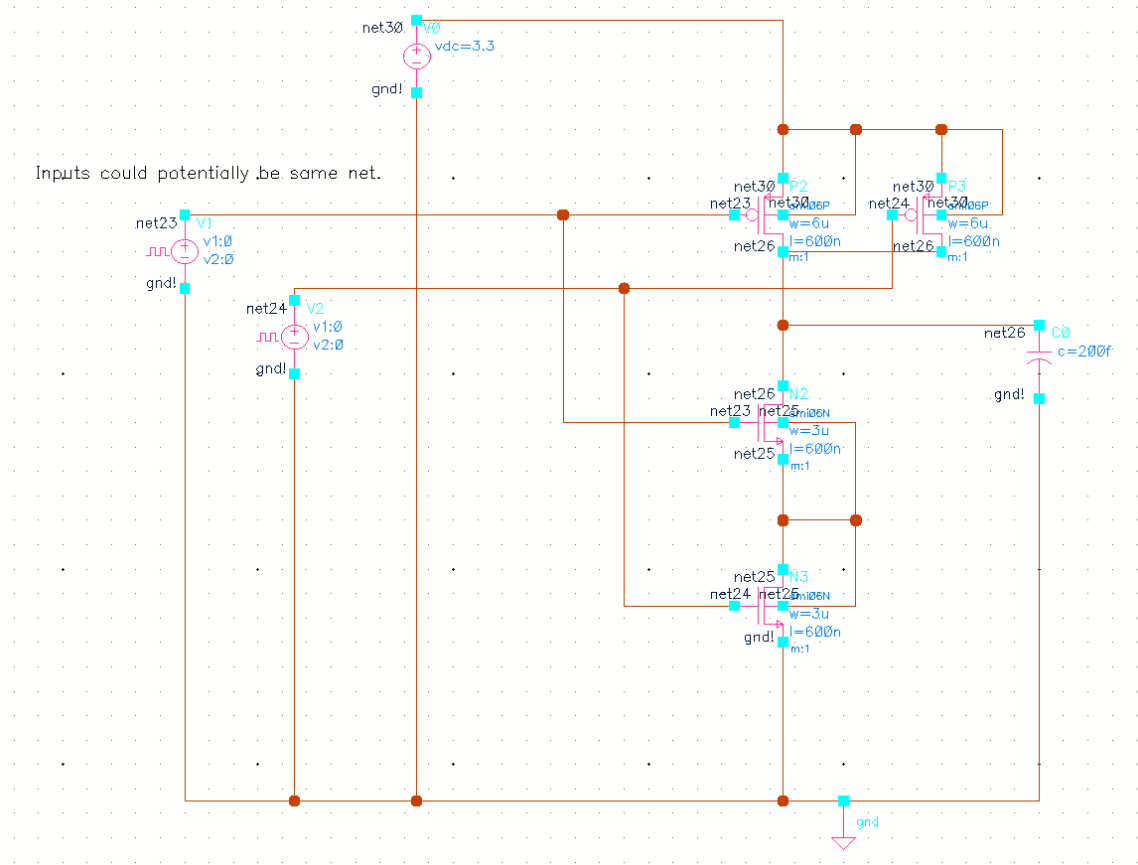


Figure 24: Transistor-level schematic of a NAND2 gate

Inputs could potentially be same net.

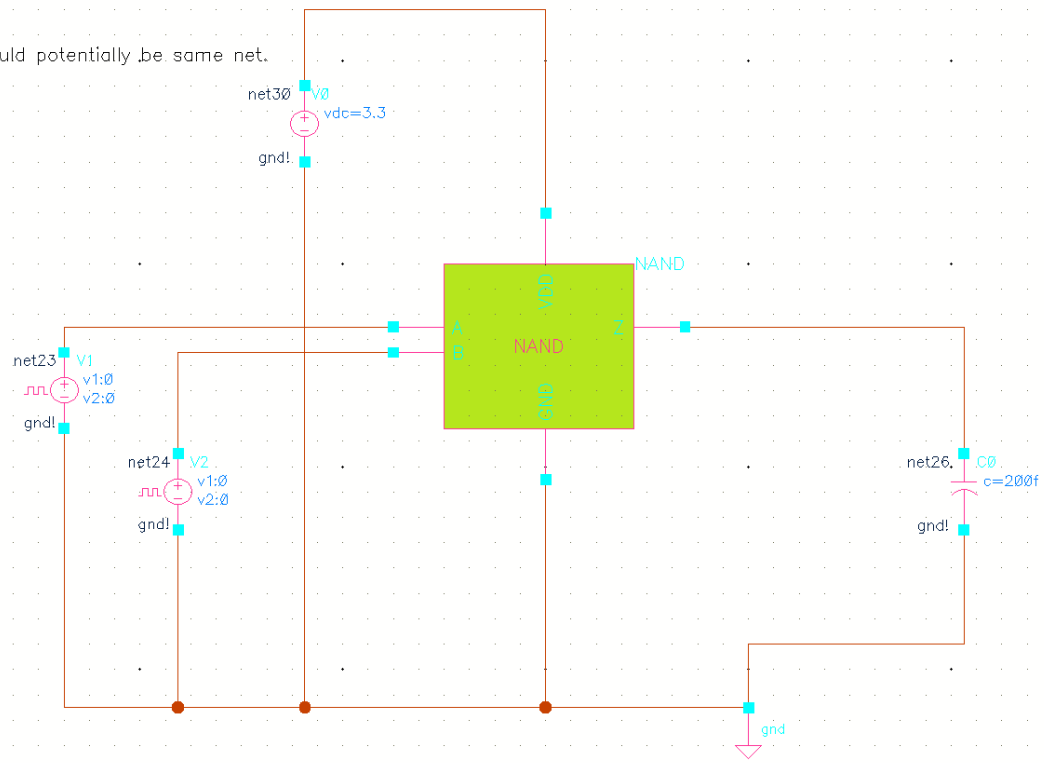
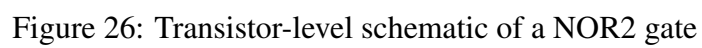


Figure 25: Gate-level schematic of a NAND2 gate

4.1.2.2 *NOR2 Gate.*

Just as NAND2 gates are identified by examining transistors connected to the inverters, NOR2 gates are found in a similar manner. NOR2 gates are included in the second abstraction level and are composed of two NMOS transistors in parallel and two PMOS transistors in series. Hence, NOR2 gates are identified by finding a PMOS transistor (P1) whose drain is connected to the source of the inverter's PMOS transistor (P0) and an NMOS transistor (N1) that shares a common source connection with the NMOS transistor of the inverter (N0), a common drain connection with N0, and a common gate with P1. Figure 26 depicts the transistor-level schematic of the NOR2 gate, and Figure 27 shows the equivalent gate-level schematic.



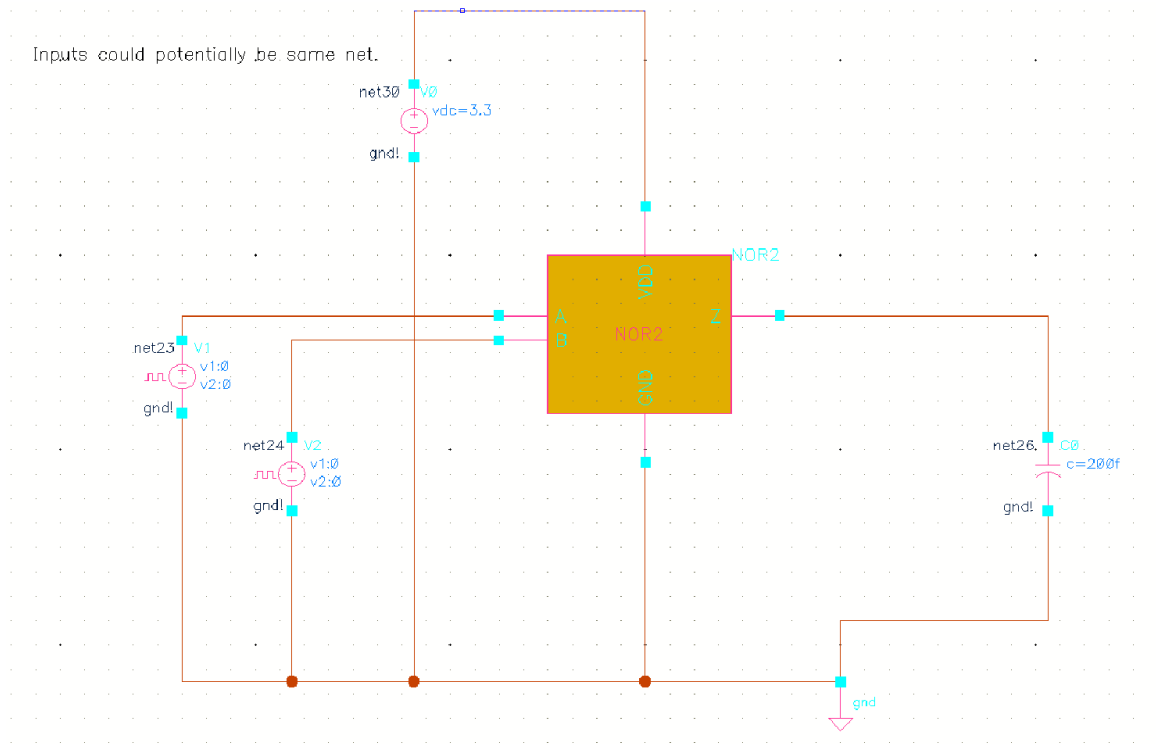


Figure 27: Gate-level schematic of a NOR2 gate

4.1.2.3 AND2 Gate.

AND2 gates are in the third abstraction level and are composed of a NAND2 gate and an inverter; therefore, the AND2 gates are identified by examining the connections between NAND2 gates and inverters. Specifically, the SCR code detects an AND2 gate by matching the output of a NAND2 gate (the drain of transistor N0) to the input of an inverter (the gate of the NMOS transistor of the inverter). Figure 28 depicts the transistor-level schematic of the AND2 gate, and Figure 29 shows the equivalent gate-level schematic.

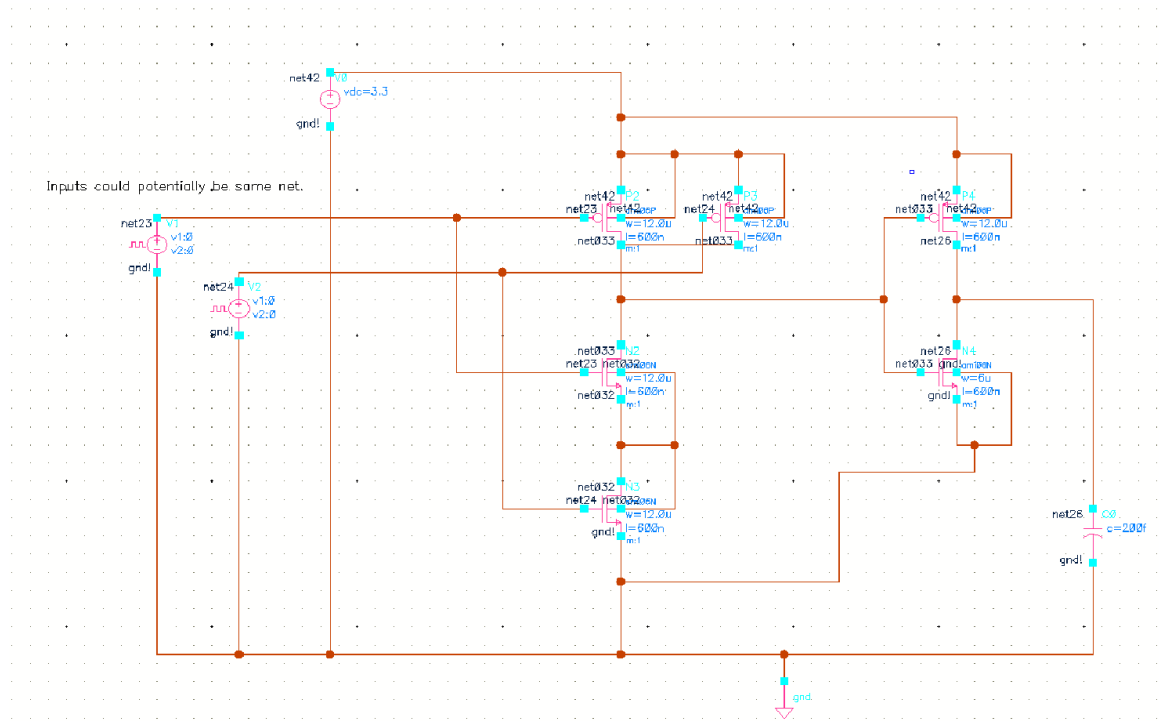


Figure 28: Transistor-level schematic of a AND2 gate

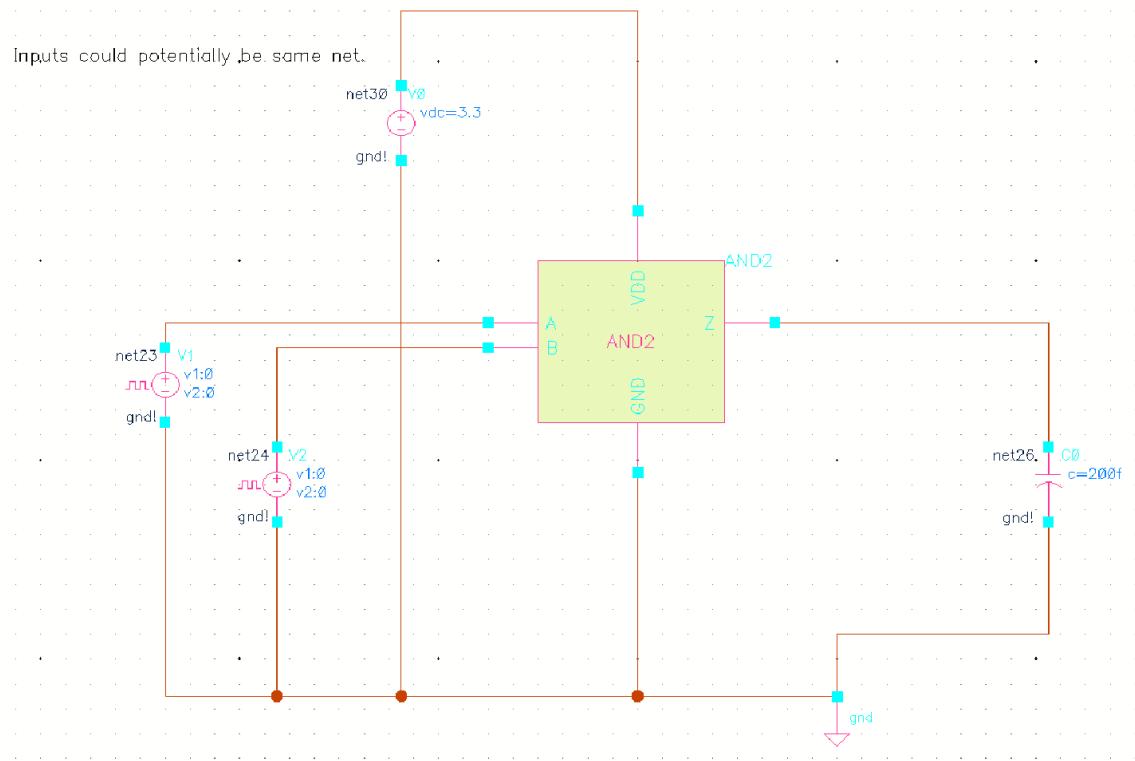


Figure 29: Gate-level schematic of a AND2 gate

4.1.2.4 OR2 Gate.

Similar to the AND2 gate, the OR2 gate is included in the third abstraction level and is composed of a NOR2 gate and an inverter. For this reason, OR2 gates are identified by matching the output of a NOR2 gate (the drain of transistor N0) to the input of an inverter (the gate of the NMOS transistor of the inverter). Figure 30 depicts the transistor-level schematic of the OR2 gate, and Figure 31 shows the equivalent gate-level schematic.

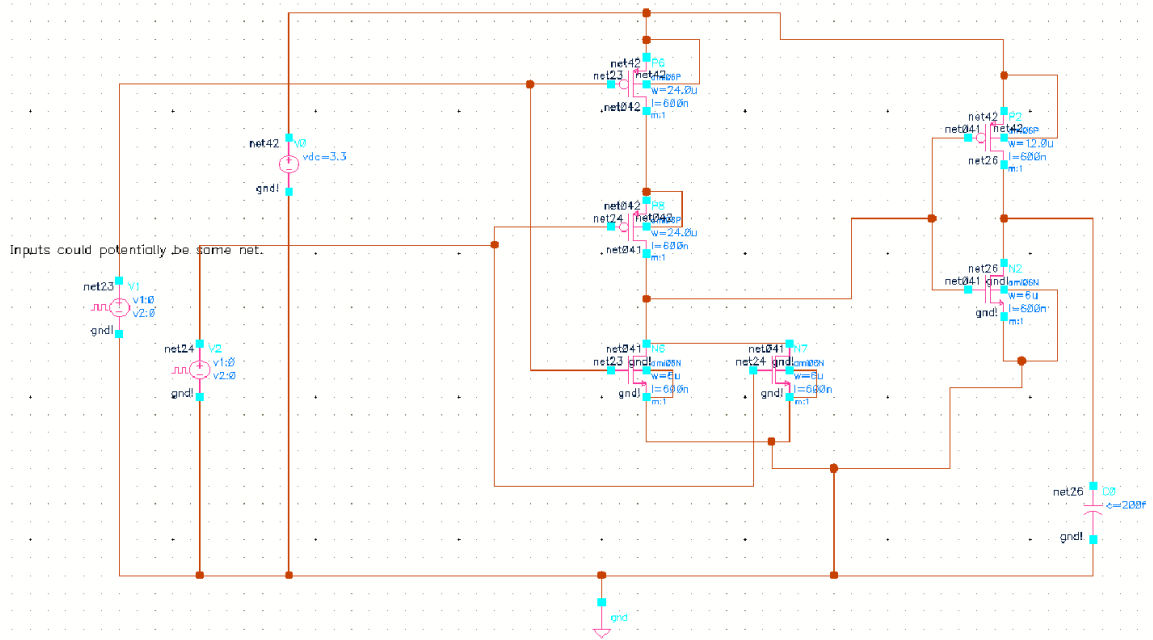


Figure 30: Transistor-level schematic of an OR2 gate

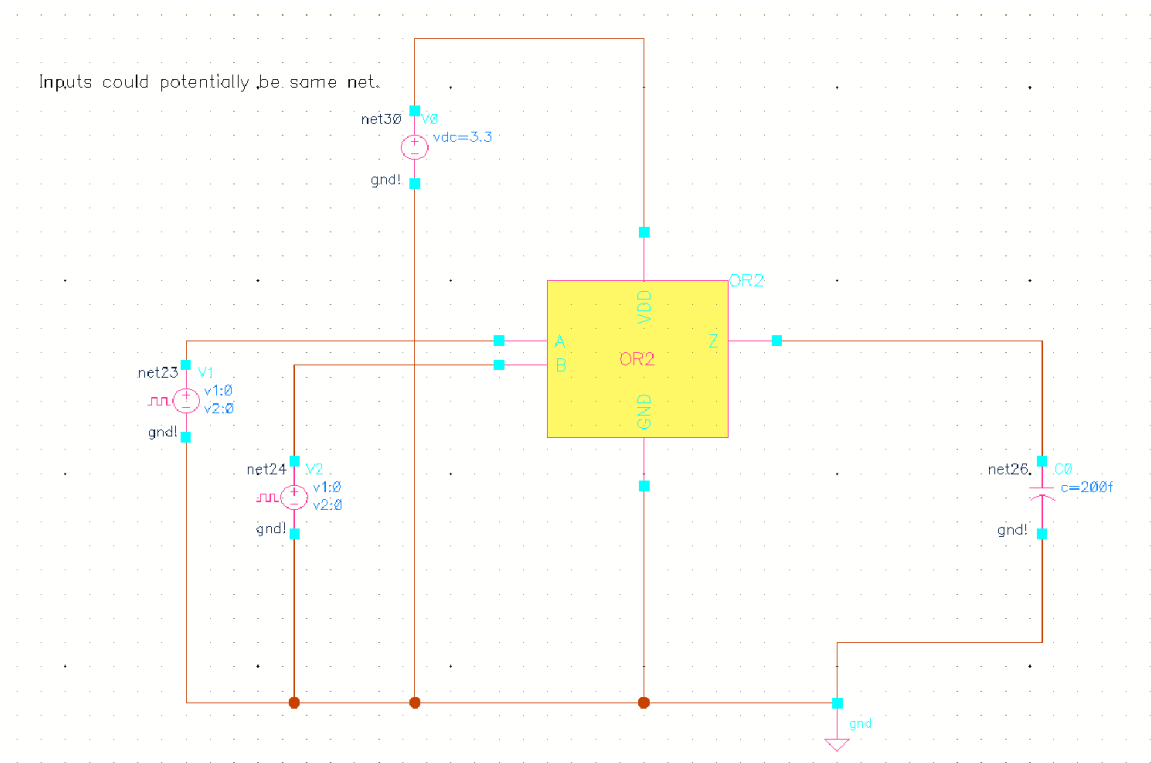


Figure 31: Gate-level schematic of a OR2 gate

4.1.2.5 NAND2b0 Gate.

The NAND2b0 gate is in the fourth abstraction level and is composed of an inverter and an OR2 gate. It is found by matching the output of an inverter (the drain of the NMOS transistor) to an input of an OR2 gate (the gate of either NMOS transistor of the NOR2 component). Figure 32 depicts the transistor-level schematic of the NAND2b0 gate, and Figure 33 shows the equivalent gate-level schematic.

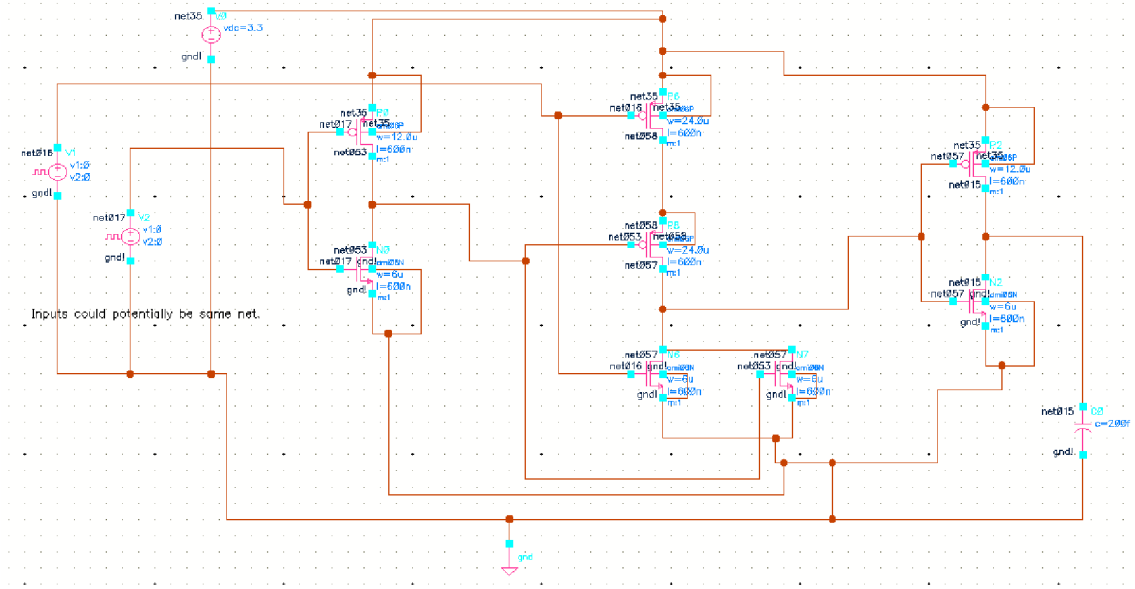


Figure 32: Transistor-level schematic of a NAND2b0 gate

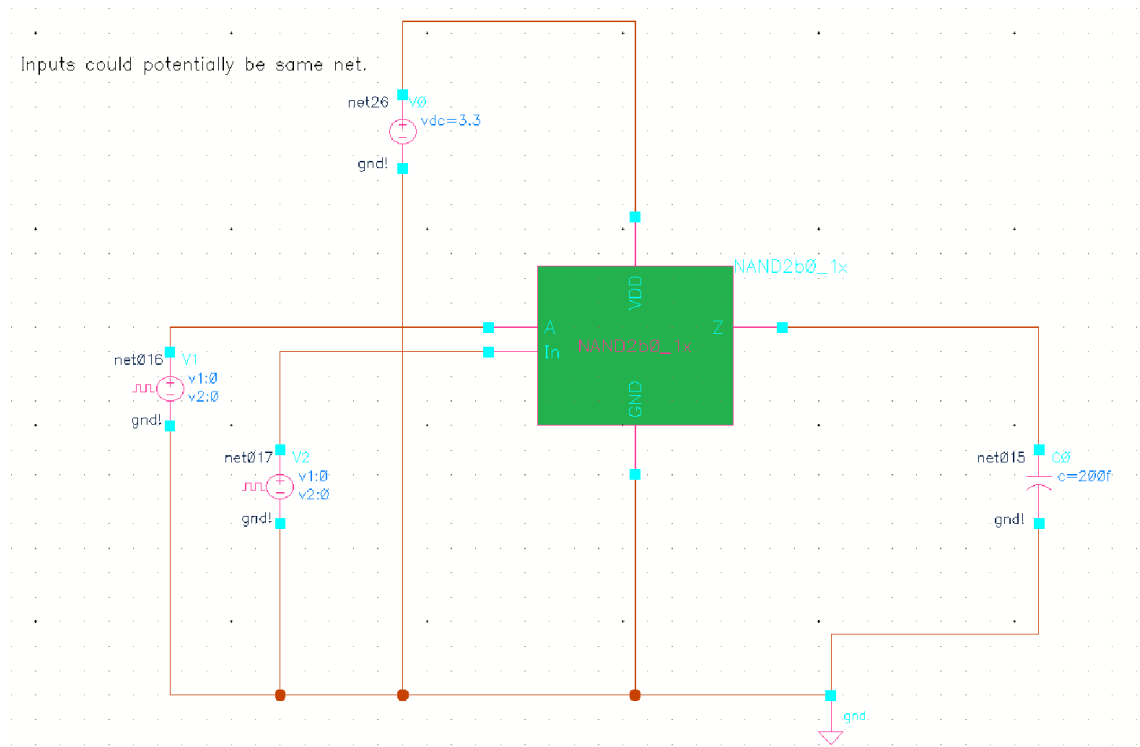


Figure 33: Gate-level schematic of a NAND2b0 gate

4.1.2.6 OAI21 Gate.

The OAI21 gate is included in the fourth abstraction level and is composed of an OR2 gate and a NAND2 gate. It is identified by matching the output of an OR2 gate (the drain of the NMOS transistor of the inverter component) to an input of a NAND2 gate (the gate of either NMOS transistor). Figure 34 depicts the transistor-level schematic of the OAI21 gate, and Figure 35 shows the equivalent gate-level schematic. Given that the output of the OAI21 gate comes from a NAND2 gate, there is a possibility for misidentification if the output is connected to an inverter that is a component of a complex gate. Techniques for avoiding this misidentification are discussed in Section 4.2.

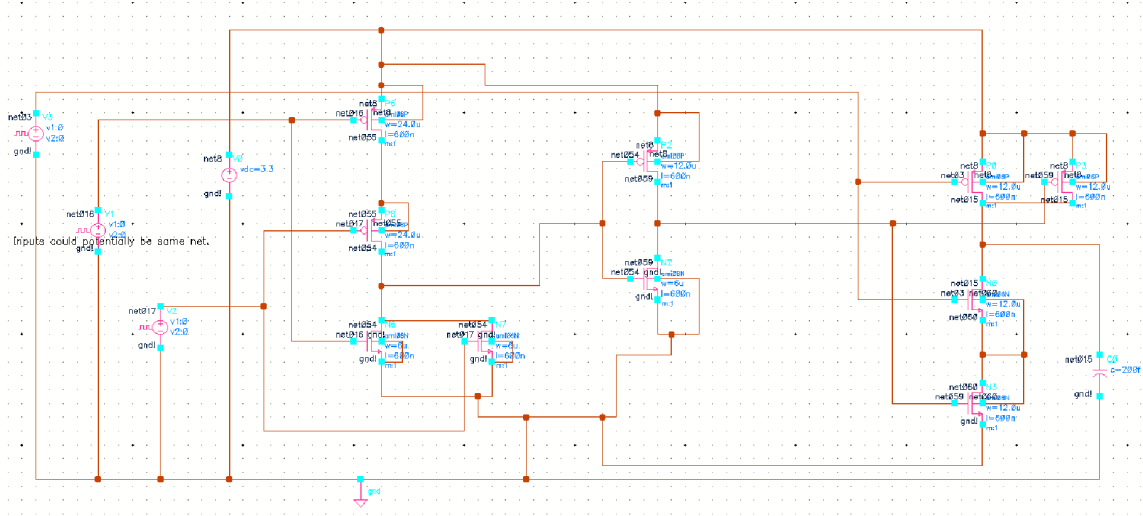


Figure 34: Transistor-level schematic of an OAI21 gate

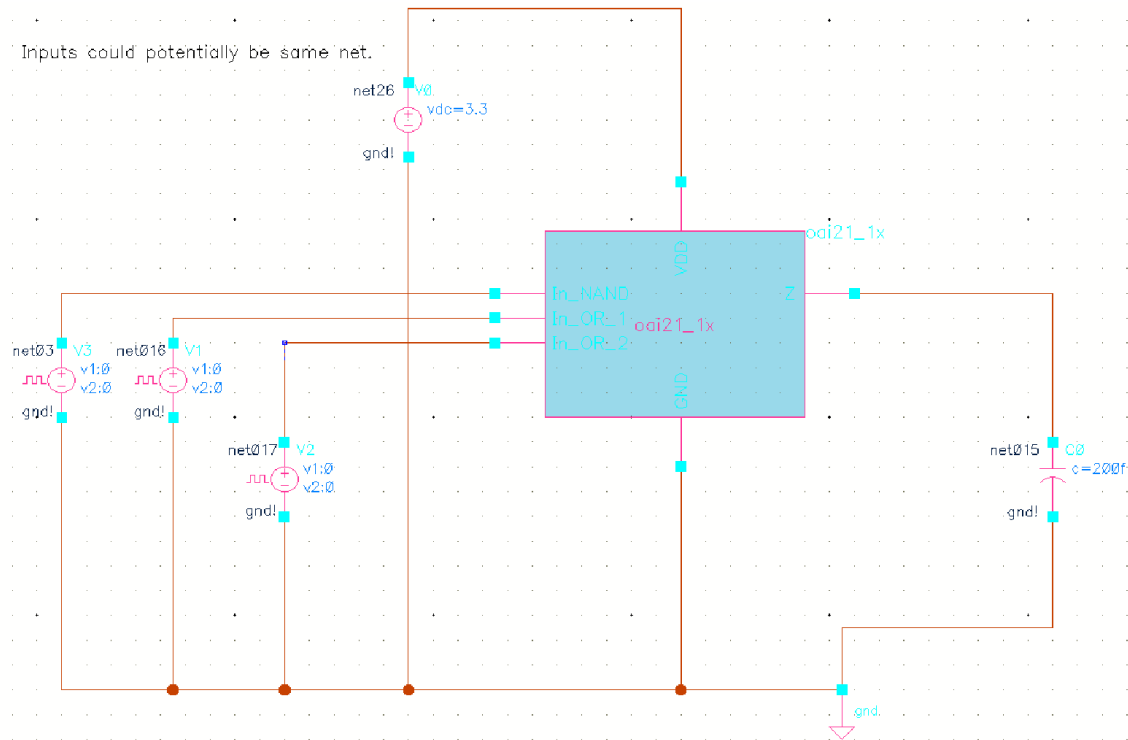


Figure 35: Gate-level schematic of an OAI21 gate

4.1.2.7 OAI21b1 Gate.

The OAI21b1 gate is included in the fourth abstraction level and consists of an inverter, OR2 gate, and NAND2 gate. Its composition makes this gate particularly difficult to identify given that it can also appear as either an inverter and OAI21 gate or a NAND2b0 and NAND2 gate. Hence, three separate approaches are taken to identify this gate.

The first approach searches for OAI21b1 gates that are composed of an inverter, OR2 gate, and NAND2 gate. It matches the output of an inverter (the drain of the NMOS transistor) to an input of an OR2 gate (the gate of either NMOS transistor of the NOR2 component), and the output of the OR2 gate (the drain of the NMOS transistor of the inverter component) to an input of a NAND2 gate (the gate of either NMOS transistor).

The second approach searches for an inverter and OAI21 gate. It matches the output of an inverter (the drain of the NMOS transistor) to the OR2 input of an OAI21 gate (the gate of either NMOS transistor of the NOR2 component).

The third approach searches for a NAND2b0 and NAND2 gate. It matches the output of a NAND2b0 gate (the NMOS transistor drain of the inverter part of the OR2 component) to an input of a NAND2 gate (the gate of either NMOS transistor). Figure 36 depicts the transistor-level schematic of the OAI21b1 gate, and Figure 37 shows the equivalent gate-level schematic.

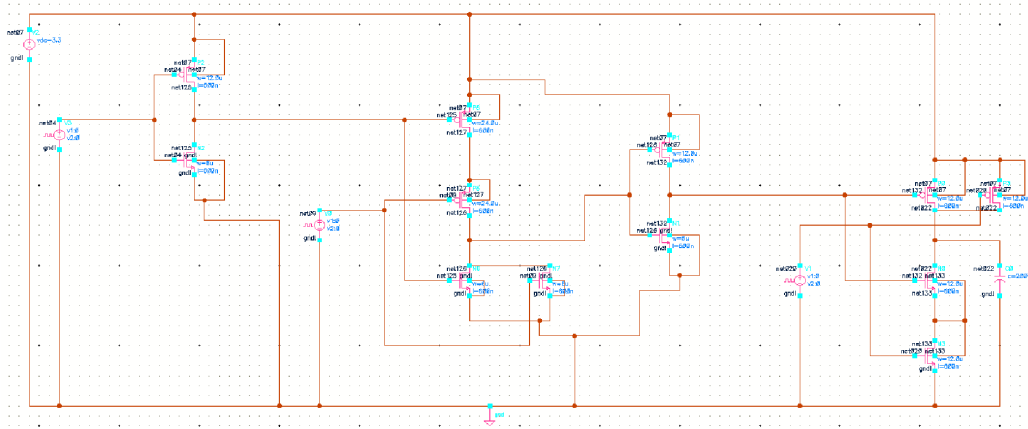


Figure 36: Transistor-level schematic of an OAI21b1 gate

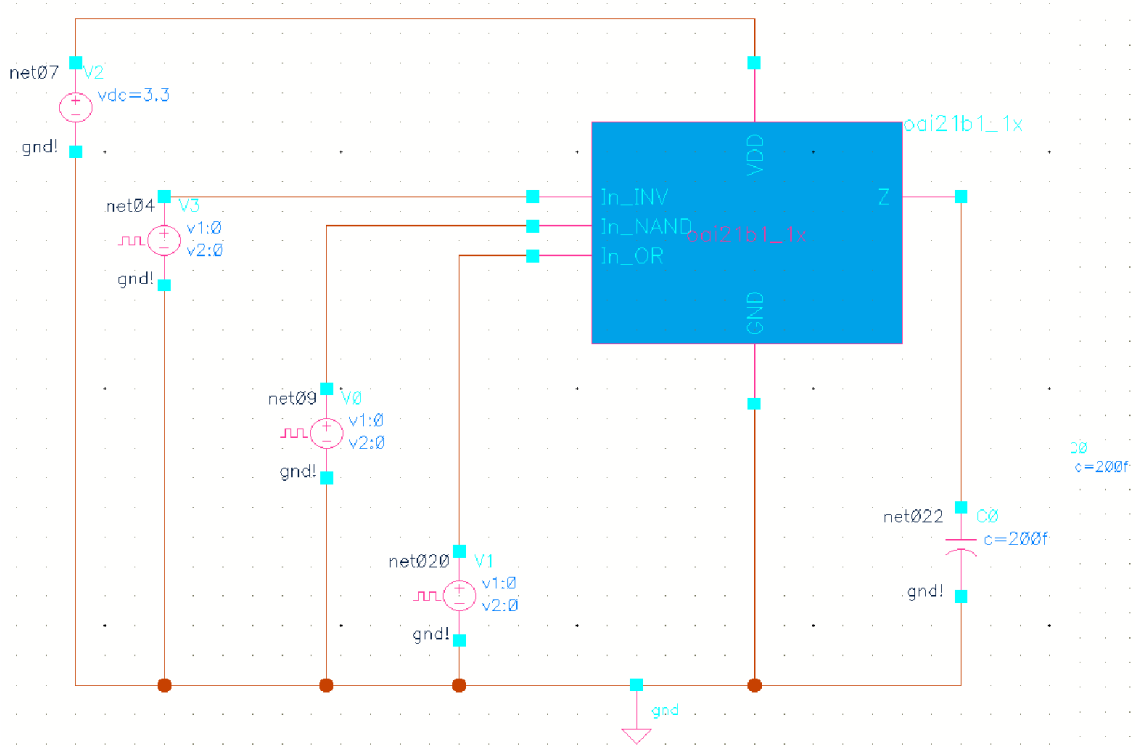


Figure 37: Gate-level schematic of an OAI21b1 gate

4.1.2.8 OAI21b0b1 Gate.

The OAI21b0b1 gate is included in the fourth abstraction level and consists of an inverter, AND2 gate, and an OR2 gate. Like the OAI21b1 gate, the composition of the OAI21b0b1 gate renders it difficult to identify as it can also appear as an AND2 and NAND2b0 gate. For this reason, there are two approaches to identifying this gate.

First, the output of an inverter (the drain of the NMOS transistor) is matched to one input of an OR2 gate (either NMOS transistor gate of the NOR2 component), and the output of an AND2 gate (the NMOS transistor drain of the inverter component) is matched to the other input of the OR2 gate.

Second, the output of an AND2 gate is matched to the OR2 input of a NAND2b0 gate (either NMOS transistor gate of the NOR2 component).

Figure 38 depicts the transistor-level schematic of the OAI21b0b1 gate, and Figure 39 shows the equivalent gate-level schematic. Given that an input of the OAI21b0b1 gate first enters an inverter, there is a possibility for misidentification if the input is connected to a NAND2 or NOR2 gate. Techniques for avoiding this misidentification are discussed in Section 4.2.

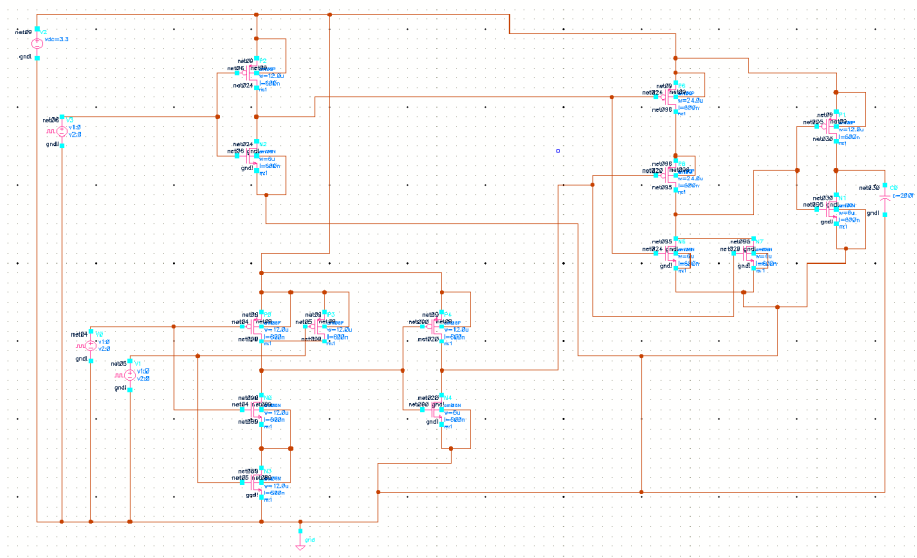


Figure 38: Transistor-level schematic of an OAI21b0b1 gate

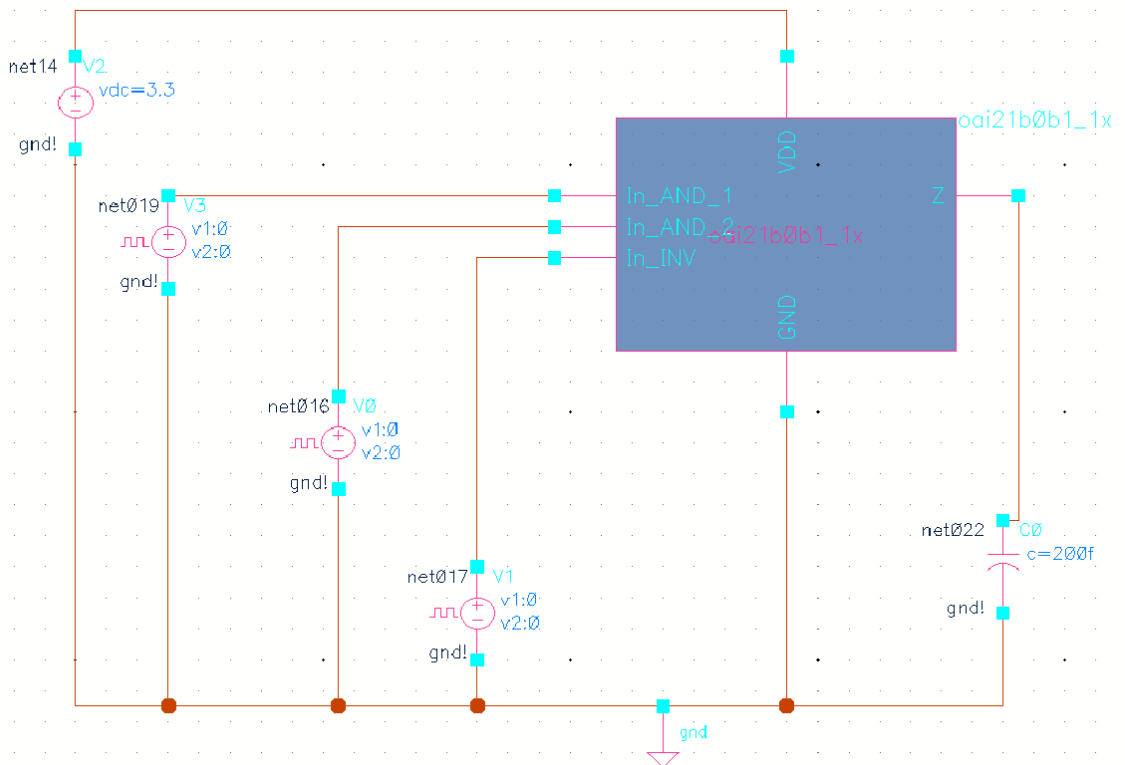


Figure 39: Gate-level schematic of an OAI21b0b1 gate

4.1.3 Phase 3 Results - Software Tool Application to Circuit D Equivalent.

For this section and Section 4.1.4, the results of the research will be represented by presenting the complexity of the circuit tested (transistor and gate count), the number of each type of gate utilized in the circuit, and two metrics to represent the accuracy of the SCR code: True Positive Rate (TPR) and False Positive Rate (FPR). While TPR and FPR are conceptually equivalent to P_D and P_{FA} , two metrics presented in Chapter 2, the terminology is shifted in this chapter to emphasize the contextual difference between the two sets of metrics. P_D and P_{FA} are used in this document to discuss the accuracy of verification methods; TPR and FPR are used in this document to discuss the gate-recognition accuracy of the SCR code. TPR represents the demonstrated rate of correctly identifying a gate that exists in the circuit, and FPR represents the demonstrated rate of falsely identifying a gate that does not exist in the circuit. Hence, ideal results include a TPR that approaches 1 and an FPR that approaches 0.

The results for the full adder cell tested in Phase 3 of the research is shown in Table 4. Perfect results were achieved for this phase such that for all types of gates included in this circuit, $TPR = 1$ and $FPR = 0$.

Table 4: Full adder test results

Transistor count	60
Gate count	9
Found inverter count / actual inverter count	2 / 2
Inverter TPR, FPR	$2/2 = 1.0$, $0/7 = 0.0$
Found NAND2 count / actual NAND2 count	3 / 3
NAND2 TPR, FPR	$3/3 = 1.0$, $0/6 = 0.0$
Found NAND2b0 count / actual NAND2b0 count	1 / 1
NAND2b0 TPR, FPR	$1/1 = 1.0$, $0/8 = 0.0$
Found OAI21 count / actual OAI21 count	1 / 1
OAI21 TPR, FPR	$1/1 = 1.0$, $0/8 = 0.0$
Found OAI21b1 count / actual OAI21b1 count	1 / 1
OAI21b1 TPR, FPR	$1/1 = 1.0$, $0/8 = 0.0$
Found OAI21b0b1 count / actual OAI21b0b1 count	1 / 1
OAI21b0b1 TPR, FPR	$1/1 = 1.0$, $0/8 = 0.0$

4.1.4 Phase 4 Results - SCR Code Analysis.

4.1.4.1 Level of Maturity.

To elaborate upon the topic concerning code maturity in Section 3.4, the SCR code maturity level is based upon recognition accuracy and comprehensiveness. Two aspects of recognition accuracy are presented: First, raw results of each of the circuit tests are presented in the same format as Table 4 from Section 4.1.3. Second, the data from the tables are re-arranged in order to depict overall recognition accuracy per type of gate. With regard to comprehensiveness, the percentage of cells in circuits and in standard cell libraries that the SCR code can currently identify is discussed.

Recognition Accuracy - Part 1 Overview. Since the test circuits used in this phase are limited to containing the nine gates recognized by the SCR algorithm, only five of the circuits perform a useful function: XOR gate, master/slave DFF, 2-to-1 MUX, digital comparator, and 4-bit adder. The remaining four circuits (Test1, Test2, Test3, and Test4) do not perform useful functions, but rather are random groups of gates intended to provide a variation in gate configuration and circuit complexity for the collection of test circuits. Furthermore, four of the circuits (XOR gate, master/slave DFF, 2-to-1 MUX, and 4-bit adder) are designed by a third party without visibility into the development of the SCR algorithm.

Recognition Accuracy - XOR Gate. Figure 40 and Figure 41 show the transistor-level schematic and the equivalent gate-level schematic, respectively, of the XOR gate tested to help determine the recognition accuracy of this research. This circuit was designed by 2d Lt Ralph K. Tatum. The results for the XOR gate are shown in Table 5. As seen from the table, perfect results ($TPR = 1$) were achieved for this circuit.

Recognition Accuracy - Master/Slave DFF. Figure 42 and Figure 43 show the transistor-level schematic and the equivalent gate-level schematic, respectively, of the

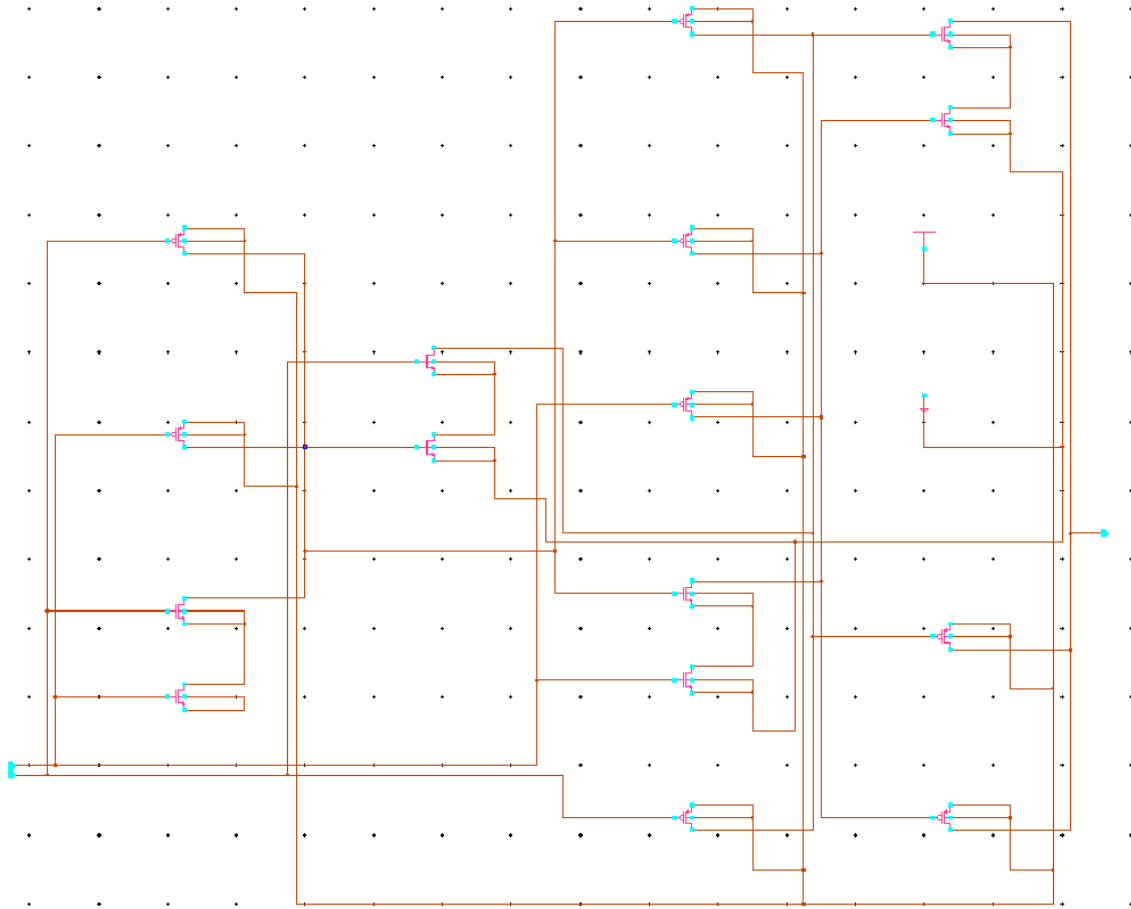


Figure 40: Transistor-level schematic of the XOR circuit tested

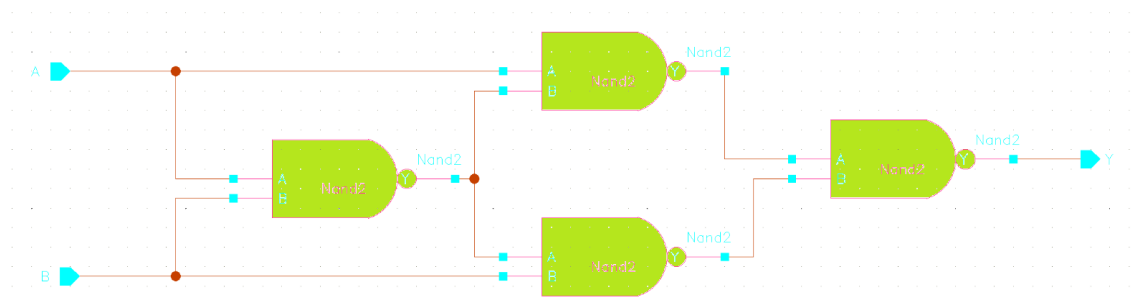


Figure 41: Gate-level representation of the XOR circuit tested

Table 5: XOR gate test results

Transistor count	16
Gate count	4
Found NAND2 count / actual NAND2 count	4 / 4
NAND2 TPR, FPR	$4/4 = 1.0$, -

master/slave DFF tested to determine the recognition accuracy of this research. The design for this circuit is inspired by the master/slave DFF displayed in [2]. The results for the master/slave DFF are shown in Table 6. As seen from the table, perfect results (TPR = 1 and FPR = 0) were achieved for this circuit. The achievement of perfect results for this circuit is significant because it indicates that the SCR algorithm and code has the capability to conduct SCR on feedback structures (discussed in Section 4.2.2).

Table 6: Master/Slave DFF test results

Transistor count	36
Gate count	10
Found inverter count / actual inverter count	2 / 2
Inverter TPR, FPR	$2/2 = 1.0$, $0/8 = 0.0$
Found NAND2 count / actual NAND2 count	8 / 8
NAND2 TPR, FPR	$8/8 = 1.0$, $0/2 = 0.0$

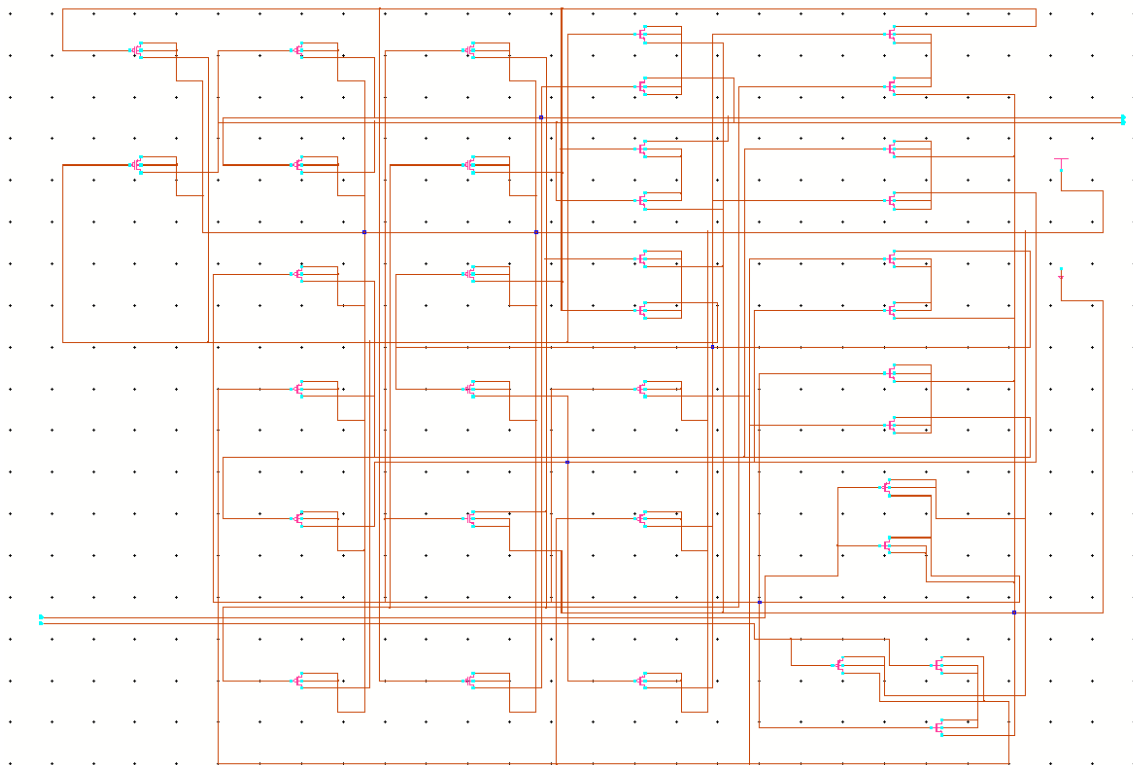


Figure 42: Transistor-level schematic of the Master/Slave DFF circuit tested

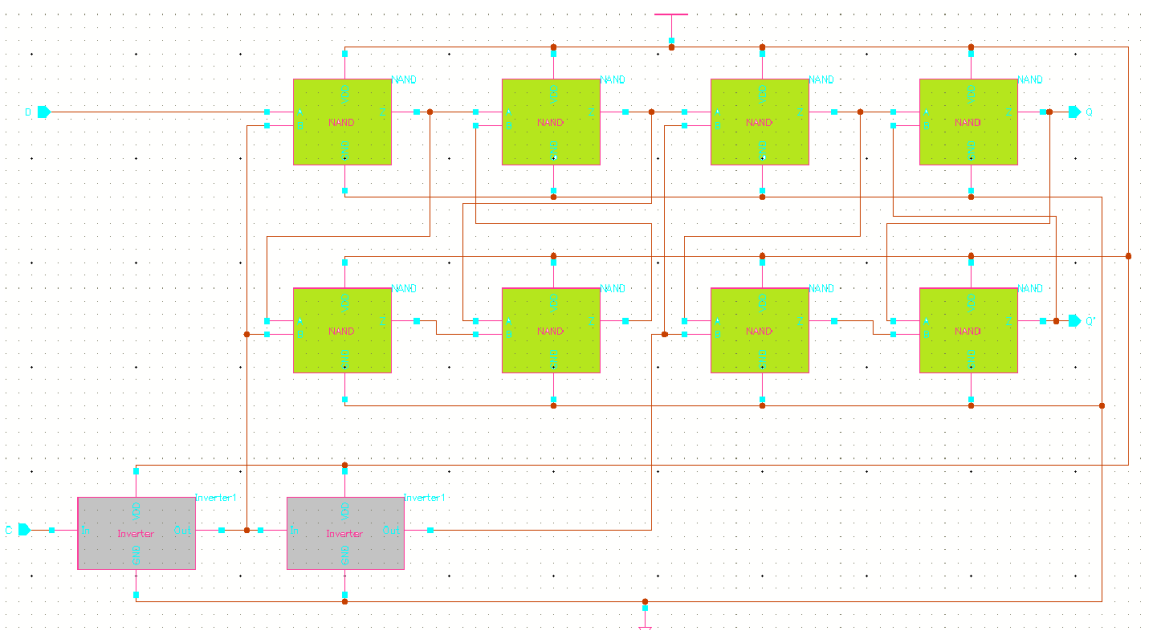


Figure 43: Gate-level representation of the Master/Slave DFF circuit tested

Recognition Accuracy - 2-to-1 MUX. Figure 44 and Figure 45 show the transistor-level schematic and the equivalent gate-level schematic, respectively, of the 2-to-1 MUX tested to determine the recognition accuracy of this research. The design for this circuit is inspired by the 2-to-1 MUX displayed in [3]. The SCR results for the 2-to-1 MUX are shown in Table 7. As seen from the table, perfect results ($TPR = 1$) were achieved for this circuit. The achievement of perfect results for this circuit is significant because it indicates that the SCR algorithm and code have the capability to conduct SCR on circuits that contain gates with the same input. For example, in the 2-to-1 MUX, the MUX input “S” is tied to both gate inputs of a NAND gate.

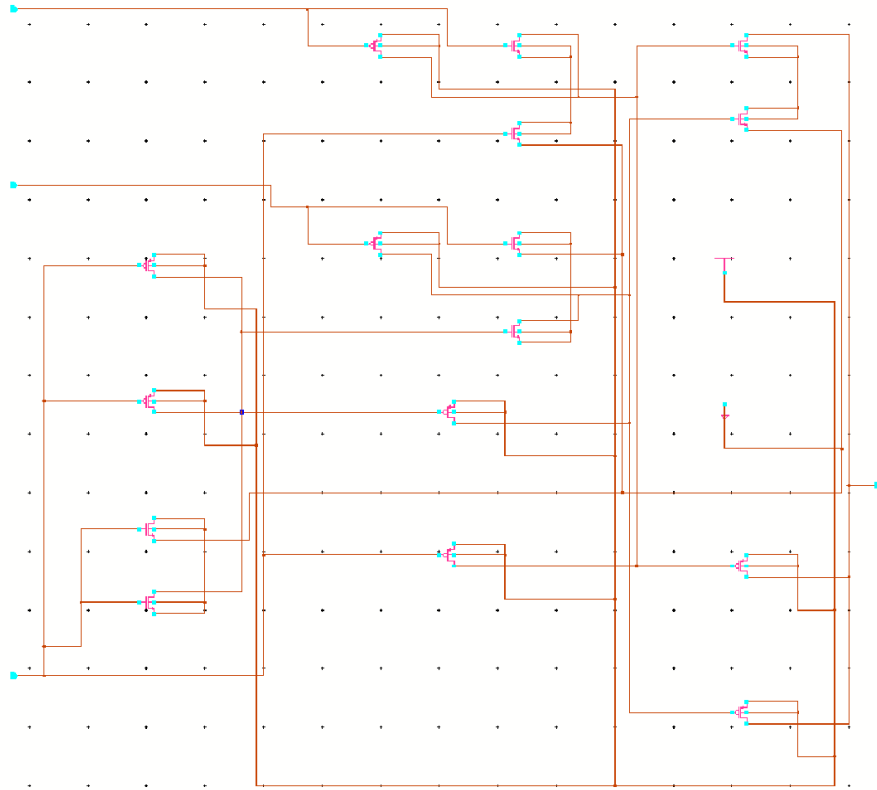


Figure 44: Transistor-level schematic of the 2-to-1 MUX circuit tested

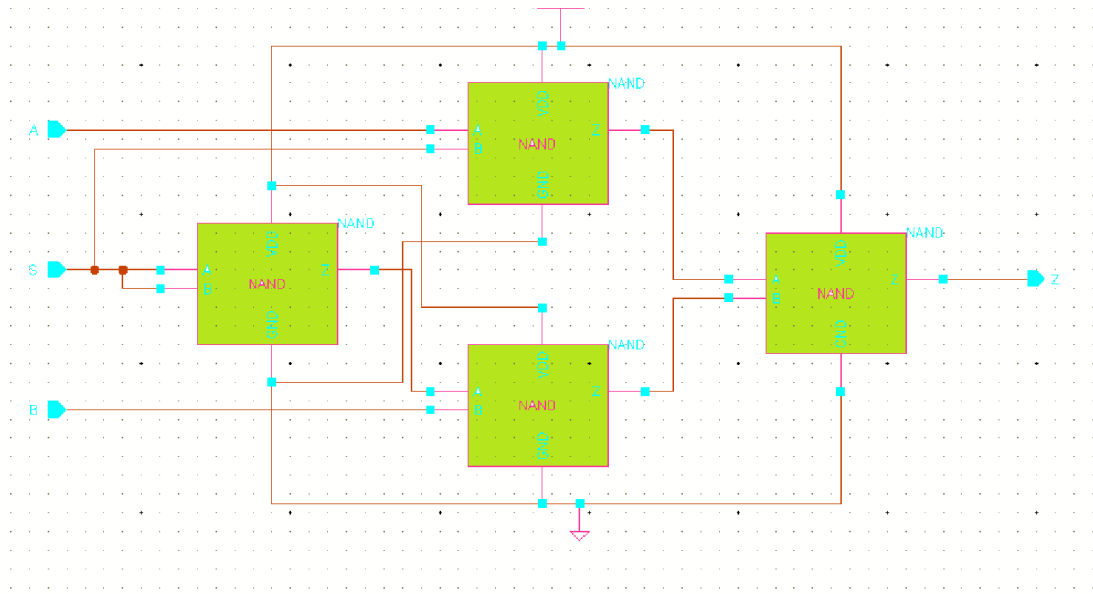


Figure 45: Gate-level representation of the 2-to-1 MUX circuit tested

Table 7: 2-to-1 MUX gate test results

Transistor count	16
Gate count	4
Found NAND2 count / actual NAND2 count	4 / 4
NAND2 TPR, FPR	$4/4 = 1.0$, -

Recognition Accuracy - Digital Comparator. Figure 46 and Figure 47 show the transistor-level schematic and the equivalent gate-level schematic, respectively, of the digital comparator tested to determine the recognition accuracy of this research. The circuit design is inspired by the digital comparator displayed in [1]. The results for the digital comparator are shown in Table 8. As seen from the table, perfect SCR results (TPR = 1 and FPR = 0) were achieved for this circuit. The perfect results lend support to the claim that the SCR algorithm and code can be successfully applied to simple circuits.

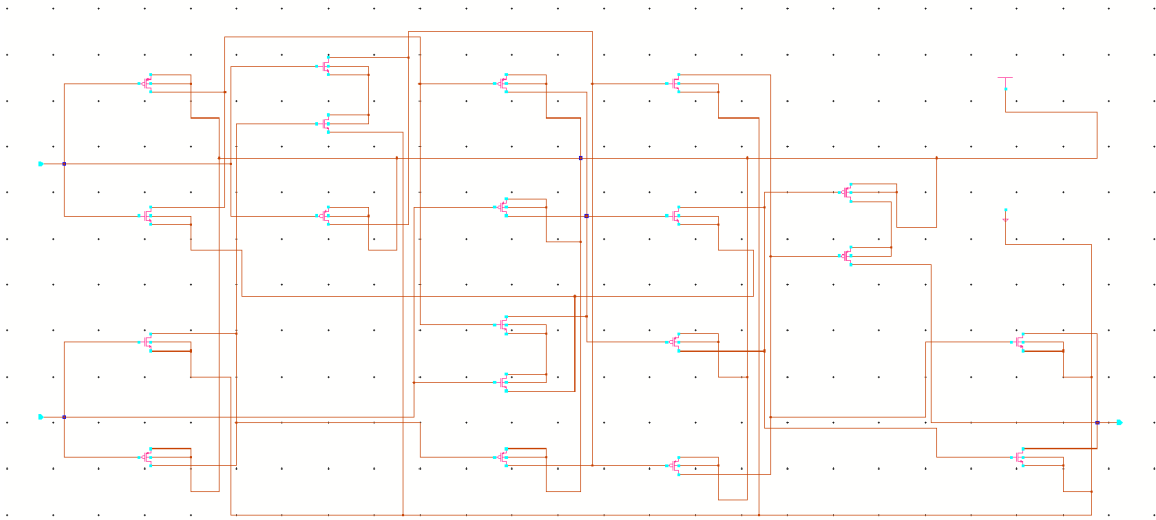


Figure 46: Transistor-level schematic of the Digital Comparator circuit tested

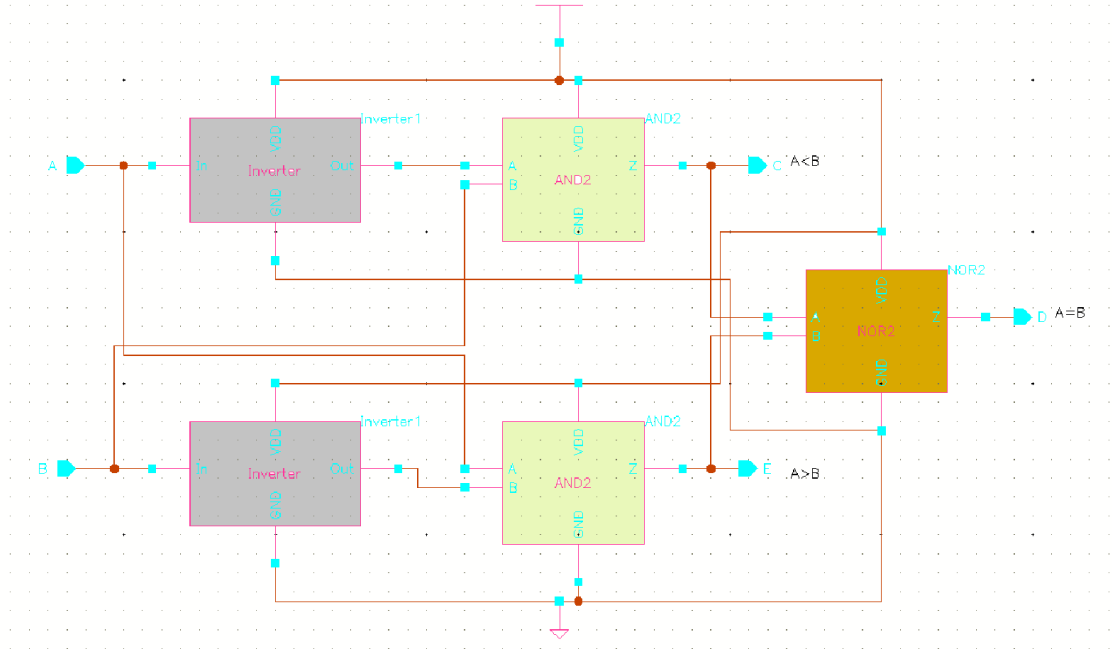


Figure 47: Gate-level representation of the Digital Comparator circuit tested

Table 8: Digital comparator test results

Transistor count	20
Gate count	5
Found inverter count / actual inverter count	2 / 2
Inverter TPR, FPR	$2/2 = 1.0$, $0/3 = 0.0$
Found NOR2 count / actual NOR2 count	1 / 1
NOR2 TPR, FPR	$1/1 = 1.0$, $0/4 = 0.0$
Found AND2 count / actual AND2 count	2 / 2
AND2 TPR, FPR	$2/2 = 1.0$, $0/3 = 0.0$

Recognition Accuracy - 4-bit Ripple Carry Adder. Figure 48 and Figure 49 show the transistor-level schematic and the equivalent gate-level schematic, respectively, of the 4-bit adder tested to determine the recognition accuracy of this research. This circuit was designed by connecting four of the full adder cells inherited from previous research (shown in Figures 15 through 18) to make a 4-bit adder. The results for the 4-bit ripple carry adder are shown in Table 9. As seen from the table, perfect results ($TPR = 1$ and $FPR = 0$) were achieved for this circuit. The achievement of perfect results for this circuit is significant because it indicates great promise for the success of the SCR code as it contains gates at the highest (fourth) level of abstraction and it is the most complex functional circuit.

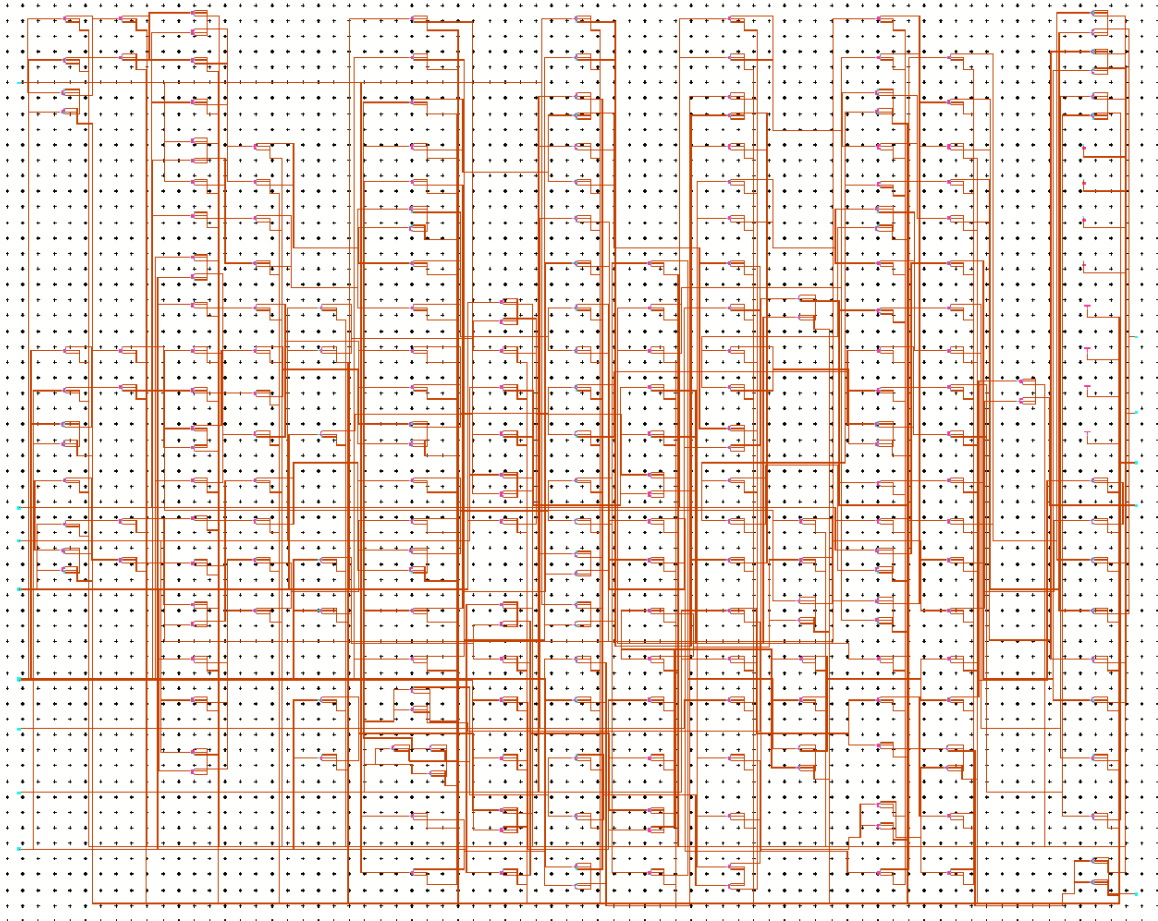


Figure 48: Transistor-level schematic of the 4-bit ripple carry adder circuit tested

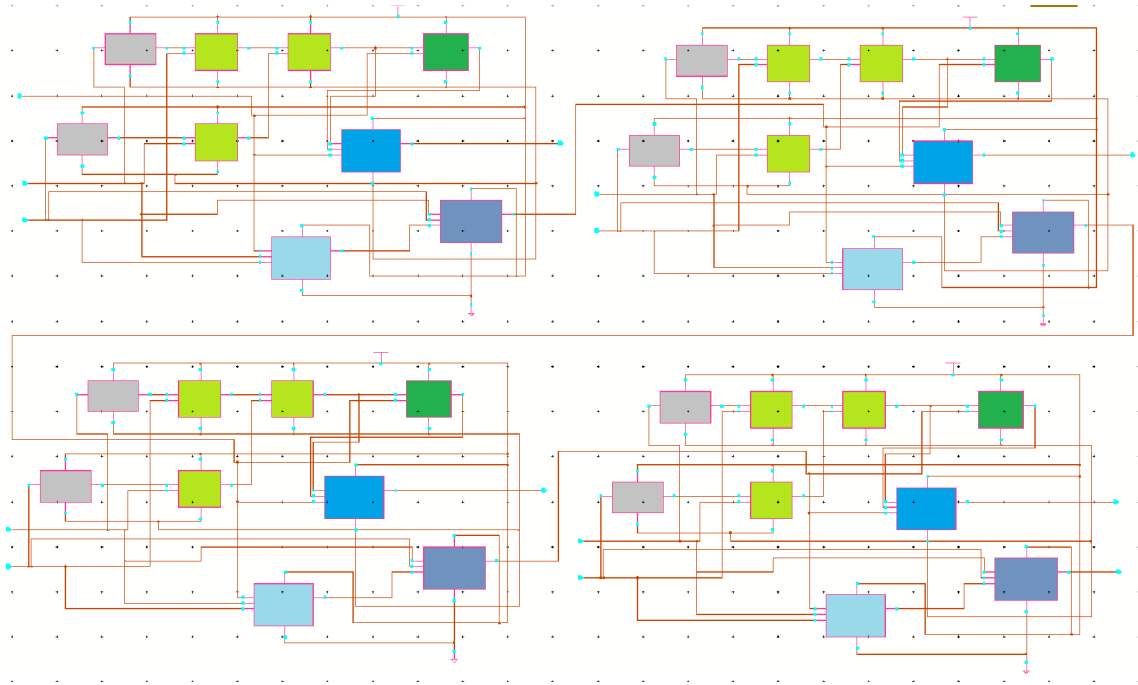


Figure 49: Gate-level representation of the 4-bit ripple carry adder adder circuit tested

Table 9: 4-bit ripple carry adder test results

Transistor count	240
Gate count	36
Found inverter count / actual inverter count	8 / 8
Inverter TPR, FPR	$8/8 = 1.0$, $0/28 = 0.0$
Found NAND2 count / actual NAND2 count	12 / 12
NAND2 TPR, FPR	$12/12 = 1.0$, $0/24 = 0.0$
Found NAND2b0 count / actual NAND2b0 count	4 / 4
NAND2b0 TPR, FPR	$4/4 = 1.0$, $0/32 = 0.0$
Found OAI21 count / actual OAI21 count	4 / 4
OAI21 TPR, FPR	$4/4 = 1.0$, $0/32 = 0.0$
Found OAI21b1 count / actual OAI21b1 count	4 / 4
OAI21b1 TPR, FPR	$4/4 = 1.0$, $0/32 = 0.0$
Found OAI21b0b1 count / actual OAI21b0b1 count	4 / 4
OAI21b0b1 TPR, FPR	$4/4 = 1.0$, $0/32 = 0.0$

Recognition Accuracy - Test1. Figure 50 and Figure 51 show the transistor-level schematic and the equivalent gate-level schematic, respectively, of the Test1 circuit tested to determine the recognition accuracy of this research. This circuit was designed by randomly selecting and connecting various types of gates. The results for Test1 are shown in Table 10. As seen from the table, perfect results were not achieved for this circuit. While the SCR algorithm identified two NAND2b0 gates, neither of the NAND2b0 gates described by the output netlist actually exist in the circuit. Rather, two other NAND2b0 gates exist in the circuit that the SCR algorithm failed to identify. As a result of the SCR algorithm's inability to accurately identify the NAND2b0 gates, the algorithm was also unable to accurately identify the inverters and NOR2 gates that exist in the circuit. This failure to achieve perfect results indicates the need for additional rules in the SCR algorithm code to guide the identification of complex gates. These rules are discussed in Section 4.2.2.

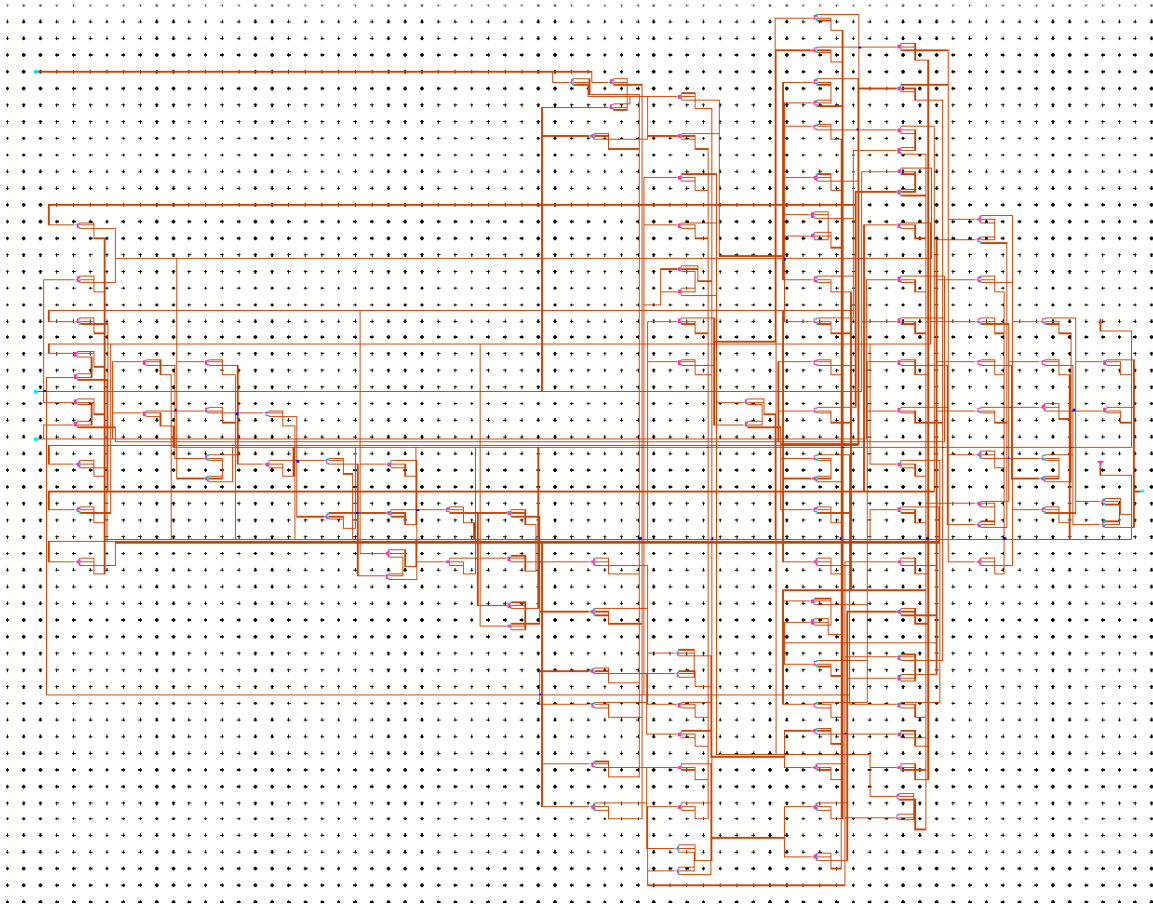


Figure 50: Transistor-level schematic of the Test1 circuit tested

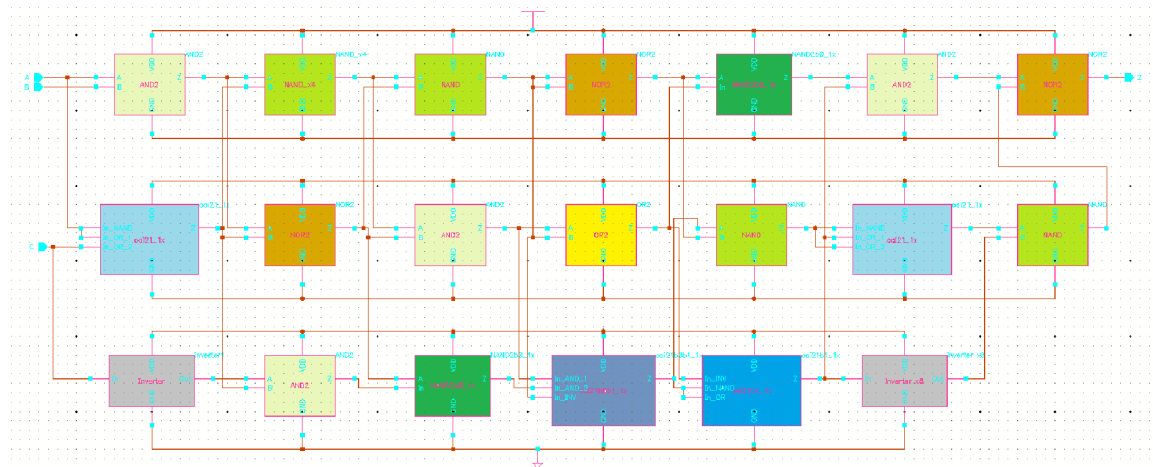


Figure 51: Gate-level representation of the Test1 circuit tested

Table 10: Test1 test results

Transistor count	124
Gate count	20
Found inverter count / actual inverter count	5 / 2
Inverter TPR, FPR	$2/2 = 1.0$, $3/19 = 0.16$
Found NAND2 count / actual NAND2 count	4 / 4
NAND2 TPR, FPR	$4/4 = 1.0$, $0/16 = 0.0$
Found NOR2 count / actual NOR2 count	1 / 3
NOR2 TPR, FPR	$1/3 = 0.33$, $0/17 = 0.0$
Found AND2 count / actual AND2 count	4 / 4
AND2 TPR, FPR	$4/4 = 1.0$, $0/16 = 0.0$
Found OR2 count / actual OR2 count	1 / 1
OR2 TPR, FPR	$1/1 = 1.0$, $0/19 = 0.0$
Found NAND2b0 count / actual NAND2b0 count	2 / 2
NAND2b0 TPR, FPR	$0/2 = 0.0$, $2/18 = 0.11$
Found OAI21 count / actual OAI21 count	2 / 2
OAI21 TPR, FPR	$2/2 = 1.0$, $0/18 = 0.0$
Found OAI21b1 count / actual OAI21b1 count	1 / 1
OAI21b1 TPR, FPR	$1/1 = 1.0$, $0/19 = 0.0$
Found OAI21b0b1 count / actual OAI21b0b1 count	1 / 1
OAI21b0b1 TPR, FPR	$1/1 = 1.0$, $0/19 = 0.0$

Recognition Accuracy - Test2. Figure 52 and Figure 53 show the transistor-level schematic and the equivalent gate-level schematic, respectively, of the Test2 circuit tested to determine the recognition accuracy of this research. Like Test1, this circuit was designed by randomly selecting and connecting various types of gates. The results for Test2 are shown in Table 11. As seen from the table, perfect results ($TPR = 1$ and $FPR = 0$) were achieved for this circuit.

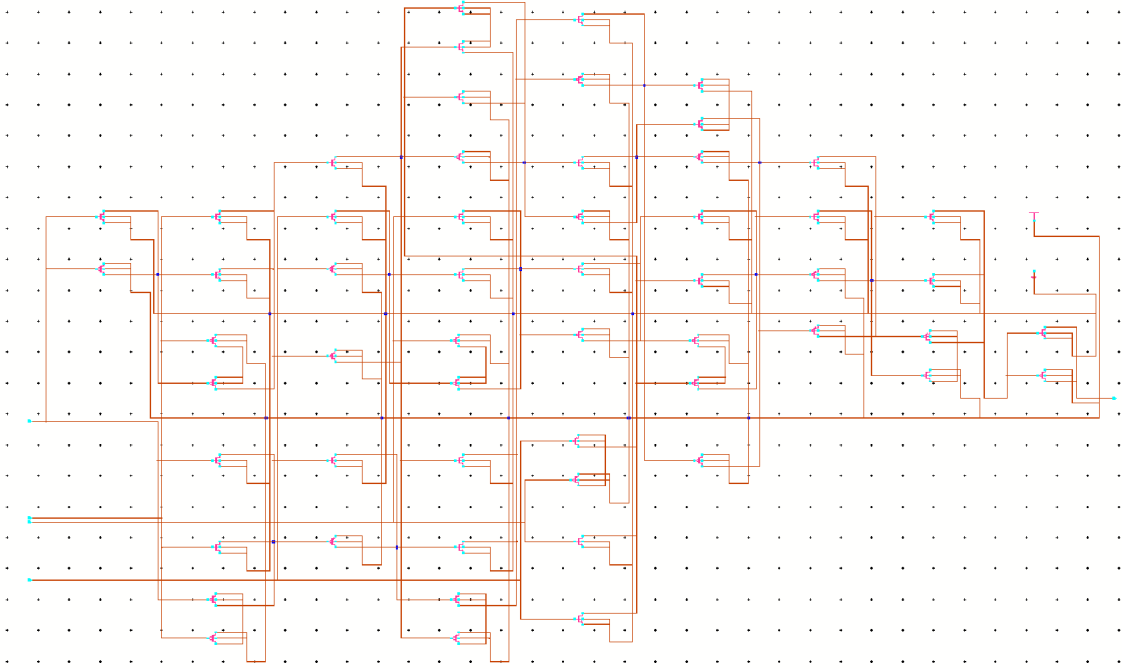


Figure 52: Transistor-level schematic of the Test2 circuit tested

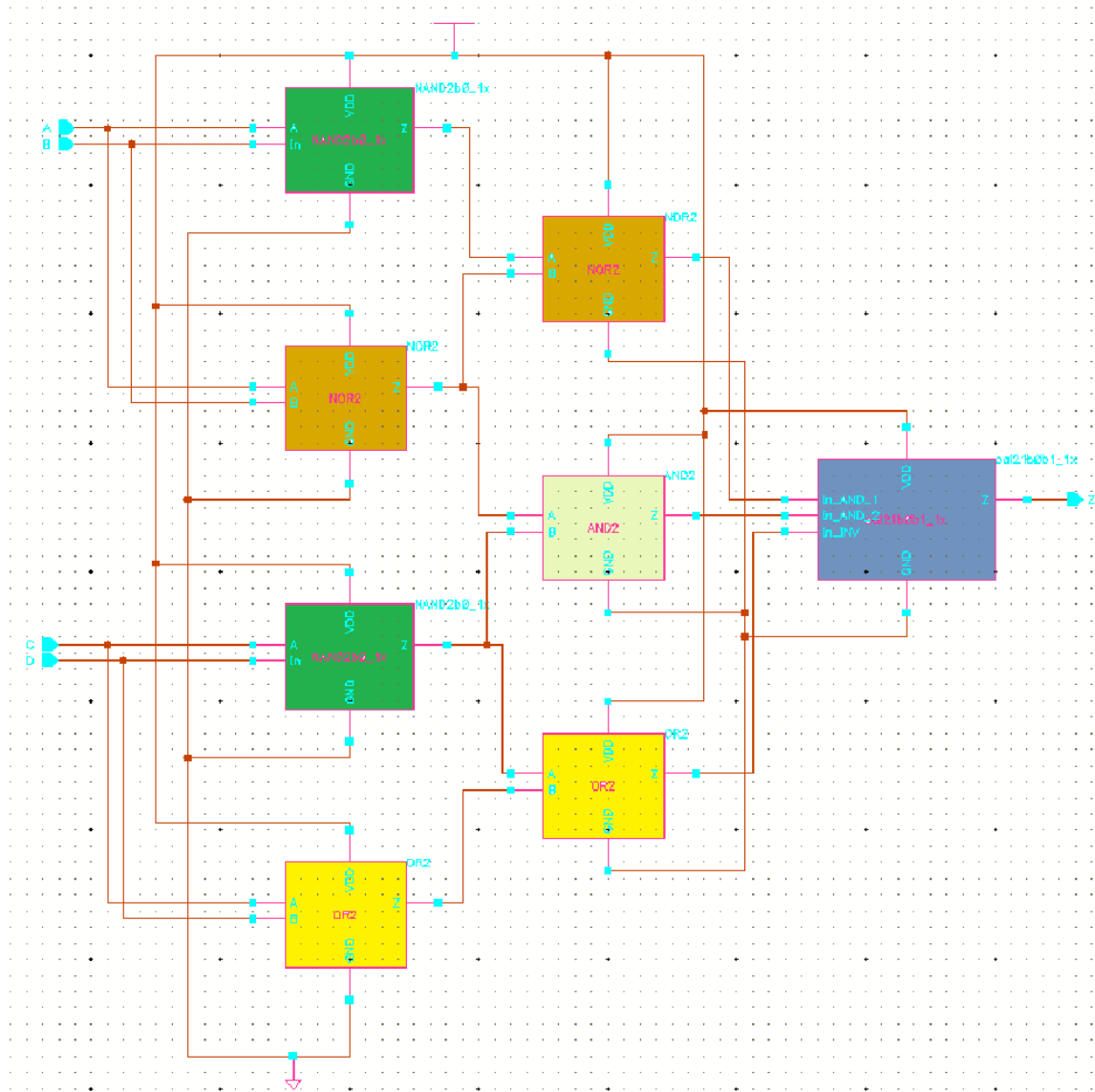


Figure 53: Gate-level representation of the Test2 circuit tested

Table 11: Test2 test results

Transistor count	54
Gate count	8
Found NOR2 count / actual NOR2 count	2 / 2
NOR2 TPR, FPR	$2/2 = 1.0$, $0/6 = 0.0$
Found AND2 count / actual AND2 count	1 / 1
AND2 TPR, FPR	$1/1 = 1.0$, $0/7 = 0.0$
Found OR2 count / actual OR2 count	2 / 2
OR2 TPR, FPR	$2/2 = 1.0$, $0/6 = 0.0$
Found NAND2b0 count / actual NAND2b0 count	2 / 2
NAND2b0 TPR, FPR	$2/2 = 1.0$, $0/6 = 0.0$
Found OAI21b0b1 count / actual OAI21b0b1 count	1 / 1
OAI21b0b1 TPR, FPR	$1/1 = 1.0$, $0/7 = 0.0$

Recognition Accuracy - Test3. Figure 54 and Figure 55 show the transistor-level schematic and the equivalent gate-level schematic, respectively, of the Test3 circuit tested to determine the recognition accuracy of this research. Like Test1 and Test2, this circuit was designed by randomly selecting and connecting various types of gates. The results for Test3 are shown in Table 12. As seen from the table, perfect results (TPR = 1 and FPR = 0) were achieved for this circuit.

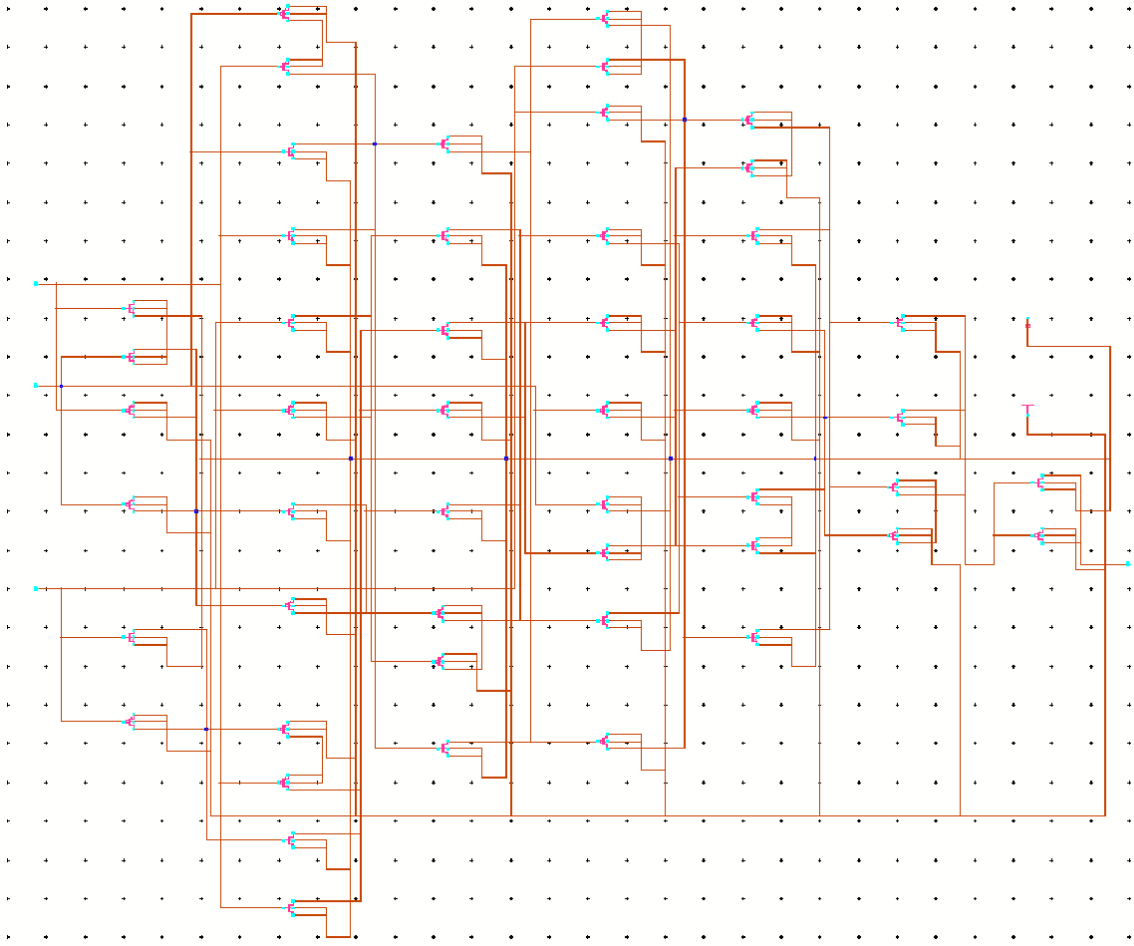


Figure 54: Transistor-level schematic of the Test3 circuit tested

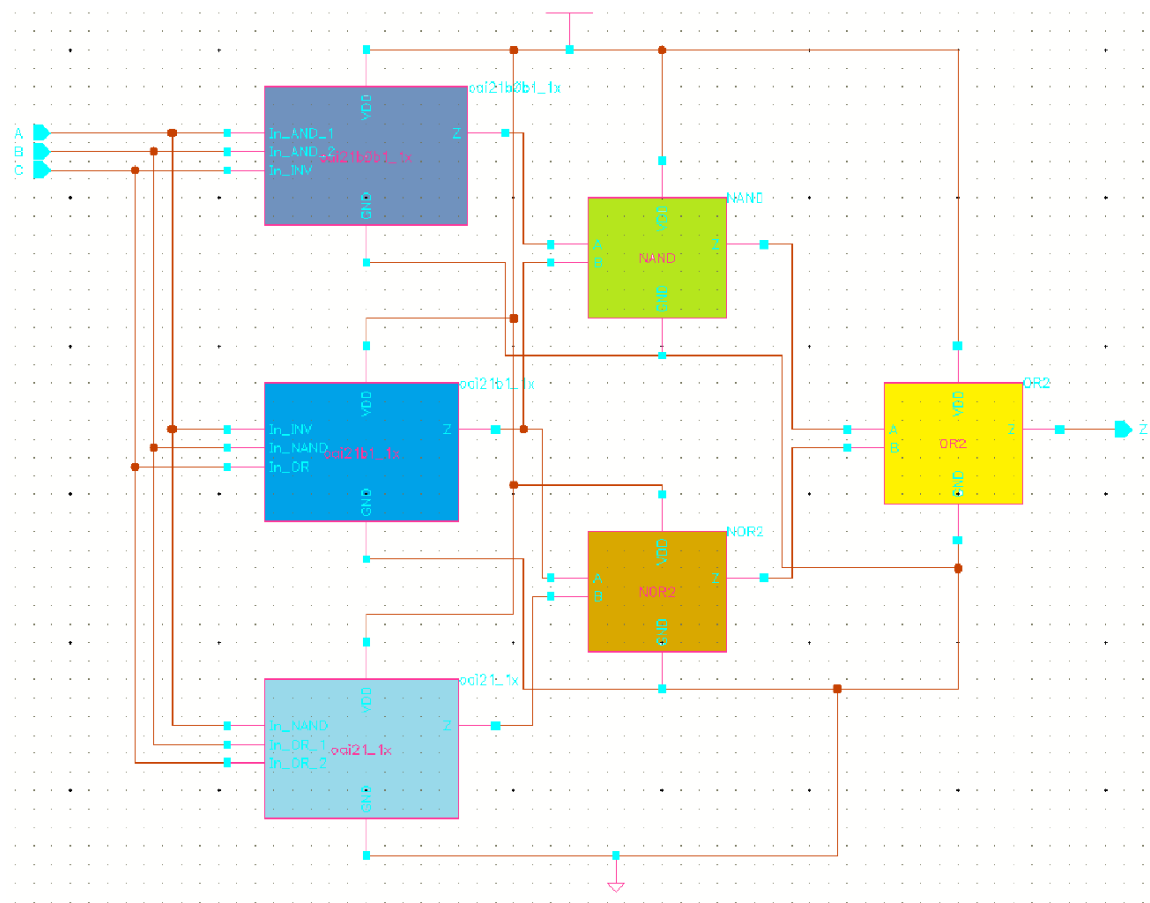


Figure 55: Gate-level representation of the Test3 circuit tested

Table 12: Test3 test results

Transistor count	50
Gate count	6
Found NAND2 count / actual NAND2 count	1 / 1
NAND2 TPR, FPR	1/1 = 1.0, 0/5 = 0.0
Found NOR2 count / actual NOR2 count	1 / 1
NOR2 TPR, FPR	1/1 = 1.0, 0/5 = 0.0
Found OR2 count / actual OR2 count	1 / 1
OR2 TPR, FPR	1/1 = 1.0, 0/5 = 0.0
Found OAI21 count / actual OAI21 count	1 / 1
OAI21 TPR, FPR	1/1 = 1.0, 0/5 = 0.0
Found OAI21b1 count / actual OAI21b1 count	1 / 1
OAI21b1 TPR, FPR	1/1 = 1.0, 0/5 = 0.0
Found OAI21b0b1 count / actual OAI21b0b1 count	1 / 1
OAI21b0b1 TPR, FPR	1/1 = 1.0, 0/5 = 0.0

Recognition Accuracy - Test4. Figure 56 and Figure 57 show the transistor-level schematic and the equivalent gate-level schematic, respectively, of the Test4 circuit tested to determine the recognition accuracy of this research. This circuit was designed connecting thirteen instances of Test3. The results for Test4 are shown in Table 13. The achievement of perfect results for this circuit is significant because it is the most complex circuit tested overall, thereby lending promise to the application of the SCR algorithm and code to large-scale circuits.

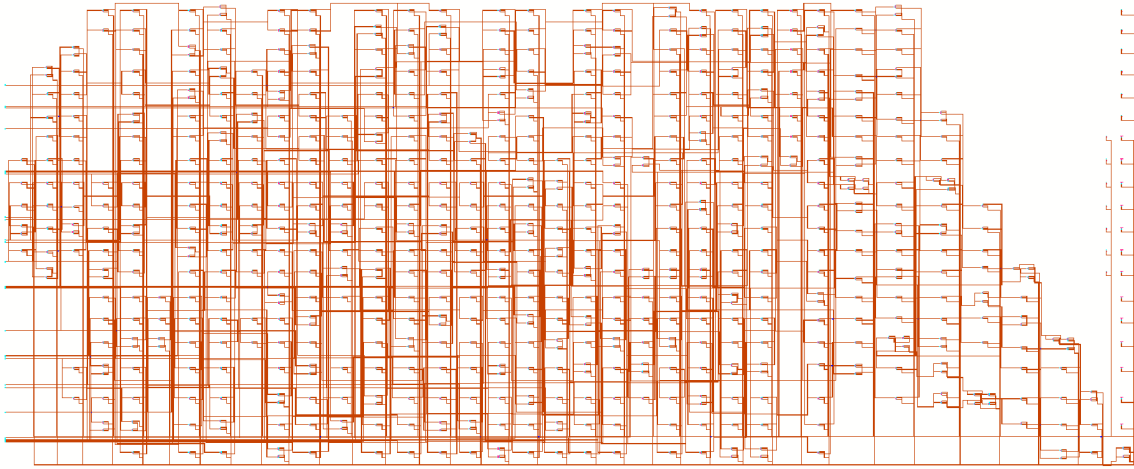


Figure 56: Transistor-level schematic of the Test4 circuit tested

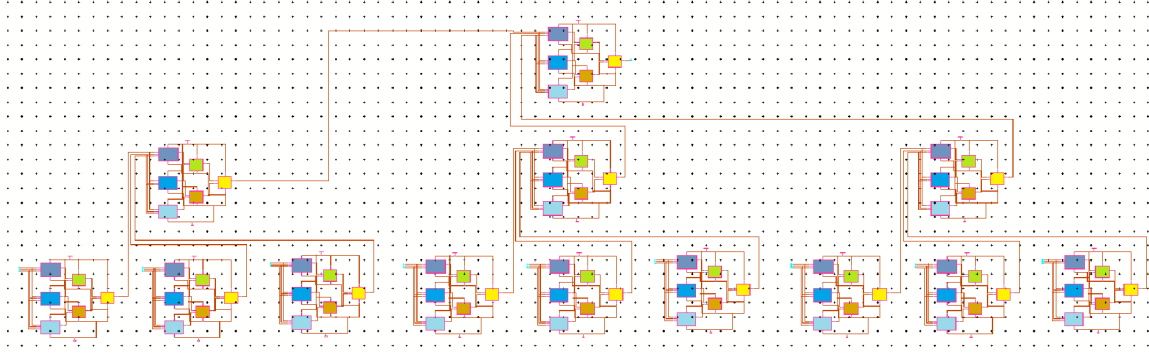


Figure 57: Gate-level representation of the Test4 circuit tested

Table 13: Test4 test results

Transistor count	650
Gate count	78
Found NAND2 count / actual NAND2 count	13 / 13
NAND2 TPR, FPR	13/13 = 1.0, 0/65 = 0.0
Found NOR2 count / actual NOR2 count	13 / 13
NOR2 TPR, FPR	13/13 = 1.0, 0/65 = 0.0
Found OR2 count / actual OR2 count	13 / 13
OR2 TPR, FPR	13/13 = 1.0, 0/65 = 0.0
Found OAI21 count / actual OAI21 count	13 / 13
OAI21 TPR, FPR	13/13 = 1.0, 0/65 = 0.0
Found OAI21b1 count / actual OAI21b1 count	13 / 13
OAI21b1 TPR, FPR	13/13 = 1.0, 0/65 = 0.0
Found OAI21b0b1 count / actual OAI21b0b1 count	13 / 13
OAI21b0b1 TPR, FPR	13/13 = 1.0, 0/65 = 0.0

Recognition Accuracy - Part 1 Summary.

Perfect SCR results were achieved for all test circuits except for Test1, which resulted in $FPR = 3/19 = 0.16$ for inverters, $TPR = 1/3 = 0.33$ for NOR2 gates, and $FPR = 2/18 = 0.11$ for NAND2b0 gates. For all other gates in Test1, the metrics achieved were $TPR = 1.0$ and $FPR = 0.0$. The SCR algorithm's inability to accurately identify NAND2b0 gates caused the inaccurate identification of the inverters and NOR2 gates in the circuit, which indicates the need for additional rules in the algorithm to guide the identification of complex gates.

Of the ten circuits tested, nine produced perfect SCR results. Most notably, successful results of the 4-bit adder and Test4 indicate greatest promise for the success of the SCR algorithm and code. Both circuits include gates at the highest (fourth) level of abstraction. The 4-bit adder is the most complex functional circuit, and Test4 is the most complex circuit overall.

Recognition Accuracy - Part 2 Overview.

Whereas Recognition Accuracy - Part 1 presented the test results achieved per type of circuit, this section presents the test results achieved per type of gate. This section is intended to show that there is no relationship between TPR, FPR, and circuit complexity for each type of gate. The absence of a relationship indicates that the recognition accuracy is not related to circuit complexity as one might expect. Instead, recognition accuracy is related to the thoroughness of the gate-recognition rules in the algorithm, as discussed in Section 4.2.2.

Recognition Accuracy - Inverter. Table 14 shows the TPR and FPR test results for the inverter in the order of increasing circuit complexity. Given that the TPR remains 1.0 for all values of FPR and all levels of circuit complexity, the results shown in the table indicate no correlation between TPR, FPR, or circuit complexity.

Table 14: Inverter results

Test Circuit Name	Transistor Count	# Inverters in Circuit	# Inverters Found	TPR, FPR
Digital comparator	20	2	2	$2/2 = 1.0$, $0/3 = 0.0$
Master/slave DFF	36	2	2	$2/2 = 1.0$, $0/8 = 0.0$
Full adder cell	60	2	2	$2/2 = 1.0$, $0/7 = 0.0$
Test1	124	2	5	$2/2 = 1.0$, $3/19 = 0.16$
4-bit adder	240	8	8	$8/8 = 1.0$, $0/28 = 0.0$

Recognition Accuracy - NAND2. Table 15 shows the TPR and FPR test results for the NAND2 in the order of increasing circuit complexity. Given that the TPR remains 1.0 and FPR remains 0.0 for all levels of circuit complexity, the results shown in the table indicate no correlation between TPR, FPR, or circuit complexity.

Recognition Accuracy - NOR2. Table 16 shows the TPR and FPR test results for the NOR2 gate in the order of increasing circuit complexity. Given that FPR remains 0.0 for all values of TPR and levels of circuit complexity, the results shown in the table indicate no correlation between TPR, FPR, or circuit complexity.

Table 15: NAND2 results

Test Circuit Name	Transistor Count	# NAND2s in Circuit	# NAND2s Found	TPR / FPR
XOR gate	16	4	4	4/4 = 1.0, -
2-to-1 MUX	16	4	4	4/4 = 1.0, -
Master/slave DFF	36	8	8	8/8 = 1.0, 0/2 = 0.0
Test3	50	1	1	1/1 = 1.0, 0/5 = 0.0
Full adder cell	60	3	3	3/3 = 1.0, 0/6 = 0.0
Test1	124	4	4	4/4 = 1.0, 0/16 = 0.0
4-bit adder	240	12	12	12/12 = 1.0, 0/24 = 0.0
Test4	650	13	13	13/13 = 1.0, 0/65 = 0.0

Table 16: NOR2 results

Test Circuit Name	Transistor Count	# NOR2s in Circuit	# NOR2s Found	TPR / FPR
Digital comparator	20	1	1	1/1 = 1.0, 0/4 = 0.0
Test3	50	1	1	1/1 = 1.0, 0/5 = 0.0
Test2	54	2	2	2/2 = 1.0, 0/6 = 0.0
Test1	124	3	1	1/3 = 0.33, 0/17 = 0.0
Test4	650	13	13	13/13 = 1.0, 0/65 = 0.0

Recognition Accuracy - AND2. Table 17 shows the TPR and FPR test results for the AND2 gate in the order of increasing circuit complexity. Given that TPR remains 1.0 and FPR remains 0.0 for all levels of circuit complexity, the results shown in the table indicate no correlation between TPR, FPR, or circuit complexity.

Table 17: AND2 results

Test Circuit Name	Transistor Count	# AND2s in Circuit	# AND2s Found	TPR / FPR
Digital comparator	20	2	2	$2/2 = 1.0$, $0/3 = 0.0$
Test2	54	1	1	$1/1 = 1.0$, $0/7 = 0.0$
Test1	124	4	4	$4/4 = 1.0$, $0/16 = 0.0$

Recognition Accuracy - OR2. Table 18 shows the TPR and FPR test results for the OR2 gate in the order of increasing circuit complexity. Given that TPR remains 1.0 and FPR remains 0.0 for all levels of circuit complexity, the results shown in the table indicate no correlation between TPR, FPR, or circuit complexity.

Table 18: OR2 results

Test Circuit Name	Transistor Count	# OR2s in Circuit	# OR2s Found	TPR / FPR
Test3	50	1	1	$1/1 = 1.0$, $0/5 = 0.0$
Test2	54	2	2	$2/2 = 1.0$, $0/6 = 0.0$
Test1	124	1	1	$1/1 = 1.0$, $0/19 = 0.0$
Test4	650	13	13	$13/13 = 1.0$, $0/65 = 0.0$

Recognition Accuracy - NAND2b0. Table 18 shows the TPR and FPR test results for the OR2 gate in the order of increasing circuit complexity. Given that TPR remains 1.0 and FPR remains 0.0 for all levels of circuit complexity except for the outlying results corresponding to Test1, the results shown in the table indicate no correlation between TPR, FPR, or circuit complexity.

Table 19: NAND2b0 results

Test Circuit Name	Transistor Count	# NAND2b0s in Circuit	# NAND2b0s Found	TPR / FPR
Test2	54	2	2	2/2 = 1.0, 0/6 = 0.0
Full adder cell	60	1	1	1/1 = 1.0, 0/8 = 0.0
Test1	124	2	2	0/2 = 0.0, 2/18 = 0.11
4-bit adder	240	4	4	4/4 = 1.0, 0/32 = 0.0

Recognition Accuracy - OAI21. Table 20 shows the TPR and FPR test results for the OAI21 gate in the order of increasing circuit complexity. Given that TPR remains 1.0 and FPR remains 0.0 for all levels of circuit complexity, the results shown in the table indicate no correlation between between TPR, FPR, or circuit complexity.

Recognition Accuracy - OAI21b1. Table 21 shows the TPR and FPR test results for the OAI21b1 gate in the order of increasing circuit complexity. Given that TPR remains 1.0 and FPR remains 0.0 for all levels of circuit complexity, the results shown in the table indicate no correlation between between TPR, FPR, or circuit complexity.

Table 20: OAI21 results

Test Circuit Name	Transistor Count	# OAI21s in Circuit	# OAI21s Found	TPR / FPR
Test3	50	1	1	1/1 = 1.0, 0/5 = 0.0
Full adder cell	60	1	1	1/1 = 1.0, 0/8 = 0.0
Test1	124	2	2	2/2 = 1.0, 0/18 = 0.0
4-bit adder	240	4	4	4/4 = 1.0, 0/32 = 0.0
Test4	650	13	13	13/13 = 1.0, 0/65 = 0.0

Table 21: OAI21b1 results

Test Circuit Name	Transistor Count	# OAI21b1s in Circuit	# OAI21b1s Found	TPR / FPR
Test3	50	1	1	1/1 = 1.0, 0/5 = 0.0
Full adder cell	60	1	1	1/1 = 1.0, 0/8 = 0.0
Test1	124	1	1	1/1 = 1.0, 0/19 = 0.0
4-bit adder	240	4	4	4/4 = 1.0, 0/32 = 0.0
Test4	650	13	13	13/13 = 1.0, 0/65 = 0.0

Recognition Accuracy - OAI21b0b1. Table 22 shows the TPR and FPR test results for the OAI21b0b1 gate in the order of increasing circuit complexity. Given that TPR remains 1.0 and FPR remains 0.0 for all levels of circuit complexity, the results shown in the table indicate no correlation between between TPR, FPR, or circuit complexity.

Table 22: OAI21b0b1 results

Test Circuit Name	Transistor Count	# OAI21b0b1s in Circuit	# OAI21b0b1s Found	TPR / FPR
Test3	50	1	1	1/1 = 1.0, 0/5 = 0.0
Test2	54	1	1	1/1 = 1.0, 0/7 = 0.0
Full adder cell	60	1	1	1/1 = 1.0, 0/8 = 0.0
Test1	124	1	1	1/1 = 1.0, 0/19 = 0.0
4-bit adder	240	4	4	4/4 = 1.0, 0/32 = 0.0
Test4	650	13	13	13/13 = 1.0, 0/65 = 0.0

Gate Recognition Comprehensiveness. The comprehensiveness of the gate recognition will be evaluated using two methods. The first method (Method 1) is to compare the list of gates recognized by the algorithm presented in this research (Section 3.2) to the list of gates included in a standard cell library. The second method (Method 2) is to compare the list of gates recognized by the algorithm to the lists of gates included in five TRUST test article circuits.

Method 1. The NCSU Digital Parts standard cell library contains 51 gates, which are listed in Appendix A. Of these 51 gates, the SCR algorithm is currently configured to identify five, which is approximately 10%. The SCR algorithm can identify four

additional cells not included in the NCSU Digital Parts standard cell library: NAND2b0, OAI21, OAI21b1, and OAI21b0b1.

Method 2. The first of the five TRUST test article circuits is composed of 330 cells of four types, as seen in Table 26 in Appendix B. The algorithm is capable of recognizing the cells shown in bold. Of the 330 cells contained in the circuit, the algorithm can recognize 306 cells; thus a 92.7% comprehensiveness is demonstrated for this test circuit.

The second TRUST test article circuits is composed of 9,423 cells of 179 types, as seen in Table 27 in Appendix B. The algorithm is capable of recognizing the cells shown in bold. Of the 9,423 cells contained in the circuit, the algorithm can recognize 5,381 cells; thus a 57.1% comprehensiveness is demonstrated for this test circuit.

The third TRUST test article circuits is composed of 22,097 cells of 167 types, as seen in Table 28 in Appendix B. The algorithm is capable of recognizing the cells shown in bold. Of the 22,097 cells contained in the circuit, the algorithm can recognize 9,736 cells; thus a 44.1% comprehensiveness is demonstrated for this test circuit.

The fourth TRUST test article circuits is composed of 62,783 cells of four types, as seen in Table 28. The algorithm is capable of recognizing the cells shown in bold. Of the 62,783 cells contained in the circuit, the algorithm can recognize 21,215 cells; thus a 33.8% comprehensiveness is demonstrated for this test circuit.

The fifth TRUST test article circuits is composed of 401,176 cells of 179 types, as seen in Table 28 in Appendix B. The algorithm is capable of recognizing the cells shown in bold. Of the 401,176 cells contained in the circuit, the algorithm can recognize 112,977 cells; thus a 28.2% comprehensiveness is demonstrated for this test circuit.

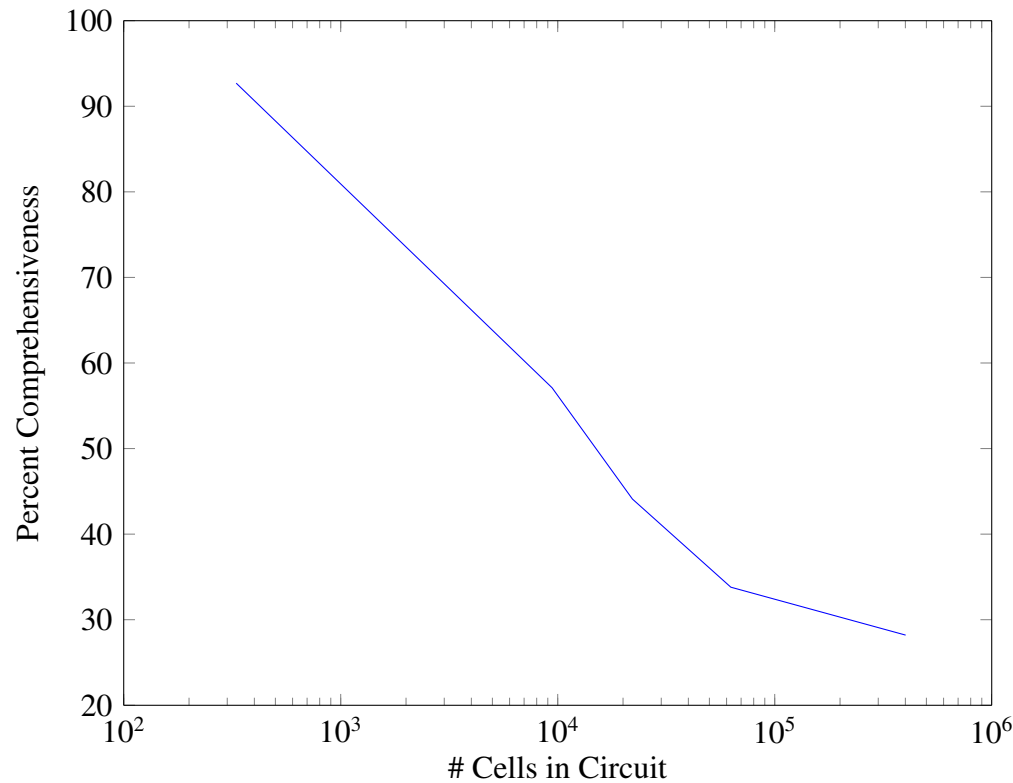
The SCR algorithm comprehensiveness for each of the TRUST test circuits are summarized in Table 23 below. The relationship between SCR algorithm comprehensiveness and circuit complexity is shown on a semi-logarithmic plot in Figure 58. It can be seen from the figure that there is an exponential decay for the percent

comprehensiveness as the circuit complexity increases. Hence, at present, the algorithm can only be usefully applied to a very small set of real-world circuits. Significant development of the algorithm is needed in order to apply it to a larger variety of real-world circuits.

Table 23: Algorithm comprehensiveness for TRUST test circuits

TRUST Test Circuit	# Cells in Circuit	# Cells Recognized by Algorithm	Percent Comprehensiveness
Test Circuit 1	330	306	92.7%
Test Circuit 2	9,423	5,381	57.1%
Test Circuit 3	22,097	9,736	44.1%
Test Circuit 4	62,783	21,215	33.8%
Test Circuit 5	401,176	112,944	28.2%

Figure 58: Percent comprehensiveness as a function of number of cells in circuit



4.2 Discussion of SCR Algorithm and Code

This section provides a thorough discussion about the SCR code developed in this SCR research, including: algorithms used, SCR algorithm attributes, and extensibility. The SCR code in its entirety can be found in Appendix C.

4.2.1 Explication of SCR Algorithm.

As seen in Figure 59, the code written to implement the SCR algorithm is divided into two parts: class definitions and functions. The class definitions represent each type of object, and the functions conduct the operations necessary to perform SCR.

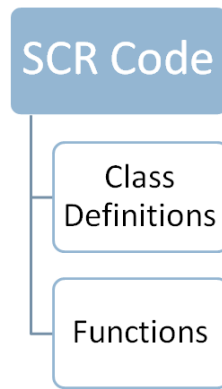


Figure 59: Main components of SCR code

4.2.1.1 Class Definitions.

The types of objects defined by the class definitions include transistors, gates, and gate types. The organization of these objects is represented in Figure 60. There are two distinct types of transistors: the NMOS transistor and the PMOS transistor. The gates are the inverter, NAND2, NOR2, AND2, OR2, NAND2b0, OAI21, OAI21b1, and OAI21b0b1. The gate types are: inverter_type, NAND2_type, NOR2_type, AND2_type, OR2_type, NAND2b0_type, OAI21_type, OAI21b1_type, and OAI21b0b1_type. The

purpose of the gate types is to distinguish between various types of a given gate (for example, NAND2s of varying widths).

The class definitions of the transistors attach the following attributes to each transistor object: width, length, nets, and lines. Naturally, the width and length attributes are the width and length of the transistor. The nets attribute refers to the nets, or the connections, of the transistor. The lines attribute records the text strings that defined the transistor in the input netlist.

The class definitions of the gates attach the certain attributes to each gate object depending on the gate's level of abstraction. For gates in the second level of abstraction, the following attributes are attached: the NMOS and PMOS transistors that compose the gate, along with each transistor's width, length, nets, and lines. For gates in the third level of abstraction or higher, the following attributes are attached: the sub-gates that compose the gate, along with each sub-gate's transistors' widths, lengths, nets, and lines.

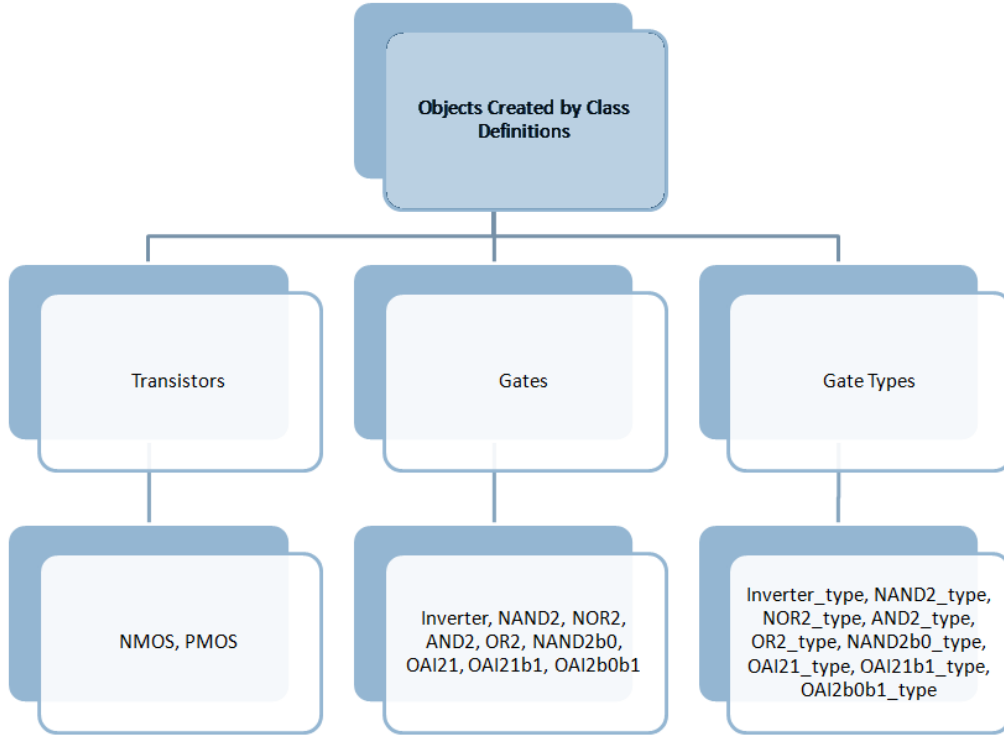


Figure 60: Organization of objects created by class definitions in the SCR code

4.2.1.2 Functions.

The functions in the SCR code construct the algorithm that conducts the SCR process. The overall process of the SCR algorithm can be summarized in seven steps:

1. Input (read) transistor-level netlist
2. Identify transistors in netlist (first level of abstraction)
3. Identify gates at the second level of abstraction
4. Identify gates at the third level of abstraction
5. Identify gates at the fourth level of abstraction
6. Identify unique gate sizes for each type of gate

7. Output (write) gate-level netlist

A graphical representation of this process is seen in Figure 61.

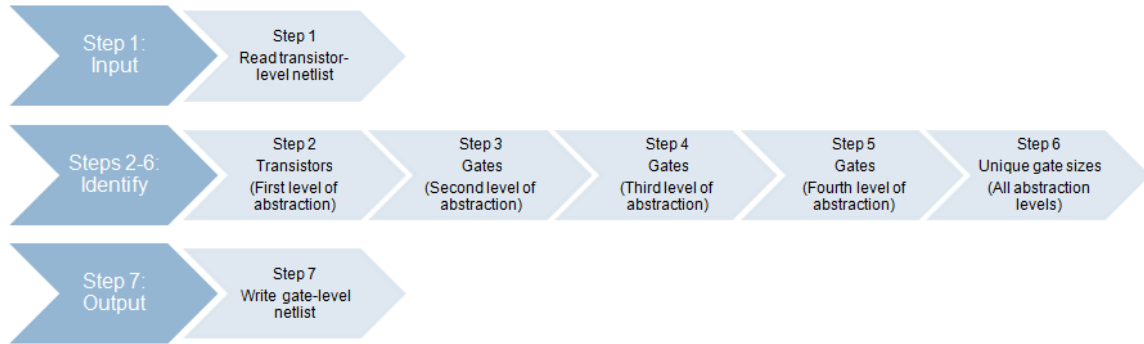


Figure 61: The seven steps of the SCR algorithm

Within the section of the SCR code that describes the functions, there is a four-level hierarchy of functions. In this system, the parent level calls functions located in the child level to perform a subroutine.

Top (First) Level of Hierarchy. The top level of hierarchy contains only one function: SCR. This function calls the child functions that perform the seven steps of the algorithm depicted in Figure 61. Global variables and arrays initialized prior to calling the SCR function are used to keep track of the objects created and used by each of the child functions. The child functions called by SCR are: `find_tx`, `find_cells`, `find_cell_types`, and `replace_cells`. Figure 62 represents the flow of the SCR function in the top level of hierarchy.

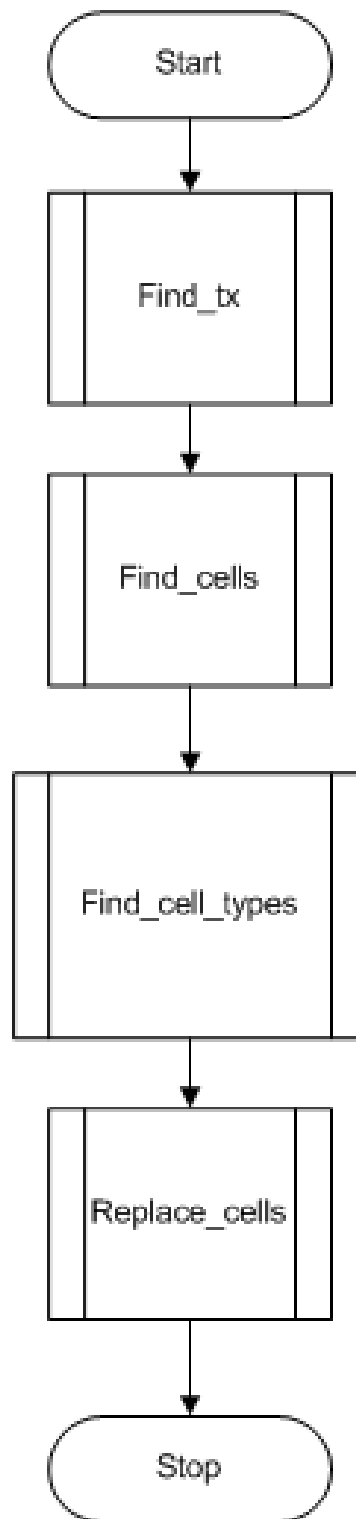


Figure 62: Flowchart of the SCR function in the top level of hierarchy

Second Level of Hierarchy. The functions in the second level of hierarchy are those called by the top level function: Find_tx, Find_cells, Find_cell_types, and Replace_cells. The function “Find_tx” serves to read the input transistor-level netlist line-by-line to identify which lines of the netlist describe the NMOS transistors and the PMOS transistors. It then calls two third-level functions: Create_NMOS_objects and Create_PMOS_objects. Figure 63 represents the flow of the Find_tx function. The flowcharts for Create_NMOS_objects and Create_PMOS_objects will be presented in the next section.

The function “Find_cells” identifies each of the nine gates by calling several third-level functions: Find_inverter, Find_NAND2, Find_NOR2, Find_AND2, Find_OR2, Find_OAI21b0b1, Find_OAI21b1, Find_OAI21, and Find_NAND2b0. Figure 64 represents the flow of the Find_cells function. The flowchart for the third-level functions called by the Find_cells function will be presented in the next section.

For each of the gates found by the “Find_cells” function, the function “Find_cell_types” identifies unique types of a given gate (for example, it distinguishes between a standard-sized inverter and a double-wide inverter). It accomplishes this by calling several third-level functions: Find_inverter_types, Find_NAND2_types, Find_NOR2_types, Find_AND2_types, Find_OR2_types, Find_OAI21b0b1_types, Find_OAI21b1_types, Find_OAI21_types, and Find_NAND2b0_types. Figure 65 represents the flow of the Find_cell_types function. The flowchart for the third-level functions called by the Find_cell_types function will be presented in the next section.

The function “Replace_cells” then identifies and saves the comments of the input netlist, identifies the portion of the input netlist that lists the components, transfers the comments to the output netlist, and then writes the gates to the gate-level output netlist by calling three third-level functions: Remove_cell_transistors, Add_cells_to_components, and Write_cells_to_netlist. Figure 66 represents the flow of the Replace_cells function. The

flowcharts for the third-level functions called by the Replace_cells function will be presented in the next section.

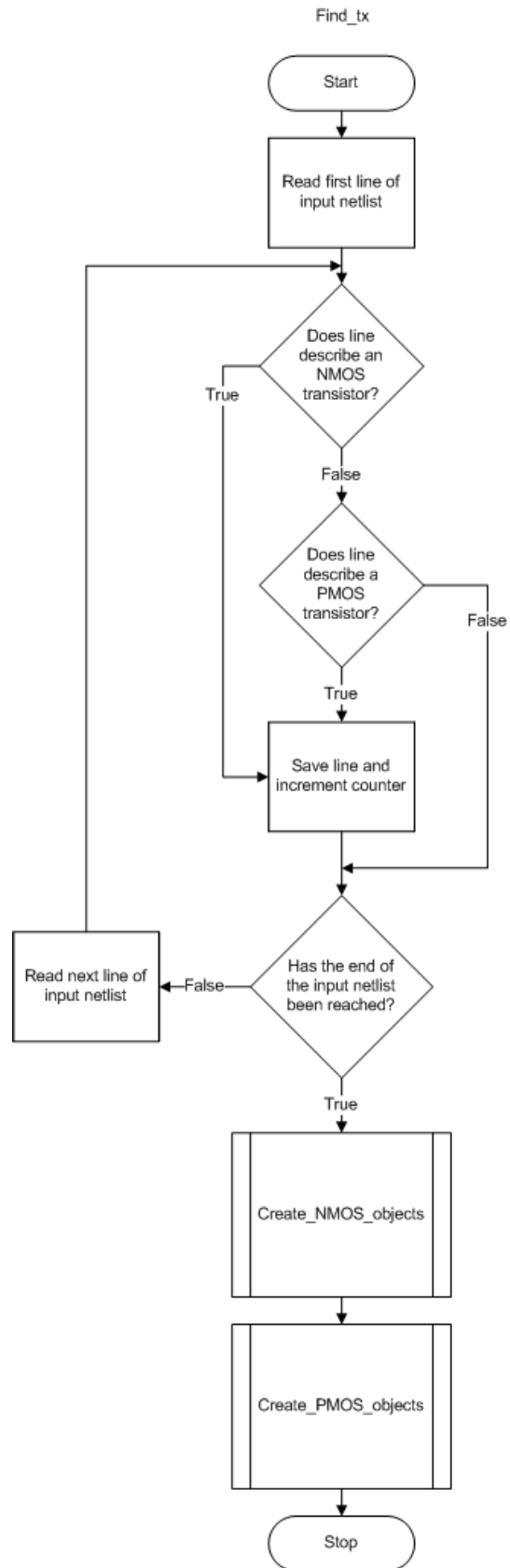


Figure 63: Flowchart of the Find_tx function in the second level of hierarchy

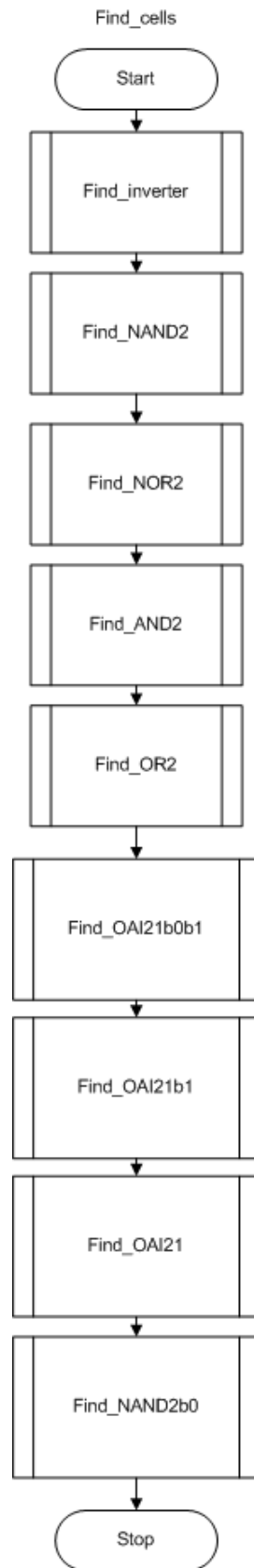


Figure 64: Flowchart of the Find_cells function in the second level of hierarchy

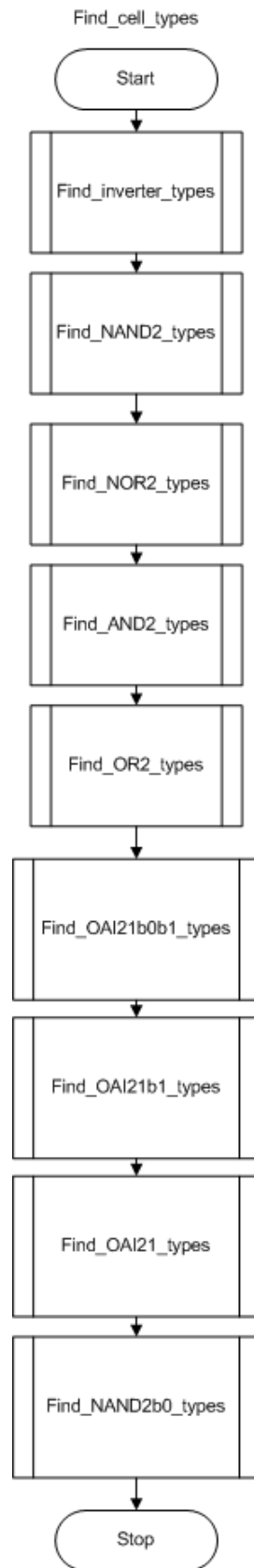


Figure 65: Flowchart of the Find_cell_types function in the second level of hierarchy

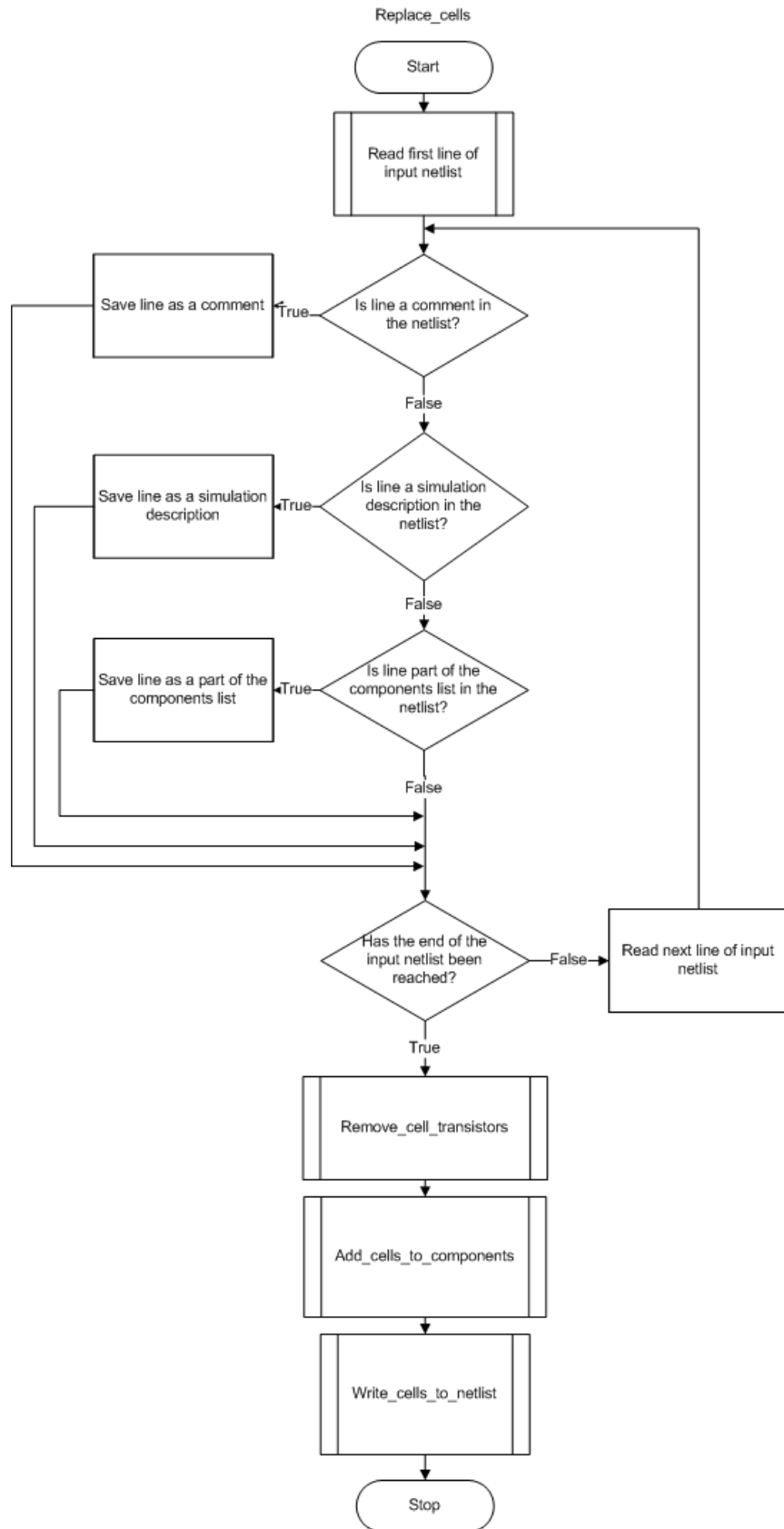


Figure 66: Process flow of the `replace_cells` function in the second level of hierarchy

Third Level of Hierarchy. The functions in the third level of hierarchy are:

Create_NMOS_objects, Create_PMOS_objects, Find_inverter, Find_NAND2, Find_NOR2, Find_AND2, Find_OR2, Find_NAND2b0, Find_OAI21, Find_OAI21b1, Find_OAI21b0b1, Find_inverter_types, Find_NAND2_types, Find_NOR2_types, Find_AND2_types, Find_OR2_types, Find_NAND2b0_types, Find_OAI21_types, Find_OAI21b1_types, Find_OAI21b0b1_types, Remove_cell_transistors, Add_cells_to_components, and Write_cells_to_netlist. The functions Create_NMOS_objects and Create_PMOS_objects perform essentially the same task. Create_NMOS_objects invokes the NMOS class to create NMOS objects that correspond to the NMOS transistors identified in the input netlist. Likewise, Create_PMOS_objects does the same for PMOS transistors. Figures 67 and 68, respectively, show the flowcharts for the two functions.

For the sake of simplicity, the functions Find_inverter, Find_NAND2, Find_NOR2, Find_AND2, Find_OR2, Find_NAND2b0, Find_OAI21, Find_OAI21b1, and Find_OAI21b0b1 will be discussed using the example of Find_OR2. The other functions perform the same series of steps for the function's respective gate. The first step of the function Find_OR2 is to identify any OR2 gates as explained Section 4.1.2. For every OR2 gate identified, the function invokes the OR2 class to create an OR2 object. The OR2 object is stored in a global array, and the objects that composed the OR2 object (the inverter and the NOR2 object) are removed from their respective global arrays and stored in separate arrays that contain other objects removed from the global arrays. The only deviation from this pattern occurs for the functions Find_inverter, Find_NAND2, and Find_NOR2. In the same way that gates are removed from their respective global arrays, these three functions call the fourth-level function Remove_transistors in order to remove the transistors that compose the inverter, NAND2, or NOR2 gates. The flowchart for the Find_OR2 function is depicted in Figure 69.

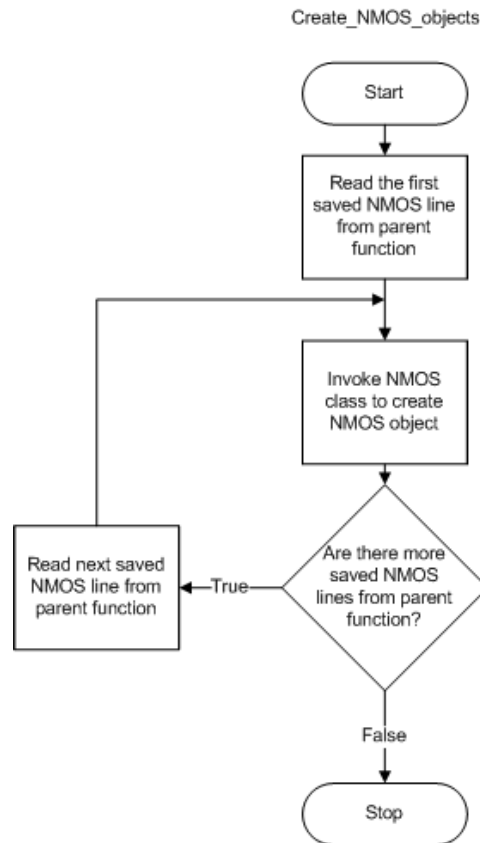


Figure 67: Flowchart of the Create_NMOS_objects function in the third level of hierarchy

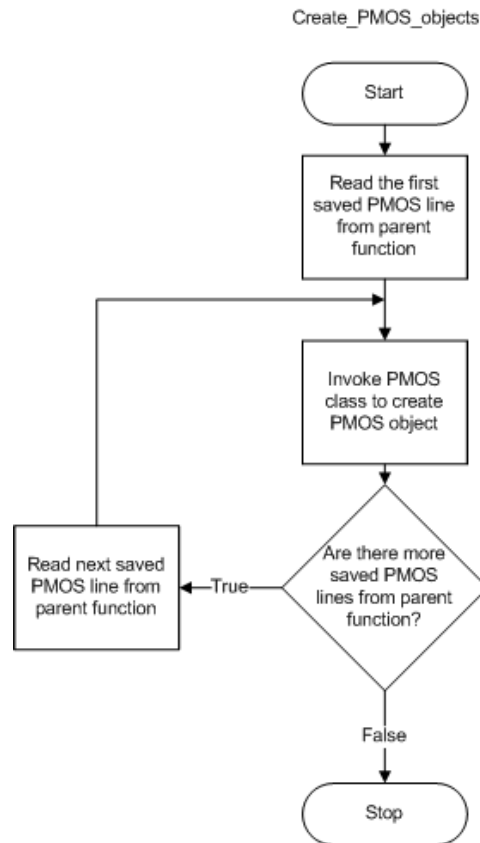


Figure 68: Flowchart of the Create_PMOS_objects function in the third level of hierarchy

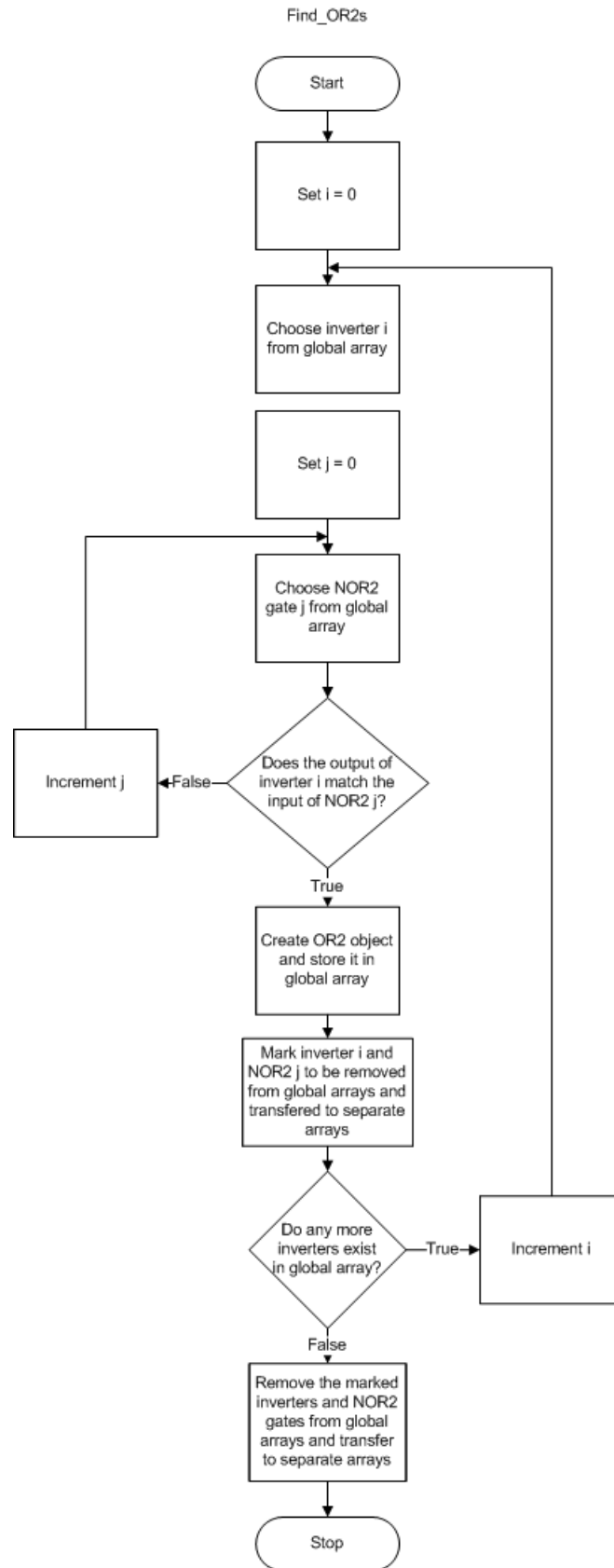


Figure 69: Flowchart of the Find_OR2s function in the third level of hierarchy

Again, for the sake of simplicity, the functions Find_inverter_types, Find_NAND2_types, Find_NOR2_types, Find_AND2_types, Find_OR2_types, Find_NAND2b0_types, Find_OAI21_types, Find_OAI21b1_types, and Find_OAI21b0b1_types will be discussed using the example of Find_OR2_types. The other functions perform the same series of steps for the function's respective gate. The first step of the function Find_OR2_types is to examine each OR2 object in the global array to identify the set of distinct OR2 types and to assign the type as an attribute to the OR2 object. Next, it examines any OR2 objects in the separate "removed objects" array to identify any other distinct OR2 types and assign the type as an attribute to the OR2 object. This process is detailed in the flowcharts shown in Figures 70 and 71. No deviations from this pattern occur.

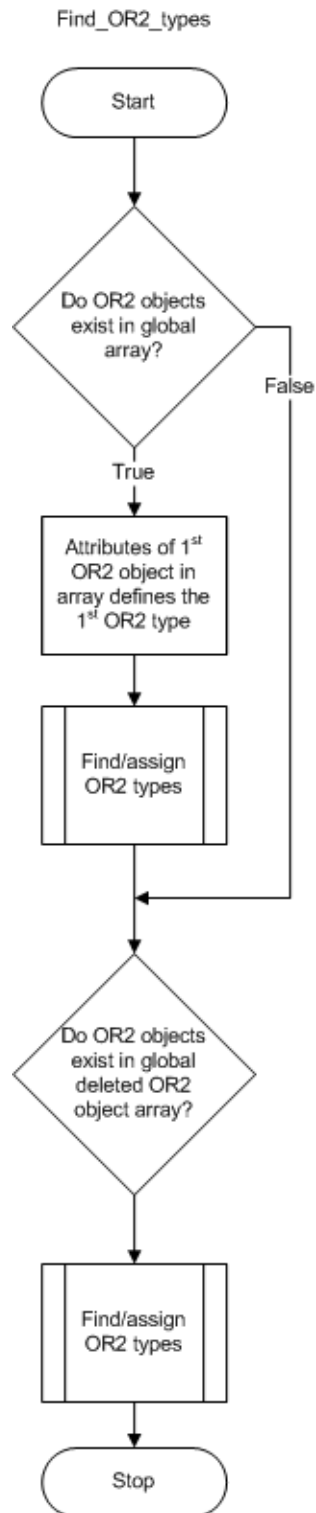


Figure 70: Flowchart for the Find_OR2_types function

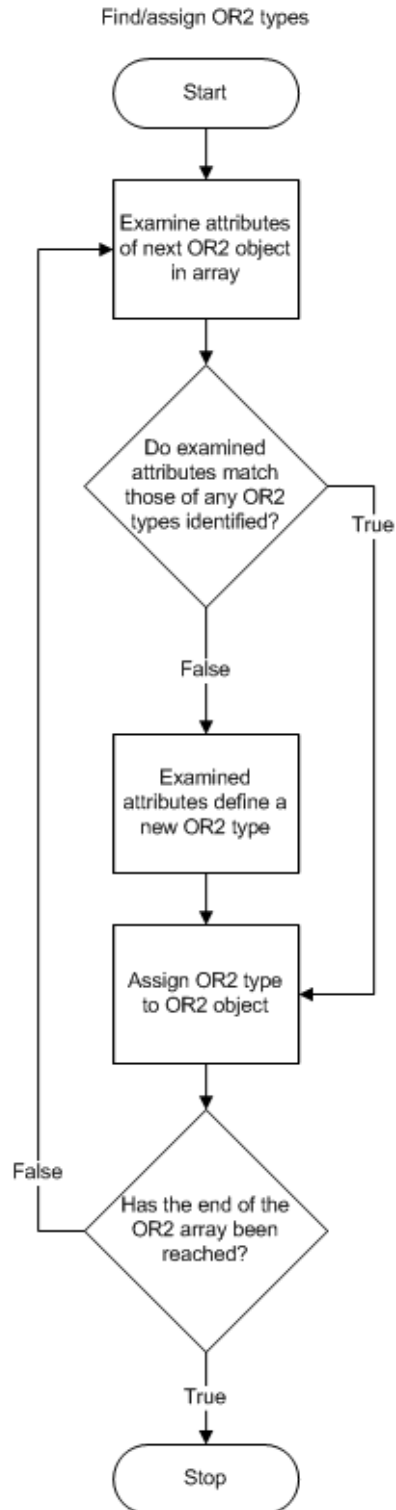


Figure 71: Flowchart for the Find/assign OR2 types subroutine used in Figure 70

The function `Remove_cell_transistors` simply calls the following fourth-level functions to remove the transistors that compose gates from the components list of the input netlist: `Remove_OAI21b0b1_transistors_from_components`, `Remove_OAI21b1_transistors_from_components`, `Remove_OAI21_transistors_from_components`, `Remove_NAND2b0_transistors_from_components`, `Remove_OR2_transistors_from_components`, `Remove_AND2_transistors_from_components`, `Remove_NOR2_transistors_from_components`, `Remove_NAND2_transistors_from_components`, and `Remove_inverter_transistors_from_components`. The flowchart for `Remove_cell_transistors` is shown in Figure 72.

The function `Add_cells_to_components` serves two purposes. First, it provides the framework for each of the gates to be assigned a component number for insertion into the components list. Second, it calls the following fourth-level functions to insert the gates to the components list: `Add_OAI21b0b1s_to_components`, `Add_OAI21b1s_to_components`, `Add_OAI21s_to_components`, `Add_NAND2b0s_to_components`, `Add_OR2s_to_components`, `Add_AND2s_to_components`, `Add_NOR2s_to_components`, `Add_NAND2s_to_components`, and `Add_inverters_to_components`. The flowchart for `Add_cells_to_components` is shown in Figure 73.

The function `Write_cells_to_netlist` serves three purposes. First, it provides the framework for the subcircuit definition of each gate type to be written to the output netlist by passing a subcircuit definition array between child functions. Second, it calls the following child (fourth-level) functions to create the subcircuit definitions of the gates: `Write_OAI21b0b1s_to_netlist`, `Write_OAI21b1s_to_netlist`, `Write_OAI21s_to_netlist`, `Write_NAND2b0s_to_netlist`, `Write_OR2s_to_netlist`, `Write_AND2s_to_netlist`,

Write_NOR2s_to_netlist, Write_NAND2s_to_netlist, and Write_inverters_to_netlist. Third, Write_cells_to_netlist writes subcircuit definition array, the comments, components list, and simulation description to the output netlist. The flowchart for Add_cells_to_components is shown in Figure 74.

Bottom (Fourth) Level of Hierarchy. There are three categories of functions in the fourth level of hierarchy. The first category consists of the functions called by the third-level function Remove_cell_transistors: Remove_transistors, Remove_OAI21b0b1_transistors_from_components, Remove_OAI21b1_transistors_from_components, Remove_OAI21_transistors_from_components, Remove_NAND2b0_transistors_from_components, Remove_OR2_transistors_from_components, Remove_AND2_transistors_from_components, Remove_NOR2_transistors_from_components, Remove_NAND2_transistors_from_components, and Remove_inverter_transistors_from_components. For the sake of simplicity, these functions will be discussed using the example of Remove_OR2_transistors_from_components. The other functions perform the same series of steps for the function's respective gate. Remove_OR2_transistors_from_components searches through the components list from the input netlist for the transistors that compose the OR2 gates. The identified transistors are then removed from the components list. No deviations from this pattern occur. The flowchart for the Remove_OR2_transistors_from_components function is depicted in Figure 75.

The second category consists of the functions called by the third-level function Add_cells_to_components: Add_OAI21b0b1s_to_components,

Add_OAI21b1s_to_components, Add_OAI21s_to_components,
Add_NAND2b0s_to_components, Add_OR2s_to_components,
Add_AND2s_to_components, Add_NOR2s_to_components,
Add_NAND2s_to_components, and Add_inverters_to_components. For the sake of
simplicity, these functions will be discussed using the example of
Add_OR2s_to_components. The other functions perform the same series of steps for the
function's respective gate. Add_OR2s_to_components is a simple function; it adds the
OR2 gate instances to the components list by iterating over the OR2s global array and
inserting the relevant information (gate inputs, outputs, and power pins) of each OR2 gate
instance into the components list. No deviations from this pattern occur. The flowchart for
the Add_OR2s_to_components function is depicted in Figure 76.

The third category consists of the functions called by the third-level function
Write_cells_to_netlist: Write_OAI21b0b1s_to_netlist, Write_OAI21b1s_to_netlist,
Write_OAI21s_to_netlist, Write_NAND2b0s_to_netlist, Write_OR2s_to_netlist,
Write_AND2s_to_netlist, Write_NOR2s_to_netlist, Write_NAND2s_to_netlist, and
Write_inverters_to_netlist. For the sake of simplicity, these functions will be discussed
using the example of Write_OR2s_to_netlist. The other functions perform the same series
of steps for the function's respective gate. Write_OR2s_to_netlist accepts the subcircuit
definition array, iterates through each of the OR2 types, and writes the subcircuit
definition of each type to the array. After iterating through each of the OR2 types, it
returns the array to the parent function. No deviations from this pattern occur. The
flowchart for the Write_OR2s_to_netlist function is depicted in Figure 77.

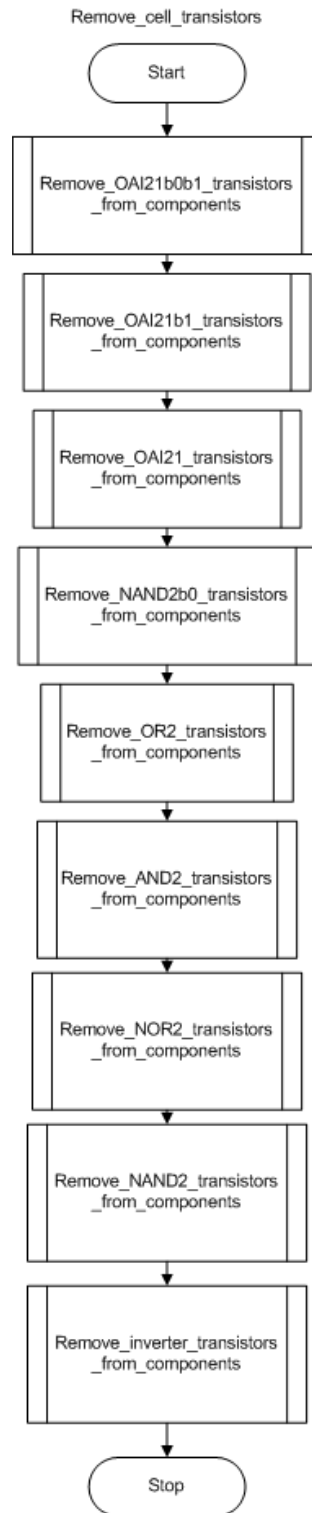


Figure 72: Flowchart for the function Remove_cell_transistors in the third level of hierarchy

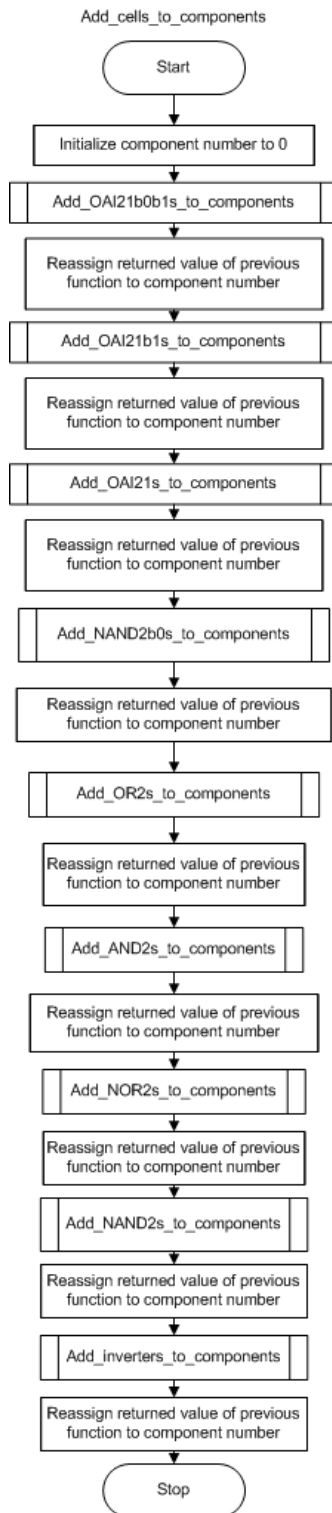


Figure 73: Flowchart for the function Add_cells_to_components in the third level of hierarchy

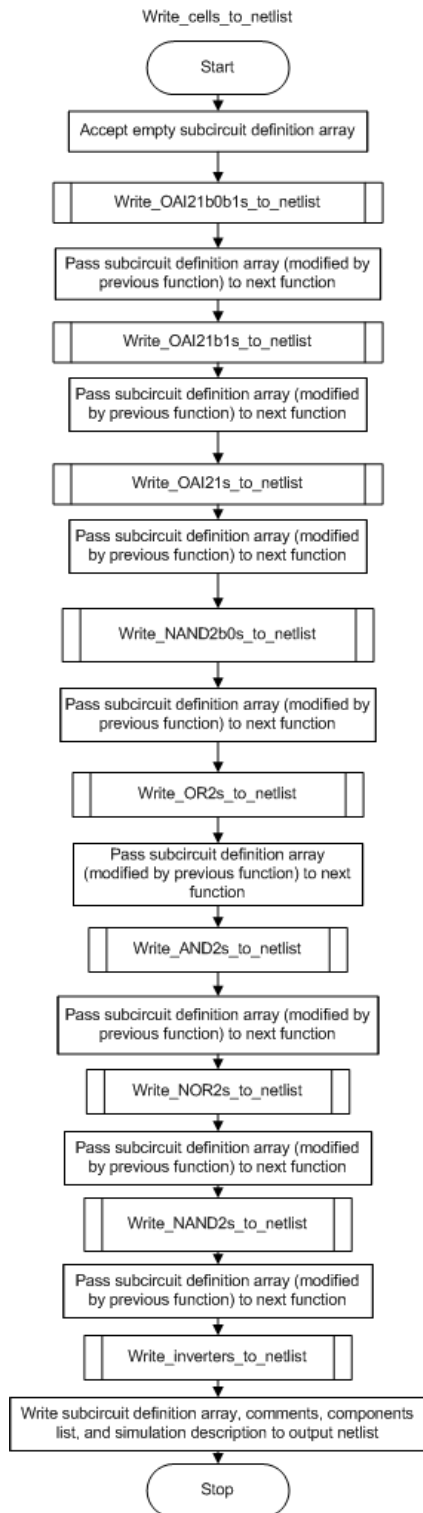


Figure 74: Flowchart for the function Write_cells_to_netlist in the third level of hierarchy

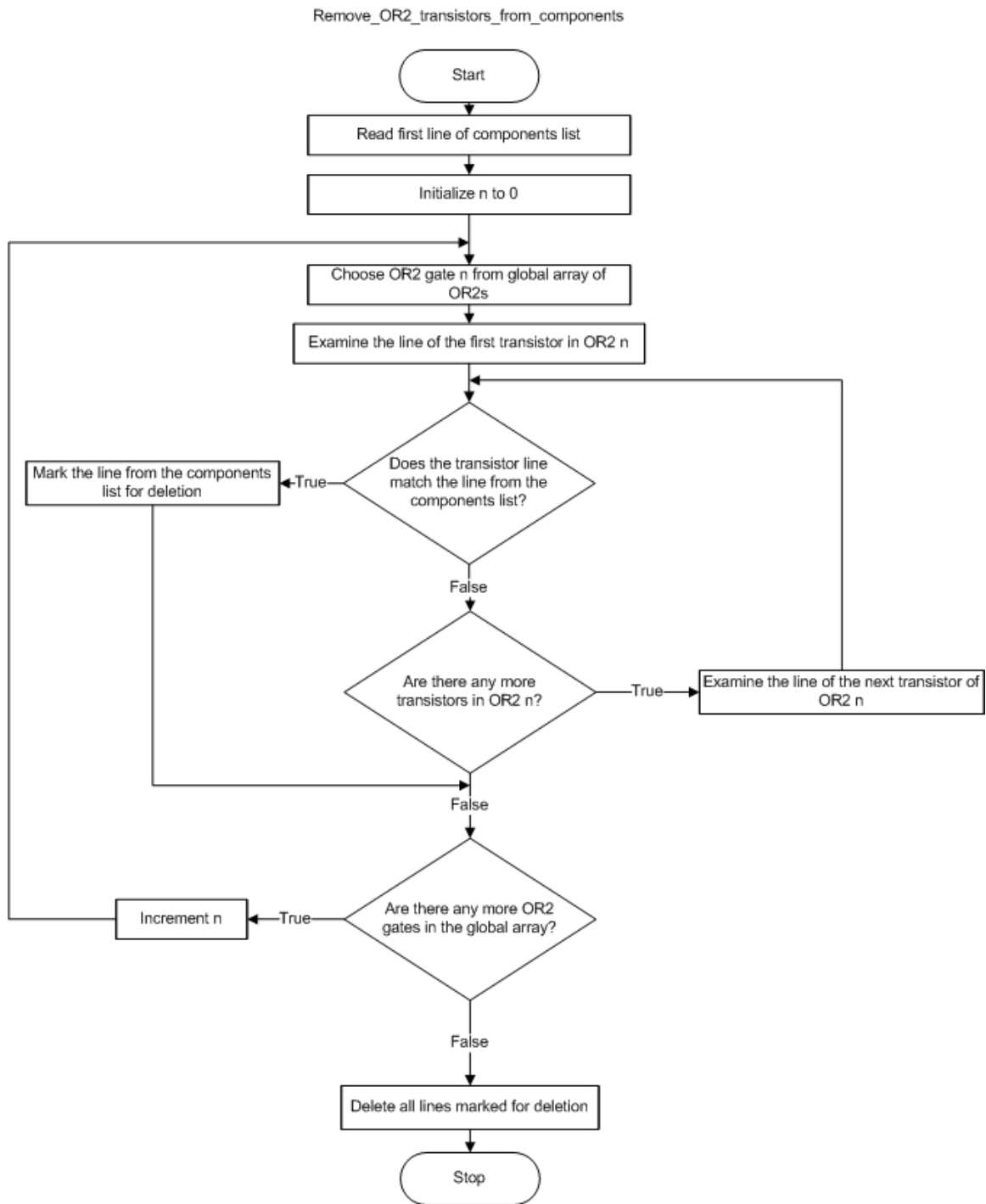


Figure 75: Flowchart for the function Remove_OR2_transistors_from_components in the fourth level of hierarchy

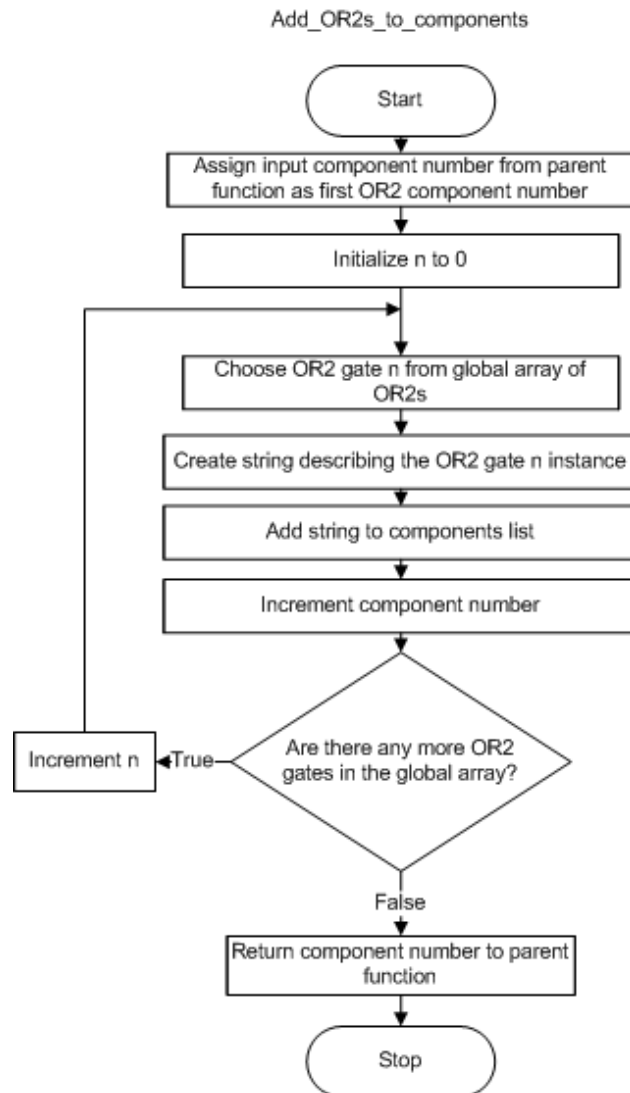


Figure 76: Flowchart for the function Add_OR2s_to_components in the fourth level of hierarchy

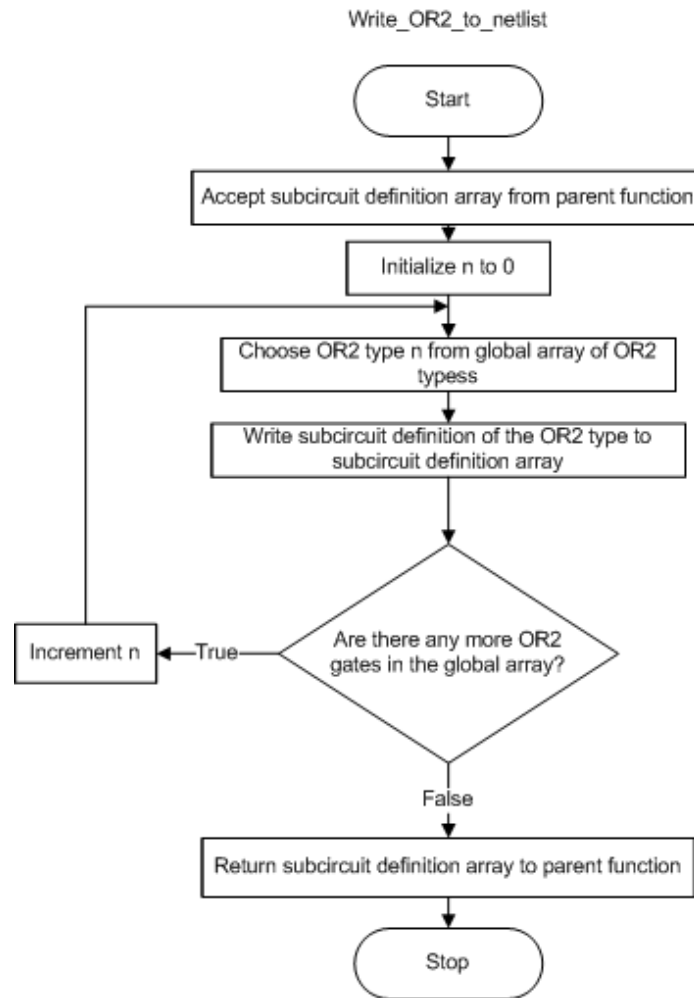


Figure 77: Flowchart for the function Write_cells_to_netlist in the fourth level of hierarchy

4.2.2 Algorithm Attributes.

There are a few attributes of the SCR algorithm that are not clearly captured by the discussion in Section 4.2.1, but are important to mention. These attributes include the algorithm's extensibility; vital characteristics that affect the success of gate recognition; and abilities to handle various types of gate inputs and outputs, accurately identify gates used in feedback structures, and successfully produce a netlist even when un-recognized gates are encountered. Each attribute will be discussed in the following paragraphs.

Extensibility. In terms of extensibility, the form of future growth that must be taken into consideration when crafting the structure of the SCR code is the addition of gates that the SCR algorithm can recognize. In order to accomodate this type of growth, the SCR code was given a hierarchical structure. The hierarchical structure provides for good organization of the SCR code, which makes it simple to understand where code must be added to provide for the processing of new gates. The following example explains how to add code in order to expand the SCR algorithm to recognize an OR3 gate. No adjustments would be made to the SCR function in the first hierarchical level, as functions for specific gates are detailed in lower levels of hierarchy. In the second hierarchical level, the functions Find_cells and Find_cell_types would need to be modified. Find_cells would be modified to call an additional child function: Find_OR3. Similarly, Find_cell_types would be modified to call an additional child function: Find_OR3_types. As Find_cells is structured to call its child functions in the order of increasing level of gate abstraction (this concept is discussed in the next section), Find_OR3 would be inserted between the functions Find_OR2 and Find_NAND2b0. Likewise, in the parent function Find_cell_types, the child function Find_OR3_types would be inserted between the child functions Find_OR2_types and Find_NAND2b0_types. In the third hierarchical level, the functions Find_OR3 and Find_OR3_types would need to be defined. Additionally, the parent functions Remove_cell_transistors, Add_cells_to_components, and

Write_cells_to_netlist would need to be modified with the addition of calls to the child functions Remove_OR3_transistors_from_components, Add_OR3s_to_components, and Write_OR3s_to_netlist, respectively. In the fourth hierarchical level, the functions Remove_OR3_transistors_from_components, Add_OR3s_to_components, and Write_OR3s_to_netlist would need to be defined. In summary, the process for expanding the SCR algorithm to recognize additional gates requires modifying parent functions in the second and third hierarchical levels and defining child functions in the third and fourth hierarchical levels.

Vital Characteristics for Success in Gate Recognition. There are a few important characteristics of the algorithm that greatly affect its ability to accurately recognize the gates. First, the algorithm executes a specific order in finding the gates. The gates are found in order of increasing levels of abstraction, but decreasing complexity within each level of abstraction. Gates within a level of abstraction must be found in order of decreasing complexity since certain gates in a given level can appear to be composed of other gates within the same level. For example, as seen in Figure 78, a NAND2b0 consists of an inverter and an OR2 gate, and an OAI21b1 consists of a NAND2 gate, OR2 gate, and inverter. Thus, if the gates of fourth-level abstraction were identified in order of increasing complexity (OAI21b1 identified after NAND2b0), the OAI21b1 gate would be falsely identified as a NAND2b0 gate connected to a NAND2 gate. Identifying gates in order of decreasing complexity (NAND2b0 identified after OAI21b1) enables accurate identification.

Second, the algorithm must specify rules to identify fourth-level gates. For example, Figure 78 shows that the OAI21b0b1 consists of an OR2, AND2, and inverter. In the figure, a NAND2 gate is connected to the inverter of the OAI21b0b1; thus, given that gate identification occurs in order of increasing levels of abstraction, instead of identifying an OAI21b0b1 connected to a NAND2 gate, the algorithm (without rules) would identify an

OR2 gate connected to two AND2 gates. The rules guide the algorithm to separate the AND2 gate (that has the OR2s output as an input) into a NAND2 gate and an inverter so that the OAI2b0b1 can be identified.

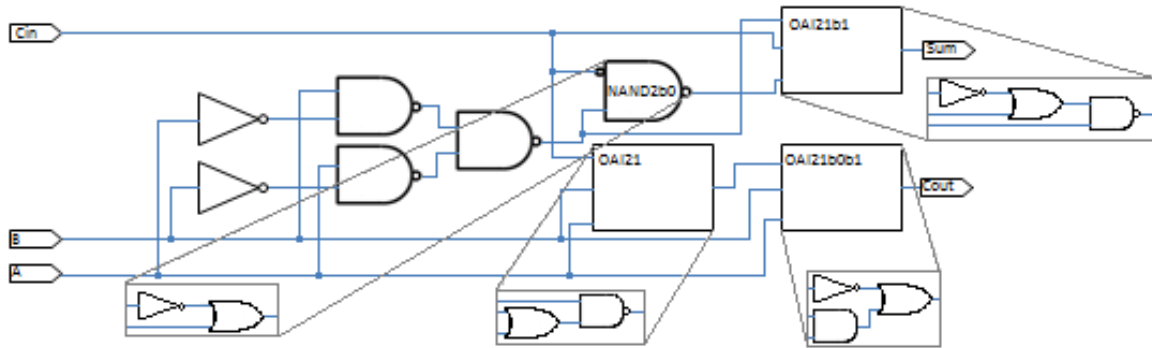


Figure 78: Full adder cell. Compositions of gates at the fourth level of abstraction are shown in boxes

Gate Inputs. There are four significant details regarding the algorithm’s ability to handle various types of gate inputs and outputs. First, the algorithm can recognize gates regardless of whether or not the inputs are pins or are connected to actual power sources. Second, the algorithm can recognize gates regardless of the output connection (a pin or a load). The first and second details are evidenced by the success of the algorithm in recognizing both the gates in Section 4.1.2, which are connected to power sources, and the gates in Section 4.1.4, which are not connected to power sources. Third, one signal can be connected to more than one input of a single gate, as evidenced in Section 4.1.4 with the example of the 2-to-1 MUX circuit. This example also provides evidence for the fourth detail, that one signal can be connected to inputs of more than one gate.

Feedback Structures. In the example of the Master/Slave DFF in Section 4.1.4, it can be seen from Figure 43 that an output of one NAND2 gate (Gate A) feeds into the input of another NAND2 gate (Gate B), and the output of Gate B feeds into the input of

Gate A. Hence, it is a feedback structure. The algorithm achieved perfect recognition results for the Master/Slave DFF, indicating that the algorithm is capable of processing feedback structures.

Encounters with Unrecognized Gates. A significant feature of the algorithm is that it is still capable of producing an output netlist even when it is unable to recognize all the gates in the circuit. In this type of situation, the algorithm simply maintains a transistor-level representation of the unrecognized gates in the output netlist. Hence, the output netlist will not be constructed purely at the gate level or purely at the transistor level. Rather, it will contain a mixture of both. An example of this can be seen with the test circuit Test1. The output netlist for Test1 generated by the algorithm is reproduced in Appendix D.

4.3 Advantages of Transistor-level Verification with SCR over Functional Testing

One of the shortcomings of functional testing discussed in Section 2.1.3 is the inability to detect malicious changes in an IC if the logic of the device under test remains unaffected by the change. An advantage of transistor-level verification with SCR is that it identifies transistor-level alterations even when the logic is unchanged. This advantage is demonstrated by the following two examples of transistor-level alterations to the 4-bit ripple carry adder presented in Section 4.1.4.1. The first alteration, discussed in Section 4.3.1, is a malicious change in a single gate's composition, and the second alteration, discussed in Section 4.3.2 is a malicious switch of a single gate's input signals.

4.3.1 Malicious Change in Gate Composition.

Figures 79 and 80 show the gate-level schematic and the transistor-level schematic, respectively, of the unmodified 4-bit adder. Figures 81 and 82 show the gate-level schematic and the transistor-level schematic, respectively, of a maliciously modified 4-bit adder. As seen by comparing Figures 79 and 81, the modification occurs in the lower-right full adder cell (Full Adder Cell 0). The OAI21 gate in Full Adder Cell 0 from Figure 79 is

maliciously modified in Figure 81, as indicated by the red gate marked with the “X.” In this research, an unmodified OAI21 gate is composed of an OR2 gate and a NAND2 gate. Using DeMorgan’s Theorem, the OAI21 gate is maliciously altered to maintain the same logic while changing the composition to two inverters, an AND2 gate, and a NAND2 gate. The composition of the maliciously altered OAI21 gate can be seen in Figure 83.

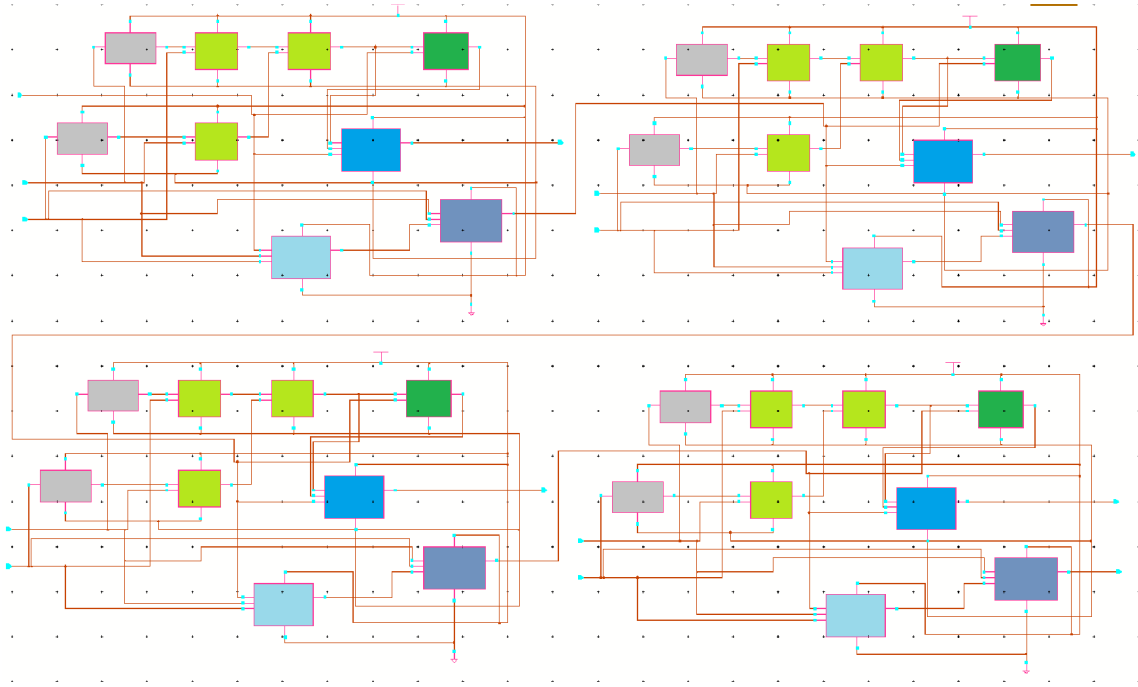


Figure 79: Gate-level representation of the unmodified 4-bit ripple carry adder circuit

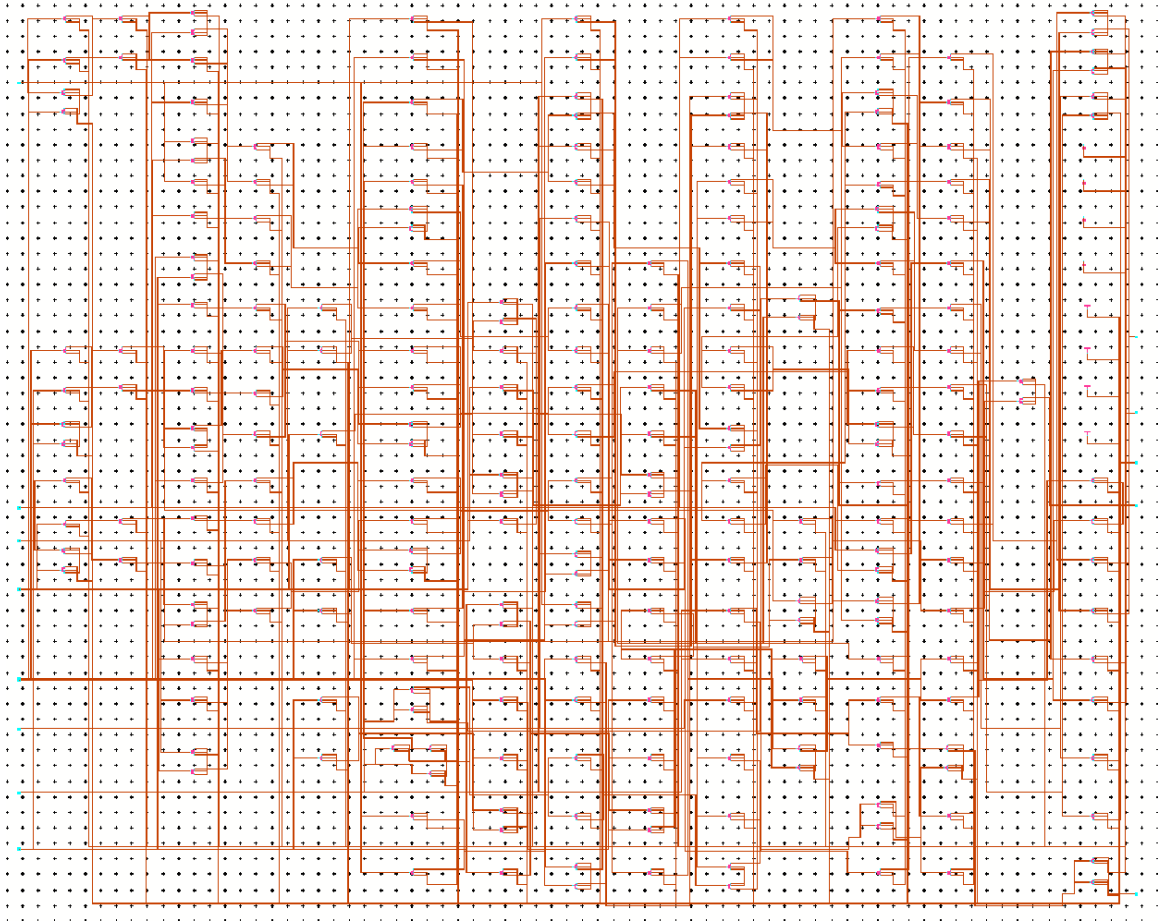


Figure 80: Transistor-level schematic of the unmodified 4-bit ripple carry adder circuit

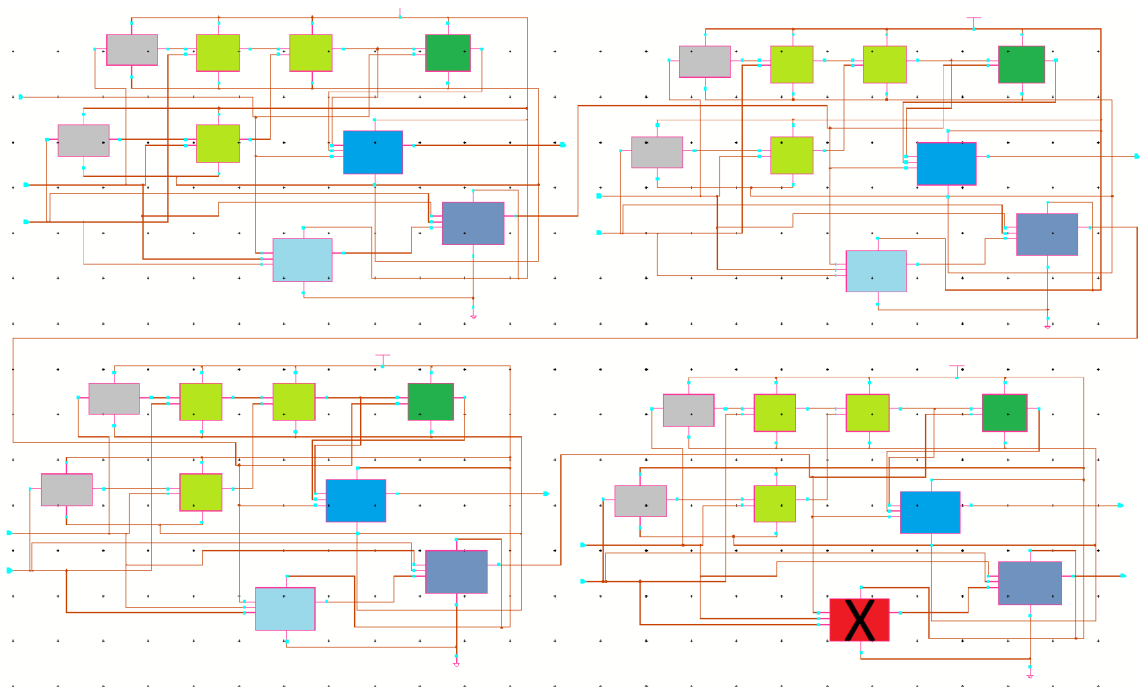


Figure 81: Gate-level representation of the altered (changed OAI21 gate composition in Full Adder Cell 0) 4-bit ripple carry adder circuit

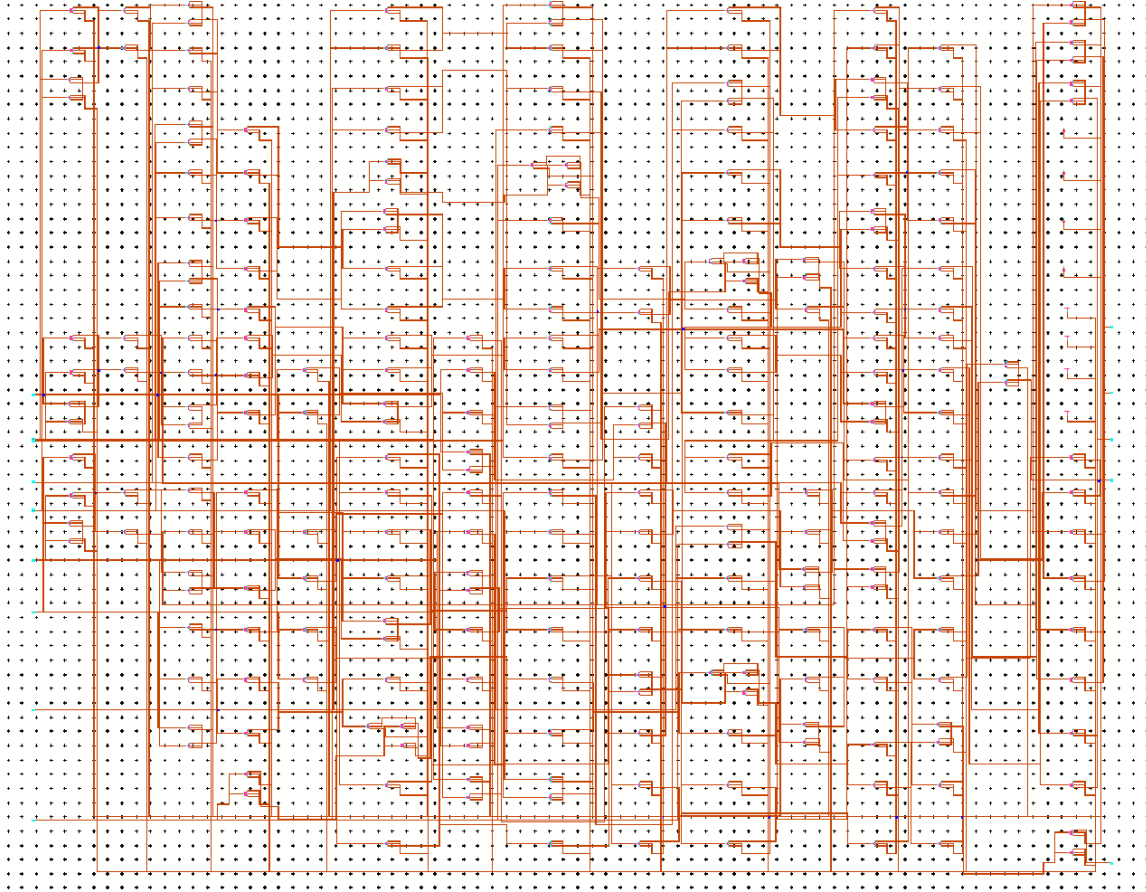


Figure 82: Transistor-level schematic of the altered (changed OAI21 gate composition in Full Adder Cell 0) 4-bit ripple carry adder circuit

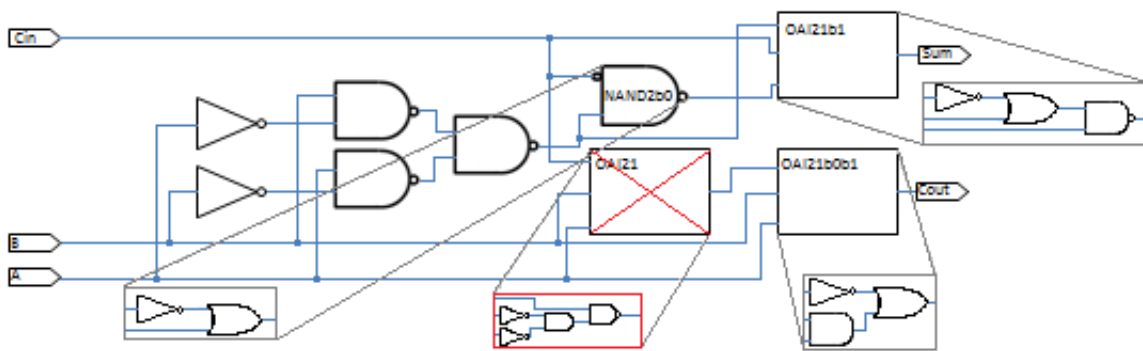


Figure 83: Altered full adder cell in the 4-bit ripple carry adder circuit. Composition of the maliciously altered OAI21 gate at the third level of abstraction is shown in the red box

Tables 24 and 25 show the truth tables for the unmodified Full Adder Cell 0 and the maliciously modified Full Adder Cell 0, respectively. The tables show the cell inputs, outputs, and intermediate signals. The inputs (A , B , and C_{in}) and outputs (S and C_{out}) are shown in black in Figures 84 and 85, and the intermediate signals ($I0$, $I1$, $I2$, $I3$, $I4$, and $I5$) are shown in gray. By comparing Tables 24 and 25, it is clear that the logic for both full adder cells is the same.

Table 24: Truth table of the unmodified Full Adder Cell 0

A	B	C_{in}	I0	I1	I2	I3	I4	I5	I6	C_{out}	Sum
0	0	0	1	1	1	1	0	1	1	0	0
0	0	1	1	1	1	1	0	0	1	0	1
0	1	0	1	0	0	1	1	1	1	0	1
0	1	1	1	0	0	1	1	1	0	1	0
1	0	0	0	1	1	0	1	1	1	0	1
1	0	1	0	1	1	0	1	1	0	1	0
1	1	0	0	0	1	1	0	1	1	1	0
1	1	1	0	0	1	1	0	0	0	1	1

Table 25: Truth table of the maliciously modified Full Adder Cell 0

A	B	C_{in}	I0	I1	I2	I3	I4	I5	I6	C_{out}	Sum
0	0	0	1	1	1	1	0	1	1	0	0
0	0	1	1	1	1	1	0	0	1	0	1
0	1	0	1	0	0	1	1	1	1	0	1
0	1	1	1	0	0	1	1	1	0	1	0
1	0	0	0	1	1	0	1	1	1	0	1
1	0	1	0	1	1	0	1	1	0	1	0
1	1	0	0	0	1	1	0	1	1	1	0
1	1	1	0	0	1	1	0	0	0	1	1

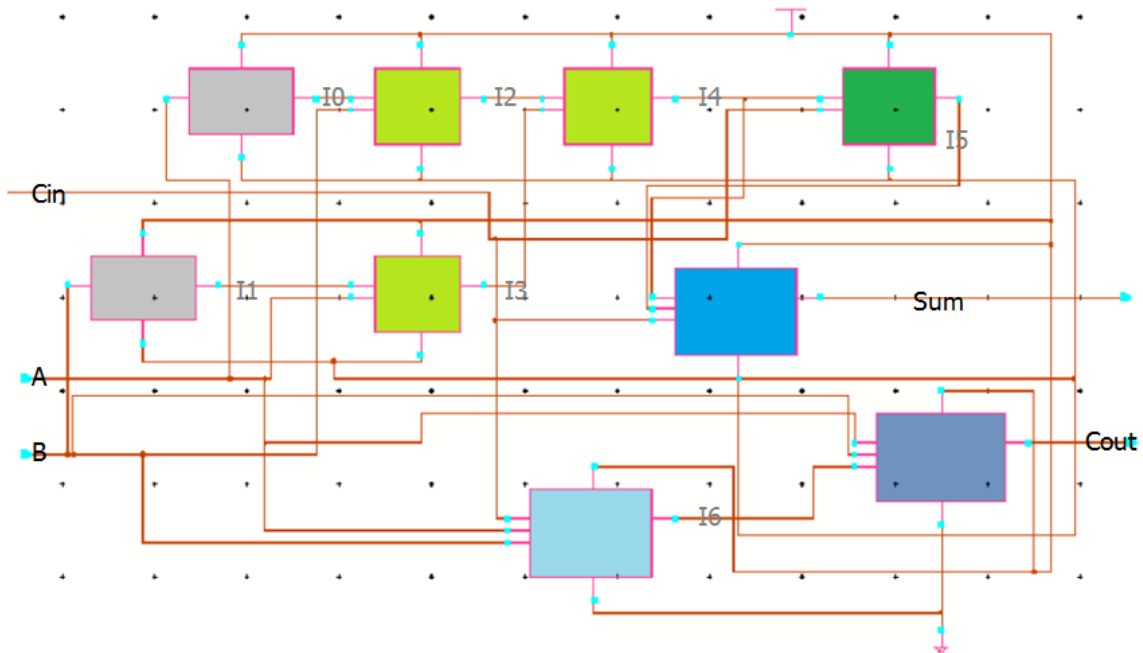


Figure 84: Unaltered full adder cell in the 4-bit ripple carry adder circuit

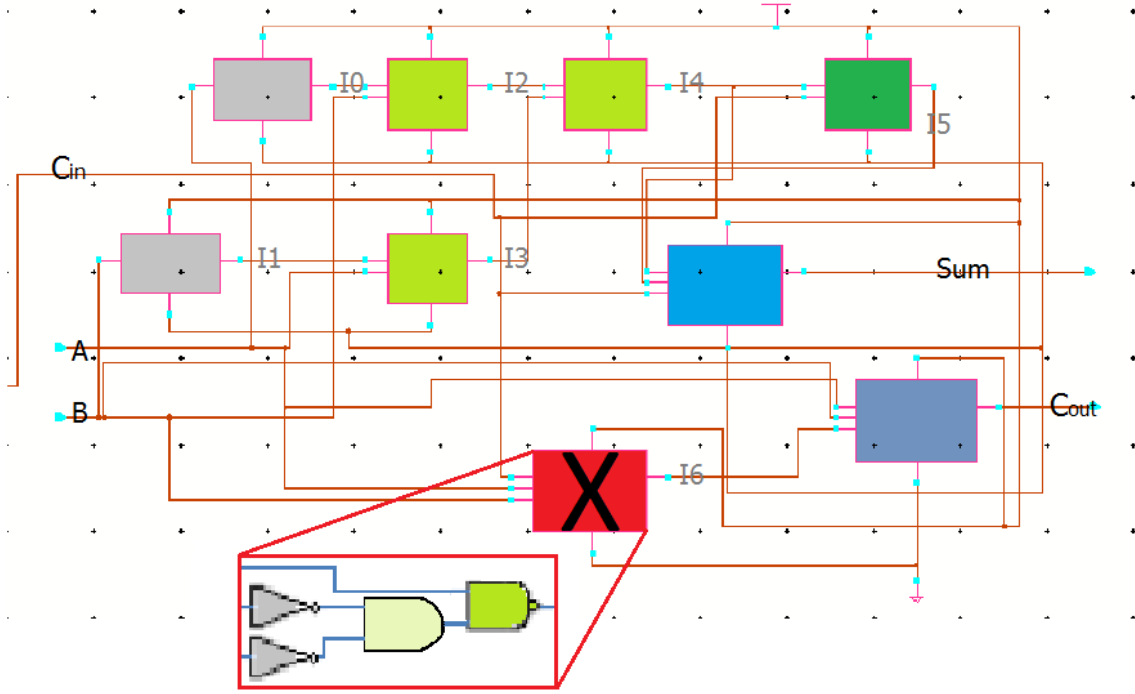


Figure 85: Altered full adder cell in the 4-bit ripple carry adder circuit. Composition of the maliciously altered OAI21 gate at the third level of abstraction is shown in the red box

Furthermore, the inputs ($A_0, B_0, A_1, B_1, A_2, B_2, A_3, B_3$, and C_{in3}), intermediate carry signals ($C_{out3}-C_{in2}, C_{out2}-C_{in1}, C_{out1}-C_{in0}$), and outputs (S_0, S_1, S_2, S_3 , and C_{out0}) of the unmodified 4-bit ripple carry adder and the modified 4-bit ripple carry adder are simulated in Figures 86 and 87, respectively. In these simulations, a value greater than or equal to 1V corresponds to a logical “1,” and a value less than 1V corresponds to a logical “0.” The signals C_{in3}, A_3, B_3, S_3 , and $C_{out3}-C_{in2}$ correspond to the fourth full adder cell, the signals $C_{out3}-C_{in2}, A_2, B_2, S_2$, and $C_{out2}-C_{in1}$ correspond to the third full adder cell, the signals $C_{out2}-C_{in1}, A_1, B_1, S_1$, and $C_{out1}-C_{in0}$ correspond to the second full adder cell, and the signals $C_{out1}-C_{in0}, A_0, B_0, S_0$, and C_{out0} correspond to the first full adder cell.

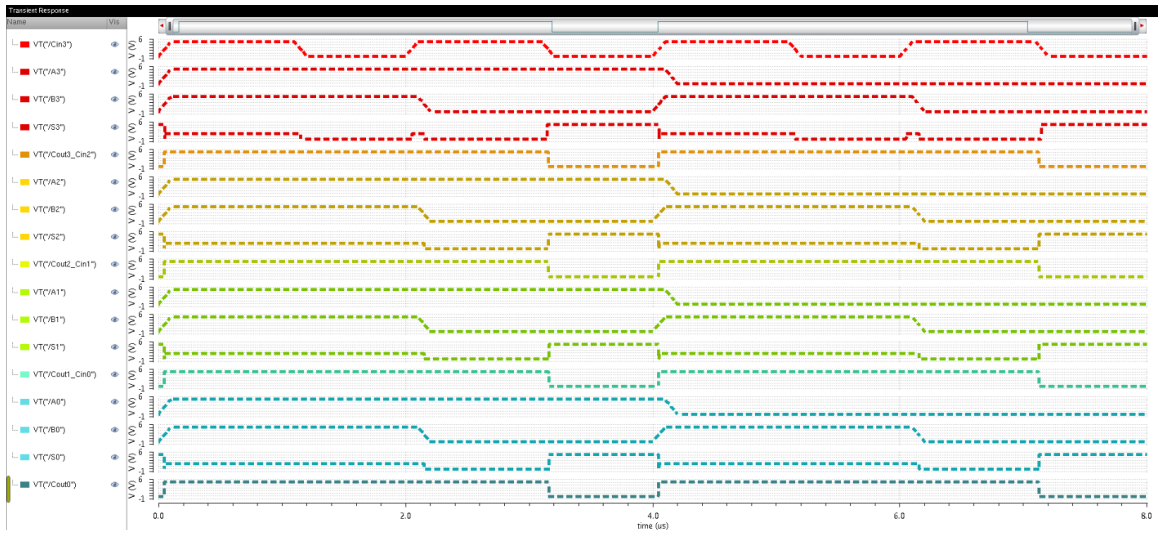


Figure 86: Simulation results of the unmodified 4-bit ripple carry adder

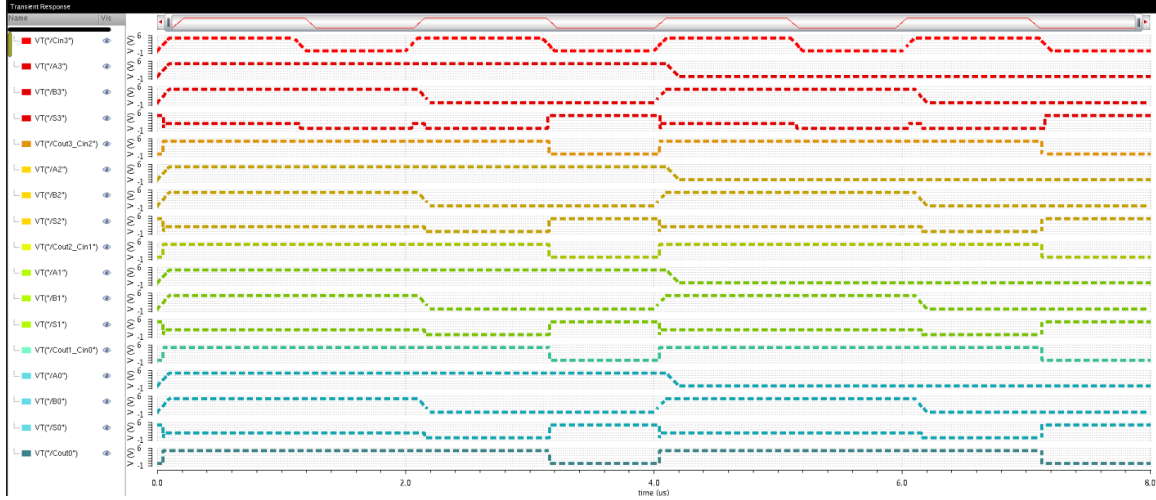


Figure 87: Simulation results of the modified (changed OAI21 gate composition in Full Adder Cell 0) 4-bit ripple carry adder

By comparing the simulations in Figures 86 and 87, it can be seen that no discernible difference exists between the two plots. Thus, it is clear that the alteration in Full Adder Cell 0 does not affect the logical output of the 4-bit ripple carry adder. Since the logical output is unaffected, functional testing would not identify the malicious modification.

However, transistor-level and gate-level testing with the implementation of the SCR algorithm is capable of identifying the change. The gate-level netlist for the unmodified 4-bit ripple carry adder is provided in Section D.2, and the transistor-level netlist for the maliciously modified 4-bit ripple carry adder is provided in Section D.3. The SCR algorithm developed in this research is applied to the transistor-level netlist for the maliciously modified 4-bit ripple carry adder, and the output gate-level netlist of the maliciously modified 4-bit ripple carry adder is provided in Section D.4.

It can be observed that the netlist in Section D.4 for the modified adder does not match the netlist in Section D.2 for the unmodified adder. Instead of identifying the four OAI21 gates that exist in the unmodified adder, the algorithm identifies three OAI21 gates. In the place of the fourth OAI21 gate (which has been maliciously modified), the

algorithm accurately identifies the malicious modification of two inverters, an OR2 gate, and a NAND2 gate.

4.3.2 *Malicious Switch in Gate Input Signals.*

The gate-level schematic of the unmodified 4-bit ripple carry adder is reproduced in Figure 88, and the gate-level schematic of the maliciously modified 4-bit ripple carry adder is shown in Figure 89. Figure 90 displays the composition of the maliciously modified 4-bit ripple carry adder at the third level of abstraction. The transistor-level schematic of the modified 4-bit ripple carry adder is presented in Figure 91. As seen by comparing Figures 88 and 89, two of the inputs into the OAI21 gate of the lower-right full adder cell (Full Adder Cell 0) have been switched. The switch is denoted by the red dashed circle on Figure 89.

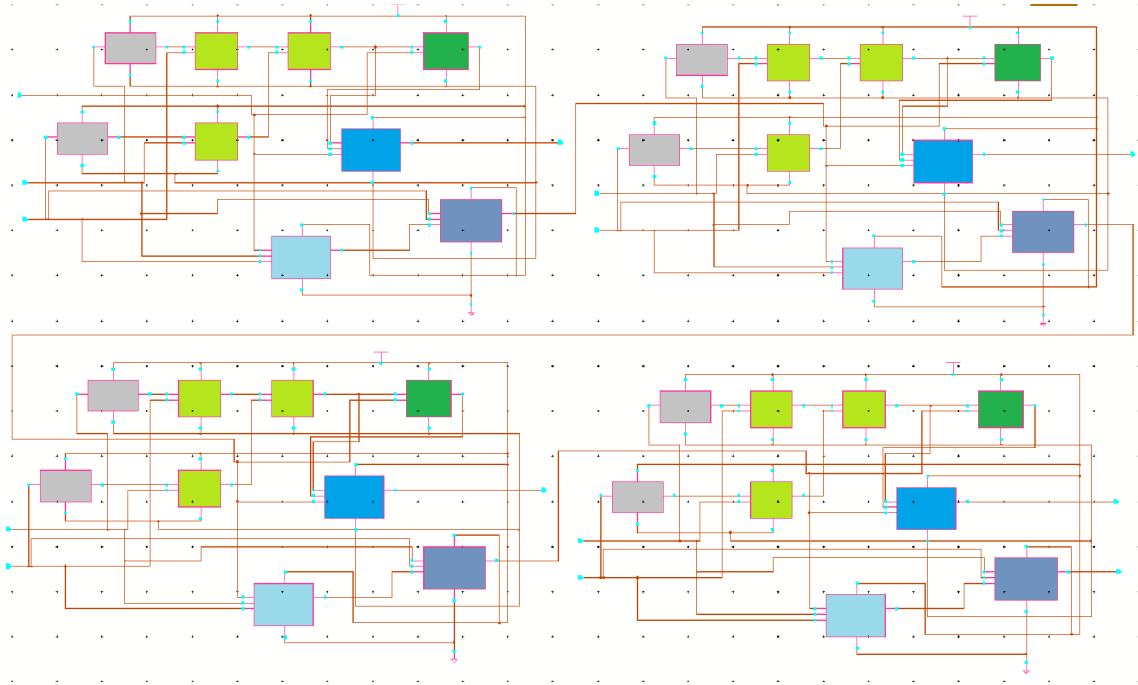


Figure 88: Gate-level representation of the unmodified 4-bit ripple carry adder circuit

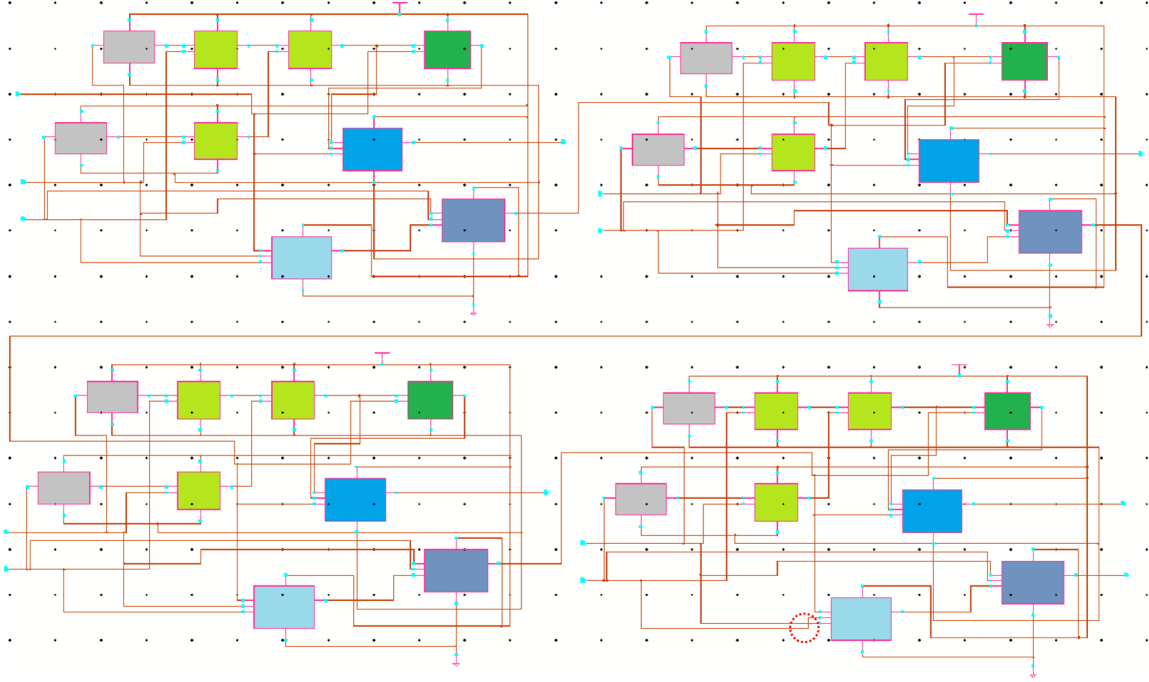


Figure 89: Gate-level representation of the maliciously modified (switched OAI21 gate inputs in Full Adder Cell 0) 4-bit ripple carry adder circuit

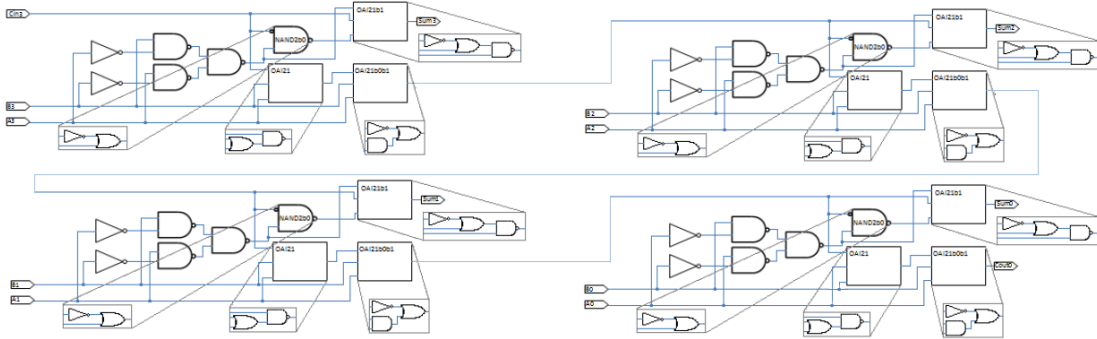


Figure 90: Maliciously modified (switched OAI21 gate inputs in Full Adder Cell 0) 4-bit ripple carry adder circuit. Compositions of gates at the fourth level of abstraction are shown in boxes

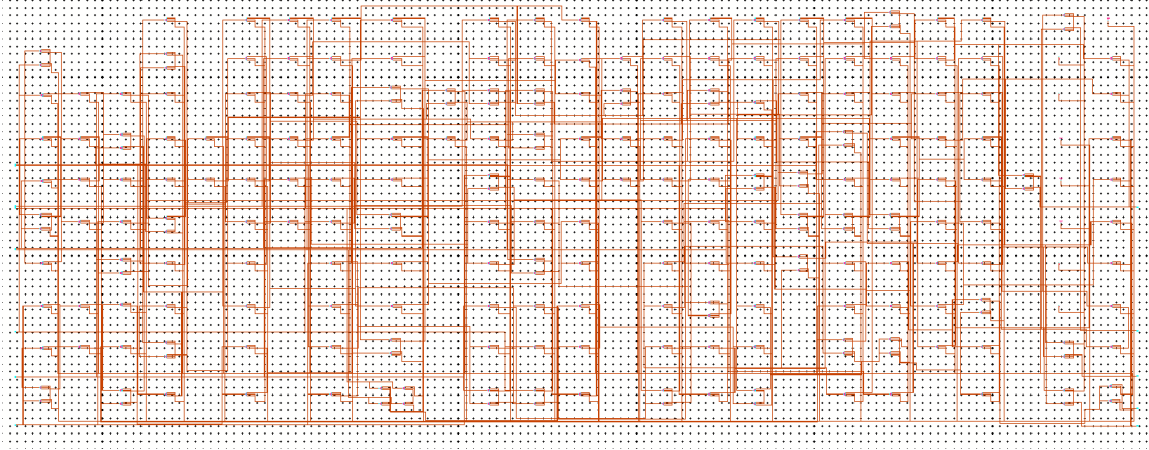


Figure 91: Transistor-level schematic of the maliciously modified (switched OAI21 gate inputs in Full Adder Cell 0) 4-bit ripple carry adder circuit

By referring back to Figure 78, it can be observed that the two inputs switched are the two inputs into the OR2 component of the OAI21 gate; hence, this modification does not affect the logic of the circuit. Since the logic remains unaffected, functional testing is incapable of identifying the modification. Transistor-level testing with SCR, however, is capable of recognizing the modification. The gate-level netlist for the unmodified 4-bit ripple carry adder is provided in Section D.2, and the transistor-level netlist for the maliciously modified 4-bit ripple carry adder is provided in Section D.5. The SCR algorithm developed in this research is applied to the transistor-level netlist for the maliciously modified 4-bit ripple carry adder, and the output gate-level netlist of the maliciously modified 4-bit ripple carry adder is provided in Section D.6. It can be observed that the netlist for the modified adder does not match the netlist for the unmodified adder. The original OAI21 gate in the unmodified Full Adder Cell 0 is represented in the netlist in Section D.2 as:

```
I8 (0 net01405 B0 A0 vdd! net01470) OAI21_type_0
```

The altered OAI21 gate in the maliciously modified Full Adder Cell 0 is represented in the netlist in Section D.6 as:

```
I8 (0 net01405 A0 B0 vdd! net01470) OAI21_type_0
```

By inspecting the OAI21 representations, it is clear that the signals A0 and B0 have been switched.

V. Conclusion and Future Work

THIS conclusion chapter serves three purposes. First, the chapter summarizes the information presented in this document. Second, the chapter describes future work that is necessary for further development of this research. Third, the chapter evaluates the impact of the research on IC verification applied to DoD systems. Each of the purposes is realized in the three sections of the chapter.

5.1 Summary

The DARPA TRUST program was established to address the need to verify integrated circuits to prevent counterfeit electronics from entering DoD systems. Incongruencies in abstraction levels from previous TRUST-related research involving gate-level verification of a nine-gate full adder cell prompted the need for an SCR technique to build a gate-level netlist from a flat transistor-level netlist. This thesis has presented the development of an algorithm implemented in Python code to conduct SCR on a transistor-level input netlist to generate a gate-level output netlist. The Python script developed in this research successfully resolves the incongruence in levels of abstraction with the full adder cell. The SCR code and algorithm developed in this research has demonstrated a 90% success rate of perfectly performing SCR on circuits scoped within 650 transistors and 78 gates of nine different types up to the fourth level of abstraction. Additionally, the SCR code has been successfully implemented using Python V2.6.6 on the Linux machines at the AFRL MSDC, the machines on which future development of the SCR code will occur. Thus, the SCR code and algorithm show considerable potential for resolving more complex circuits.

5.2 Future Work

Further development of the code is required to increase the success rate to 100% for the circuits tested in this research, to enable application to more complex circuits, to

facilitate successful integration with prior research, and to promote integration with a variety of environments.

Increase of Success Rate. Revisiting Section 4.1.4.1, “Level of Maturity,” the only circuit that failed to achieve perfect SCR results was test circuit Test1. In order to increase the success rate to 100% for the circuits tested in this research, more rules must be added to guide the gate identification algorithms for gates in the fourth level of abstraction, specifically to the NAND2b0 gate. Application of the algorithm to more complex circuits may require further development of the rules.

Application to Complex Circuits. In addition to developing the rules, enabling application of the algorithm to more complex circuits will require the following actions:

1. The types of gates and number of gate inputs the algorithms can identify must be expanded;
2. The levels of gate abstraction must be increased;
3. Information about circuit layout geometries, power distribution systems, and clock trees must be incorporated into the algorithm;
4. A method of determining the runtime must be implemented;
5. The code must be optimized to minimize the runtime;
6. The behavior of the output netlist must be tested.

The discussion of gate recognition comprehensiveness in Section 4.1.4.1, “Level of Maturity,” provides examples of other types of gate and gate inputs and varying levels of abstraction that the algorithm must be able to identify in order for successful application to more complex circuits. While it is sufficient to identify gates solely by transistor connections for simple circuits, the immense variation in gate configurations of complex circuits will necessitate the incorporation of information about circuit layout geometries,

power distribution systems, and clock trees into the algorithm to determine which sub-gates compose which gates. Additionally, the current simplicity of the circuits tested in the SCR research results in a negligible runtime; thus, determining the runtime was considered unnecessary. However, as more complex circuits are processed, the runtime of the code will increase. Thus, in order to accommodate increasing complexity, a method of determining the runtime must be created, and the code must be optimized to minimize the runtime. Furthermore, the accuracy of the output netlists produced by the algorithm have previously been evaluated by human inspection. As circuit complexity increases, human inspection will no longer suffice as an evaluation method. As a result, the behavior of the output netlist must be tested in order to evaluate its accuracy.

Integration with Prior Research. In order to enable successful integration with the TRUST-related research, it is important to note that the prior research generated the revised netlist (which is the input netlist for the algorithm) from a layout, as described in Section 3.2, “Phase 2 Methodology - Software Tool Application to Elementary Gates.” Therefore, future work will require generating the input netlist from a layout rather than a schematic. As a result of this action, the algorithm will require modification to account for parasitic capacitances and merged series transistors. Alternatively, a method could be determined to convert a layout netlist into a schematic netlist prior to using the schematic netlist as an input for the algorithm. If a conversion method is determined, no modification to the code is necessary.

Integration with Variety of Environments. The process of adjusting the algorithm to also accept input netlists generated from layouts will contribute to the ability of the code to integrate with a variety of circuit verification environments. Additionally, the code is presently written so that it only inputs/outputs netlists written in the Spectre language. Future work should enhance the code so that it can input/output in other netlist languages (Verilog, VHDL, etc.). Alternatively, a method could be determined to convert between

Spectre and other netlist languages. If a conversion method is determined, no modification to the code is necessary.

5.3 Conclusion

The algorithm developed and code written in this research provides a solid foundation for the development of the SCR methods deemed necessary by TRUST-related research to transform transistor-level netlists into gate-level netlists. Further development of this code will better enable the TRUST-related research to perform as intended in verifying integrated circuits. The impact of a fully functioning circuit verification process is a reduction in the funding expended in combating counterfeit electronics and in the risk posed to national security [6]. Thus, successful continuation of the code developed in this research will ultimately have an impact in ensuring both the physical and financial security of the United States of America.

Appendix A: NCSU Digital Parts Standard Cell Library

1. DFF
2. DFF_Clr
3. DFF_Pre
4. Dlatch
5. and2
6. and3
7. and4
8. and5
9. and6
10. inv
11. mux_2to1_1bit
12. mux_2to1_32bit
13. mux_2to1_80bit
14. mux_3to1_1bit
15. mux_3to1_32bit
16. mux_4to1_1bit
17. mux_4to1_20bit

- 18. mux_4to1_24bit
- 19. mux_4to1_32bit
- 20. mux_4to1_80bit
- 21. mux_8to1_1bit
- 22. mux_8to1_32bit
- 23. nand2
- 24. nand3
- 25. nand4
- 26. nand5
- 27. nand6
- 28. nor2
- 29. nor3
- 30. nor4
- 31. nor5
- 32. nor6
- 33. or2
- 34. or3
- 35. or4
- 36. or5

- 37. or6
- 38. trinv
- 39. trinv0
- 40. trinv1
- 41. tx_gate
- 42. xnor2
- 43. xnor3
- 44. xnor4
- 45. xnor5
- 46. xnor6
- 47. xor2
- 48. xor3
- 49. xor4
- 50. xor5
- 51. xor6

Appendix B: Cells included in TRUST Test Articles

Table 26: Cells included in the first TRUST test article

Cell Name	Cell Count
inv_1x	119
dff_1x	24
or2_1x	51
nor2_1x	136

Cell Name	Cell Count
add_1x1x	67
add_2x2x	34
add_4x4x	7
add_6x6x	14
add_full_cinb_2x	11
add_full_coutb_2x	12
addh_1x1x	32
addh_2x2x	2
addh_3x3x	1
addh_coutb_2x	1
and2_1x	154
and2_2x	66
and2_3x	33

Table 27 – continued from previous page

Cell Name	Cell Count
and2_4x	9
and2_6x	3
and3_2x	11
and3_3x	6
and3_6x	2
ao21_1x	4
ao21_4x	1
ao22_1x	5
aoa211_1x	1
aoai211_1x	1
aoi21_1x	294
aoi21_2x	63
aoi21_3x	117
aoi21_4x	8
aoi21_6x	17
aoi21_8x	2
aoi21_b0b1_1x	17
aoi21_b0b1_2x	5
aoi21_b1_1x	52
aoi21_b1_2x	12
aoi21_b2_1x	222
aoi21_b2_2x	29
aoi22_1x	389

Table 27 – continued from previous page

Cell Name	Cell Count
aoi22_2x	8
aoi22_3x	180
aoi22_4x	12
aoi22_6x	7
aoi22_b0b1_1x	8
aoi22_b0b1_2x	9
aoi222_1x	47
aoi222_2x	60
aoi222_4x	1
buf_10x	26
buf_12x	1
buf_14x	21
buf_16x	3
buf_1x	161
buf_20x	10
buf_24x	1
buf_2x	48
buf_30x	1
buf_3x	60
buf_4x	14
buf_5x	45
buf_6x	8
buf_8x	12

Table 27 – continued from previous page

Cell Name	Cell Count
delay1_1x	18
delay1_4x	13
delay2_4x	1
delay3_4x	1
delay4_1x	3
dff_1x	624
dff_2x	22
dff_3x	96
dff_6x	24
dff_cpq_3x	2
dff_q_1x	3
inv_10x	4
inv_12x	1
inv_16x	1
inv_1x	1059
inv_2x	151
inv_30x	23
inv_3x	205
inv_4x	40
inv_5x	25
inv_6x	52
inv_8x	30
latch_6x	1

Table 27 – continued from previous page

Cell Name	Cell Count
mux2_6x	1
mux2_8x	1
muxi2_2x	1
muxi2_3x	18
nand2_1x	627
nand2_2x	325
nand2_3x	38
nand2_4x	100
nand2_5x	307
nand2_6x	62
nand2_8x	4
nand2b0_1x	339
nand2b0_2x	171
nand2b0_4x	77
nand3_1x	168
nand3_2x	42
nand3_3x	3
nand3_8x	3
nand3b0_1x	73
nand3b0_2x	20
nand4_1x	5
nand4_2x	3
nand4b0_1x	1

Table 27 – continued from previous page

Cell Name	Cell Count
nand4b0b1_1x	1
nor2_1x	259
nor2_2x	91
nor2_3x	188
nor2_4x	3
nor2_5x	2
nor2_8x	5
nor2b0_1x	14
nor2b0_2x	42
nor2b0_4x	1
nor2b0_8x	1
nor3_1x	6
nor3_3x	1
nor3b0_1x	7
nor4_1x	21
nor4_2x	5
nor4b0_1x	2
nor4b0_2x	1
nor4b0b1_2x	1
oa21_1x	2
oa21_2x	3
oa21_3x	1
oa211_3x	4

Table 27 – continued from previous page

Cell Name	Cell Count
oai21_1x	378
oai21_2x	33
oai21_3x	186
oai21_4x	216
oai21_6x	45
oai21_8x	5
oai211_1x	3
oai211b1_1x	1
oai21b0b1_1x	18
oai21b0b1_2x	1
oai21b0b1_4x	4
oai21b1_1x	181
oai21b1_2x	3
oai21b1_4x	53
oai21b2_1x	53
oai21b2_2x	3
oai21b2_4x	58
oai22_1x	10
oai22_2x	1
oai22_3x	11
oai22_4x	3
oai22_6x	6
oai221_1x	1

Table 27 – continued from previous page

Cell Name	Cell Count
oai22b0b1_1x	23
oai22b0b1_4x	11
oai22b1_1x	473
oai22b1_4x	17
oai31_1x	1
oai31_2x	1
oaoi211_1x	4
or2_1x	27
or2_2x	12
or2_3x	1
or3_1x	1
tiehi	1
tielo	1
tribuf_30x	1
triinv_30x	1
triinv_6x	1
xnor2_1x	321
xnor2_2x	38
xnor2_3x	5
xnor2_4x	8
xnor3_1x	3
xnor3_2x	85
xor2_1x	146

Table 27 – continued from previous page

Cell Name	Cell Count
xor2_2x	52
xor2_3x	30
xor2_4x	4
xor3_1x	13
xor3_2x	100

Table 27: Cells included in the second TRUST test article

Cell Name	Cell Count
add_1x1x	92
add_2x2x	4
add_6x6x	1
add_full_cinb_2x	19
add_full_coutb_2x	18
addh_1x1x	11
addh_2x2x	1
addh_cinb_2x	22
addh_coutb_2x	24
and2_1x	523
and2_2x	157
and2_3x	40
and2_4x	2
and2_6x	1
and3_1x	5

Table 28 – continued from previous page

Cell Name	Cell Count
and3_2x	28
and3_3x	8
and3_4x	1
ao21_1x	2
ao21_2x	4
ao22_1x	11
ao22_2x	5
ao22_b_1x	2
ao222_1x	1
aoai211_1x	18
aoi21_1x	431
aoi21_2x	37
aoi21_3x	266
aoi21_4x	6
aoi21_6x	3
aoi21_8x	3
aoi21_b0b1_1x	3
aoi21_b1_1x	80
aoi21_b1_2x	4
aoi21_b1_4x	1
aoi21_b2_1x	239
aoi21_b2_2x	10
aoi22_1x	2150

Table 28 – continued from previous page

Cell Name	Cell Count
aoi22_2x	11
aoi22_3x	649
aoi22_4x	4
aoi22_b0b1_1x	19
aoi221_1x	11
aoi222_1x	18
aoi222_2x	38
aoi31_1x	1
buf_10x	22
buf_14x	13
buf_18x	1
buf_1x	133
buf_20x	7
buf_24x	1
buf_2x	22
buf_3x	130
buf_4x	12
buf_5x	40
buf_6x	6
buf_8x	23
delay1_1x	163
delay1_4x	3
dff_1x	2812

Table 28 – continued from previous page

Cell Name	Cell Count
dff_2x	38
dff_3x	106
dff_6x	55
dff_q_1x	35
dff_q_2x	2
dff_q_6x	3
inv_14x	1
inv_16x	1
inv_1x	1673
inv_2x	474
inv_30x	2
inv_3x	235
inv_4x	26
inv_5x	2
inv_6x	27
inv_8x	14
mux2_2x	2
muxi2_1x	2
muxi2_3x	2
nand2_1x	1185
nand2_2x	1217
nand2_3x	20
nand2_4x	48

Table 28 – continued from previous page

Cell Name	Cell Count
nand2_5x	295
nand2_6x	11
nand2_8x	6
nand2b0_1x	297
nand2b0_2x	445
nand2b0_4x	64
nand3_1x	644
nand3_2x	170
nand3_3x	3
nand3b0_1x	297
nand3b0_4x	1
nand4_1x	3
nand4_2x	10
nand4b0_1x	1
nor2_1x	270
nor2_2x	64
nor2_3x	427
nor2_4x	5
nor2_8x	1
nor2b0_1x	12
nor2b0_2x	88
nor2b0_8x	1
nor3_1x	12

Table 28 – continued from previous page

Cell Name	Cell Count
nor3_2x	5
nor3b0_1x	11
nor4_1x	42
nor4_2x	20
nor4_6x	1
nor4b0_1x	49
nor4b0_2x	23
nor4b0b1_1x	1
oa21_1x	4
oa21_2x	3
oa211_1x	1
oa211_3x	2
oa31_1x	6
oai21_1x	382
oai21_2x	9
oai21_3x	492
oai21_4x	179
oai21_6x	16
oai21_8x	8
oai211_1x	1
oai211_2x	1
oai211b1_1x	3
oai21b0b1_1x	22

Table 28 – continued from previous page

Cell Name	Cell Count
oai21b0b1_4x	1
oai21b1_1x	926
oai21b1_4x	22
oai21b2_1x	116
oai21b2_2x	1
oai21b2_4x	17
oai22_1x	16
oai22_3x	34
oai22_4x	1
oai22b0b1_1x	30
oai22b0b1_2x	1
oai22b0b1_4x	11
oai22b1_1x	2095
oai22b1_2x	2
oai22b1_4x	12
oai31_1x	2
oai31_2x	6
oai31_6x	1
oao211_1x	1
or2_1x	62
or2_2x	78
or2_3x	3
or2_8x	3

Table 28 – continued from previous page

Cell Name	Cell Count
or3_1x	1
or4_4x	1
xnor2_1x	224
xnor2_2x	146
xnor2_3x	2
xnor2_4x	2
xnor3_1x	9
xnor3_2x	104
xor2_1x	107
xor2_2x	56
xor2_3x	2
xor2_4x	2
xor3_1x	5
xor3_2x	45

Table 28: Cells included in the third TRUST test article

Cell Name	Cell Count
add_1x1x	41
add_full_cinb_2x	126
add_full_coutb_2x	118
addh_1x1x	112
addh_cinb_2x	36
addh_coutb_2x	34

Table 29 – continued from previous page

Cell Name	Cell Count
and2_1x	1998
and3_1x	20
and3_2x	62
and3_3x	90
ao21_1x	2
ao22_1x	6
aoai211_1x	6
aoi21_1x	4825
aoi21_b0b1_1x	11
aoi21_b1_1x	322
aoi21_b2_1x	557
aoi22_1x	9551
aoi22_b0b1_1x	3155
aoi221_1x	4
aoi222_1x	217
aoi31_1x	1
buf_1x	353
buf_2x	7
buf_30x	7
buf_3x	24
buf_4x	11
buf_5x	1
buf_6x	6

Table 29 – continued from previous page

Cell Name	Cell Count
compress_42_1x	282
delay1_1x	10
delay2_1x	23
delay3_1x	52
delay3_4x	21
delay4_1x	346
delay4_4x	1703
dff_1x	170
dff_q_1x	1
inv_1x	2915
inv_30x	205
inv_3x	4
mux2_1x	41
mux3_1x	15
muxi2_1x	7
muxi2_3x	9
nand2_1x	5971
nand2_5x	182
nand2b0_1x	1522
nand3_1x	2423
nand3b0_1x	412
nand4_1x	8
nor2_1x	1147

Table 29 – continued from previous page

Cell Name	Cell Count
nor2b0_1x	811
nor2b0_2x	42
nor3_1x	20
nor3b0_1x	3
nor4_1x	243
nor4_2x	32
nor4b0_1x	101
nor4b0_2x	15
nor4b0b1_1x	4
nor4b0b1_2x	2
oa21_1x	7
oa211_1x	1
oa211_3x	1
oa31_1x	2
oai21_1x	3399
oai21_4x	12
oai211_1x	37
oai211b1_1x	2
oai21b0b1_1x	76
oai21b1_1x	3592
oai21b2_1x	859
oai22_1x	130
oai22b0b1_1x	32

Table 29 – continued from previous page

Cell Name	Cell Count
oai22b1_1x	1890
oai31_1x	13
oao211_1x	3
oaoi211_1x	4
or2_1x	96
or3_1x	2
or4_1x	2
or4_4x	1
ram128x20	1
ram128x256	1
scandff_1x	8859
scandff_q_1x	1320
tiehi	2
tielo	4
tribuf_12x	1
triinv_12x	1
xnor2_1x	1408
xnor3_1x	63
xor2_1x	472
xor3_1x	44
xor3_2x	1

Table 29: Cells included in the fourth TRUST test article

Cell Name	Cell Count
add_1x1x	88
add_full_cinb_2x	1680
add_full_coutb_2x	1768
addh_1x1x	89
addh_cinb_2x	152
addh_coutb_2x	157
and2_1x	7523
and2_2x	421
and2_3x	1241
and2_4x	195
and2_6x	412
and2_8x	7
and3_1x	70
and3_2x	141
and3_3x	346
and3_4	6
ao22_1x	16
aoi21_1x	80314
aoi21_2x	2
aoi21_3x	211
aoi21_4x	11
aoi21_6x	3
aoi21_8x	1
aoi21_b0b1_1x	72

Table 30 – continued from previous page

Cell Name	Cell Count
aoi21_b0b1_2x	4
aoi21_b1_1x	832
aoi21_b1_2x	16
aoi21_b1_4x	3
aoi21_b2_1x	918
aoi21_b2_2x	16
aoi21_b2_4x	4
aoi22_1x	43574
aoi22_3x	91
aoi22_4x	46
aoi22_6x	5
aoi22_b0b1_1x	18120
aoi22_b0b1_2x	38
buf_10x	172
buf_12x	8
buf_14x	85
buf_16x	68
buf_18x	42
buf_1x	1180
buf_20x	90
buf_24x	94
buf_2x	247
buf_30x	85

Table 30 – continued from previous page

Cell Name	Cell Count
buf_3x	7208
buf_4x	1763
buf_5x	232
buf_6x	2915
buf_8x	1075
compress_42_1x	1128
delay1_1x	1
dff_1x	2358
dff_2x	40
dff_3x	10
dff_6x	10
dff_q_1x	4
inv_10x	1667
inv_12x	1
inv_14x	6
inv_18x	4
inv_1x	12102
inv_20x	2
inv_24x	9
inv_2x	324
inv_30x	134
inv_3x	2013
inv_4x	795

Table 30 – continued from previous page

Cell Name	Cell Count
inv_5x	196
inv_6x	506
inv_8x	612
mux2_1x	1
nand2_1x	33870
nand2_2x	64
nand2_3x	6
nand2_4x	10
nand2_5x	605
nand2_6x	64
nand2_8x	368
nand2b0_1x	8652
nand2b0_2x	10
nand2b0_4x	141
nand2b0_8x	18
nand3_1x	13767
nand3_2x	18
nand3_3x	30
nand3_4x	8
nand3_8x	1
nand3b0_1x	1589
nand3b0_2x	18
nand3b0_4x	136

Table 30 – continued from previous page

Cell Name	Cell Count
nor2_1x	730
nor2_2x	32
nor2_3x	29
nor2_4x	13
nor2_8x	2
nor2b0_1x	10746
nor2b0_2x	4432
nor2b0_4x	17
nor3_1x	58
nor3b0_1x	16
nor3b0_2x	3
nor4_1x	1183
nor4_2x	212
nor4b0_1x	783
nor4b0_2x	361
nor4b0b1_1x	123
oa21_1x	24
oa21_3x	4
oa211_3x	4
oa31_1x	1
oai21_1x	8599
oai21_2x	19
oai21_3x	132

Table 30 – continued from previous page

Cell Name	Cell Count
oai21_4x	1022
oai21_6x	26
oai21_8x	28
oai211_1x	40
oai21b0b1_1x	88
oai21b0b1_2x	1
oai21b0b1_4x	4
oai21b1_1x	27465
oai21b1_2x	905
oai21b1_4x	1621
oai21b2_1x	356
oai21b2_2x	11
oai21b2_4x	30
oai22_1x	760
oai22_4x	1
oai22b0b1_1x	173
oai22b1_1x	6262
oai22b1_4x	6
oai31_1x	16
or2_1x	269
or2_2x	7
or2_3x	3
or2_4x	1

Table 30 – continued from previous page

Cell Name	Cell Count
or2_6x	3
or4_1x	4
or4_4x	4
ram1080x1024	8
scandff_1x	67582
scandff_q_1x	6995
tiehi	37
tielo	77
xnor2_1x	4024
xnor2_2x	2
xnor3_1x	154
xnor3_2x	4
xor2_1x	403
xor2_2x	1
xor3_1x	101
xor3_2x	4

Table 30: Cells included in the fifth TRUST test article

Appendix C: SCR Code

```
1
2 # Copyright 2015 by Leleia A. Hsia, Second Lieutenant, ↵
   USAF
3 # Developed for the thesis GATE-LEVEL COMMERCIAL ↵
   MICROELECTRONICS VERIFICATION WITH STANDARD CELL ↵
   RECOGNITION
4 # Version 1.0.4 2/9/2015
5
6 import shutil
7 import os
8
9 #####CLASSES#####
10
11 class NMOS:
12     'This class creates an instance of all NMOS ↵
       transistors in netlist'
13
14     def __init__(self, string, string2):
15         self.string = string
16         self.string2 = string2
17
18     def width(self):
19         NMOS_width = self.string[(self.string.find("w=")+↵
           len("w=")):self.string.find("l")]
20         return str(NMOS_width)
21
22     def length(self):
23         NMOS_length = self.string[(self.string.find("l=")+↵
           len("l=")):self.string.find("as")]
24         return str(NMOS_length)
25
26     def net(self):
27         new_string = self.string[(self.string.find("(")+↵
           len("(")):self.string.find(")")]
28         NMOS_net = new_string.split()
29         return NMOS_net
30
31     def lines(self):
32         NMOS_lines = [self.string, self.string2]
33         return NMOS_lines
```

```

34
35 class PMOS:
36     'This class creates an instance of all PMOS ↵
        transistors in netlist'
37
38     def __init__(self, string, string2):
39         self.string = string
40         self.string2 = string2
41
42     def width(self):
43         PMOS_width = self.string[(self.string.find("w=")+↵
            len("w=")):self.string.find("l")]
44         return str(PMOS_width)
45
46     def length(self):
47         PMOS_length = self.string[(self.string.find("l=")+↵
            len("l=")):self.string.find("as")]
48         return str(PMOS_length)
49
50     def net(self):
51         new_string = self.string[(self.string.find("(")+↵
            len("(")):self.string.find(")")]
52         PMOS_net = new_string.split()
53         return PMOS_net
54
55     def lines(self):
56         PMOS_lines = [self.string, self.string2]
57         return PMOS_lines
58
59 class inverter:
60     'This class creates an instance of inverters'
61
62     def __init__(self, NMOS_object, PMOS_object):
63         self.NMOS_object = NMOS_object
64         self.PMOS_object = PMOS_object
65         self.inverter_type = 0
66
67     def __call__(self):
68         return self
69
70     def NMOS_width(self):
71         NMOS_width = self.NMOS_object.width()
72         return NMOS_width

```

```

73
74     def PMOS_width(self):
75         PMOS_width = self.PMOS_object.width()
76         return PMOS_width
77
78     def widths(self):
79         widths = [self.NMOS_object.width(), self.↵
80                 PMOS_object.width()]
81         return widths
82
83     def NMOS_length(self):
84         NMOS_length = self.NMOS_object.length()
85         return NMOS_length
86
87     def PMOS_length(self):
88         PMOS_length = self.PMOS_object.length()
89         return PMOS_length
90
91     def lengths(self):
92         lengths = [self.NMOS_object.length(), self.↵
93                 PMOS_object.length()]
94         return lengths
95
96     def NMOS_net(self):
97         NMOS_net = self.NMOS_object.net()
98         return NMOS_net
99
100    def PMOS_net(self):
101        PMOS_net = self.PMOS_object.net()
102        return PMOS_net
103
104    def nets(self):
105        nets = [self.NMOS_object.net(), self.PMOS_object.↵
106                net()]
107        return nets
108
109    def NMOS_lines(self):
110        NMOS_lines = self.NMOS_object.lines()
111        return NMOS_lines
112
113    def PMOS_lines(self):
114        PMOS_lines = self.PMOS_object.lines()
115        return PMOS_lines

```

```

113
114     def lines(self):
115         lines = [self.NMOS_object.lines(), self.↵
                PMOS_object.lines()]
116         return lines
117
118
119 class inverter_type:
120     'This class creates an instance of an inverter type'
121
122     def __init__(self, inverter_object):
123         self.inverter_object = inverter_object
124         self.inverter_type = 0
125
126     def __call__(self):
127         return self.inverter_object
128
129     def __getitem__(self):
130         return self.inverter_object
131
132     def inverter_object(self):
133         return self.inverter_object
134
135 class NAND2:
136     'This class creates an instance of NAND2s'
137
138     def __init__(self, NMOS_object_0, NMOS_object_1, ↵
                PMOS_object_0, PMOS_object_1):
139         self.NMOS_object_0 = NMOS_object_0
140         self.PMOS_object_0 = PMOS_object_0
141         self.NMOS_object_1 = NMOS_object_1
142         self.PMOS_object_1 = PMOS_object_1
143         self.NAND2_type = 0
144
145     def __call__(self):
146         return self
147
148     def NMOS_0_width(self):
149         NMOS_0_width = self.NMOS_object_0.width()
150         return NMOS_width
151
152     def NMOS_1_width(self):
153         NMOS_1_width = self.NMOS_object_1.width()

```

```

154         return NMOS_width
155
156     def PMOS_0_width(self):
157         PMOS_0_width = self.PMOS_object_0.width()
158         return PMOS_width
159
160     def PMOS_1_width(self):
161         PMOS_1_width = self.PMOS_object_1.width()
162         return PMOS_width
163
164     def widths(self):
165         widths = [self.NMOS_object_0.width(), self.NMOS_object_1.width(), self.PMOS_object_0.width(), self.PMOS_object_1.width()]
166         return widths
167
168     def NMOS_0_length(self):
169         NMOS_0_length = self.NMOS_object_0.length()
170         return NMOS_0_length
171
172     def NMOS_1_length(self):
173         NMOS_1_length = self.NMOS_object_1.length()
174         return NMOS_1_length
175
176     def PMOS_0_length(self):
177         PMOS_0_length = self.PMOS_object_0.length()
178         return PMOS_0_length
179
180     def PMOS_1_length(self):
181         PMOS_1_length = self.PMOS_object_1.length()
182         return PMOS_1_length
183
184     def lengths(self):
185         lengths = [self.NMOS_object_0.length(), self.NMOS_object_1.length(), self.PMOS_object_0.length(), self.PMOS_object_1.length()]
186         return lengths
187
188     def NMOS_0_net(self):
189         NMOS_0_net = self.NMOS_object_0.net()
190         return NMOS_0_net
191
192     def NMOS_1_net(self):

```

```

193         NMOS_1_net = self.NMOS_object_1.net()
194         return NMOS_1_net
195
196     def PMOS_0_net(self):
197         PMOS_0_net = self.PMOS_object_0.net()
198         return PMOS_0_net
199
200     def PMOS_1_net(self):
201         PMOS_1_net = self.PMOS_object_1.net()
202         return PMOS_1_net
203
204     def nets(self):
205         nets = [self.NMOS_object_0.net(), self.↵
                NMOS_object_1.net(), self.PMOS_object_0.net(), ↵
                self.PMOS_object_1.net()]
206         return nets
207
208     def NMOS_0_lines(self):
209         NMOS_0_lines = self.NMOS_object_0.lines()
210         return NMOS_0_lines
211
212     def NMOS_1_lines(self):
213         NMOS_1_lines = self.NMOS_object_1.lines()
214         return NMOS_1_lines
215
216     def PMOS_0_lines(self):
217         PMOS_0_lines = self.PMOS_object_0.lines()
218         return PMOS_0_lines
219
220     def PMOS_1_lines(self):
221         PMOS_1_lines = self.PMOS_object_1.lines()
222         return PMOS_1_lines
223
224     def lines(self):
225         lines = [self.NMOS_object_0.lines(), self.↵
                NMOS_object_1.lines(), self.PMOS_object_0.lines↵
                (), self.PMOS_object_1.lines()]
226         return lines
227
228
229 class NAND2_type:
230     'This class creates an instance of a NAND2 type'
231

```

```

232     def __init__(self, NAND2_object):
233         self.NAND2_object = NAND2_object
234         self.NAND2_type = 0
235
236     def __call__(self):
237         return self.NAND2_object
238
239     def __getitem__(self):
240         return self.NAND2_object
241
242     def NAND2_object(self):
243         return self.NAND2_object
244
245 class NOR2:
246     'This class creates an instance of NOR2s'
247
248     def __init__(self, NMOS_object_0, NMOS_object_1, ↵
249         PMOS_object_0, PMOS_object_1):
250         self.NMOS_object_0 = NMOS_object_0
251         self.PMOS_object_0 = PMOS_object_0
252         self.NMOS_object_1 = NMOS_object_1
253         self.PMOS_object_1 = PMOS_object_1
254         self.NOR2_type = 0
255
256     def __call__(self):
257         return self
258
259     def NMOS_0_width(self):
260         NMOS_0_width = self.NMOS_object_0.width()
261         return NMOS_width
262
263     def NMOS_1_width(self):
264         NMOS_1_width = self.NMOS_object_1.width()
265         return NMOS_width
266
267     def PMOS_0_width(self):
268         PMOS_0_width = self.PMOS_object_0.width()
269         return PMOS_width
270
271     def PMOS_1_width(self):
272         PMOS_1_width = self.PMOS_object_1.width()
273         return PMOS_width

```

```

274     def widths(self):
275         widths = [self.NMOS_object_0.width(), self.NMOS_object_1.width(), self.PMOS_object_0.width(), self.PMOS_object_1.width()]
276         return widths
277
278     def NMOS_0_length(self):
279         NMOS_0_length = self.NMOS_object_0.length()
280         return NMOS_0_length
281
282     def NMOS_1_length(self):
283         NMOS_1_length = self.NMOS_object_1.length()
284         return NMOS_1_length
285
286     def PMOS_0_length(self):
287         PMOS_0_length = self.PMOS_object_0.length()
288         return PMOS_0_length
289
290     def PMOS_1_length(self):
291         PMOS_1_length = self.PMOS_object_1.length()
292         return PMOS_1_length
293
294     def lengths(self):
295         lengths = [self.NMOS_object_0.length(), self.NMOS_object_1.length(), self.PMOS_object_0.length(), self.PMOS_object_1.length()]
296         return lengths
297
298     def NMOS_0_net(self):
299         NMOS_0_net = self.NMOS_object_0.net()
300         return NMOS_0_net
301
302     def NMOS_1_net(self):
303         NMOS_1_net = self.NMOS_object_1.net()
304         return NMOS_1_net
305
306     def PMOS_0_net(self):
307         PMOS_0_net = self.PMOS_object_0.net()
308         return PMOS_0_net
309
310     def PMOS_1_net(self):
311         PMOS_1_net = self.PMOS_object_1.net()
312         return PMOS_1_net

```

```

313
314     def nets(self):
315         nets = [self.NMOS_object_0.net(), self.↵
                 NMOS_object_1.net(), self.PMOS_object_0.net(), ↵
                 self.PMOS_object_1.net()]
316         return nets
317
318     def NMOS_0_lines(self):
319         NMOS_0_lines = self.NMOS_object_0.lines()
320         return NMOS_0_lines
321
322     def NMOS_1_lines(self):
323         NMOS_1_lines = self.NMOS_object_1.lines()
324         return NMOS_1_lines
325
326     def PMOS_0_lines(self):
327         PMOS_0_lines = self.PMOS_object_0.lines()
328         return PMOS_0_lines
329
330     def PMOS_1_lines(self):
331         PMOS_1_lines = self.PMOS_object_1.lines()
332         return PMOS_1_lines
333
334     def lines(self):
335         lines = [self.NMOS_object_0.lines(), self.↵
                 NMOS_object_1.lines(), self.PMOS_object_0.lines↵
                 (), self.PMOS_object_1.lines()]
336         return lines
337
338
339 class NOR2_type:
340     'This class creates an instance of a NOR2 type'
341
342     def __init__(self, NOR2_object):
343         self.NOR2_object = NOR2_object
344         self.NOR2_type = 0
345
346     def __call__(self):
347         return self.NOR2_object
348
349     def __getitem__(self):
350         return self.NOR2_object
351

```

```

352     def NOR2_object(self):
353         return self.NOR2_object
354
355
356 class OR2:
357     'This class creates an instance of OR2s'
358
359     def __init__(self, NOR2_object, inverter_object):
360         self.NOR2_object = NOR2_object
361         self.inverter_object = inverter_object
362         self.OR2_type = 0
363
364     def __call__(self):
365         return self
366
367     def widths(self):
368         widths = [(self.NOR2_object).widths(), (self.↵
369                 inverter_object).widths())
370         return widths
371
372     def lengths(self):
373         lengths = [(self.NOR2_object).lengths(), (self.↵
374                 inverter_object).lengths())
375         return lengths
376
377     def nets(self):
378         nets = [(self.NOR2_object).nets(), (self.↵
379                 inverter_object).nets())
380         return nets
381
382     def lines(self):
383         lines = [(self.NOR2_object).lines(), (self.↵
384                 inverter_object).lines())
385         return lines
386
387 class OR2_type:
388     'This class creates an instance of an OR2 type'
389
390     def __init__(self, OR2_object):
391         self.OR2_object = OR2_object
392         self.OR2_type = 0

```

```

391     def __call__(self):
392         return self.OR2_object
393
394     def __getitem__(self):
395         return self.OR2_object
396
397     def OR2_object(self):
398         return self.OR2_object
399
400 class AND2:
401     'This class creates an instance of AND2s'
402
403     def __init__(self, NAND2_object, inverter_object):
404         self.NAND2_object = NAND2_object
405         self.inverter_object = inverter_object
406         self.AND2_type = 0
407
408     def __call__(self):
409         return self
410
411     def widths(self):
412         widths = [(self.NAND2_object).widths(), (self.↵
413                 inverter_object).widths())]
414         return widths
415
416     def lengths(self):
417         lengths = [(self.NAND2_object).lengths(), (self.↵
418                 inverter_object).lengths())]
419         return lengths
420
421     def nets(self):
422         nets = [(self.NAND2_object).nets(), (self.↵
423                 inverter_object).nets())]
424         return nets
425
426     def lines(self):
427         lines = [(self.NAND2_object).lines(), (self.↵
428                 inverter_object).lines())]
429         return lines
430
431 class AND2_type:
432     'This class creates an instance of an AND2 type'

```

```

430
431     def __init__(self, AND2_object):
432         self.AND2_object = AND2_object
433         self.AND2_type = 0
434
435     def __call__(self):
436         return self.AND2_object
437
438     def __getitem__(self):
439         return self.AND2_object
440
441     def AND2_object(self):
442         return self.AND2_object
443
444
445
446 class NAND2b0:
447     'This class creates an instance of NAND2b0s'
448
449     def __init__(self, inverter_object, OR2_object):
450         self.OR2_object = OR2_object
451         self.inverter_object = inverter_object
452         self.NAND2b0_type = 0
453
454     def __call__(self):
455         return self
456
457     def widths(self):
458         widths = [(self.inverter_object).widths(), (self.↵
459                     OR2_object).widths()])
460         return widths
461
462     def lengths(self):
463         lengths = [(self.inverter_object).lengths(), (self.↵
464                     .OR2_object).lengths()])
465         return lengths
466
467     def nets(self):
468         nets = [(self.inverter_object).nets(), (self.↵
469                     OR2_object).nets()])
470         return nets
471
472     def lines(self):

```

```

470         lines = [(self.inverter_object).lines(), (self.↵
471                 OR2_object).lines()]
472         return lines
473
474
475 class NAND2b0_type:
476     'This class creates an instance of a NAND2b0 type'
477
478     def __init__(self, NAND2b0_object):
479         self.NAND2b0_object = NAND2b0_object
480         self.NAND2b0_type = 0
481
482     def __call__(self):
483         return self.NAND2b0_object
484
485     def __getitem__(self):
486         return self.NAND2b0_object
487
488     def NAND2b0_object(self):
489         return self.NAND2b0_object
490
491
492
493 class OAI21:
494     'This class creates an instance of OAI21s'
495
496     def __init__(self, OR2_object, NAND2_object):
497         self.OR2_object = OR2_object
498         self.NAND2_object = NAND2_object
499         self.OAI21_type = 0
500
501     def __call__(self):
502         return self
503
504     def widths(self):
505         widths = [(self.OR2_object).widths(), (self.↵
506                 NAND2_object).widths()]
507         return widths
508
509     def lengths(self):
510         lengths = [(self.OR2_object).lengths(), (self.↵
511                 NAND2_object).lengths()]

```

```

510         return lengths
511
512     def nets(self):
513         nets = [(self.OR2_object).nets(), (self.↵
514                 NAND2_object).nets())
515         return nets
516
517     def lines(self):
518         lines = [(self.OR2_object).lines(), (self.↵
519                 NAND2_object).lines())
520         return lines
521
522 class OAI21_type:
523     'This class creates an instance of an OAI21 type'
524
525     def __init__(self, OAI21_object):
526         self.OAI21_object = OAI21_object
527         self.OAI21_type = 0
528
529     def __call__(self):
530         return self.OAI21_object
531
532     def __getitem__(self):
533         return self.OAI21_object
534
535     def OAI21_object(self):
536         return self.OAI21_object
537
538 class OAI21b1:
539     'This class creates an instance of OAI21b1s'
540
541     def __init__(self, inverter_object, OR2_object, ↵
542                 NAND2_object):
543         self.inverter_object = inverter_object
544         self.OR2_object = OR2_object
545         self.NAND2_object = NAND2_object
546         self.OAI21b1_type = 0
547
548     def __call__(self):
549         return self
550
551     def widths(self):

```

```

550         widths = [(self.inverter_object).widths(), (self.↵
                    OR2_object).widths(), (self.NAND2_object).↵
                    widths()]
551         return widths
552
553     def lengths(self):
554         lengths = [(self.inverter_object).lengths(), (self.↵
                    .OR2_object).lengths(), (self.NAND2_object).↵
                    lengths()]
555         return lengths
556
557     def nets(self):
558         nets = [(self.inverter_object).nets(), (self.↵
                    OR2_object).nets(), (self.NAND2_object).nets()]
559         return nets
560
561     def lines(self):
562         lines = [(self.inverter_object).lines(), (self.↵
                    OR2_object).lines(), (self.NAND2_object).lines↵
                    ()]
563         return lines
564
565 class OAI21b1_type:
566     'This class creates an instance of an OAI21b1 type'
567
568     def __init__(self, OAI21b1_object):
569         self.OAI21b1_object = OAI21b1_object
570         self.OAI21b1_type = 0
571
572     def __call__(self):
573         return self.OAI21b1_object
574
575     def __getitem__(self):
576         return self.OAI21b1_object
577
578     def OAI21b1_object(self):
579         return self.OAI21b1_object
580
581
582 class OAI21b0b1:
583     'This class creates an instance of OAI21b0b1s'
584

```

```

585     def __init__(self, inverter_object, AND2_object, ←
        OR2_object):
586         self.inverter_object = inverter_object
587         self.OR2_object = OR2_object
588         self.AND2_object = AND2_object
589         self.OAI21b0b1_type = 0
590
591     def __call__(self):
592         return self
593
594     def widths(self):
595         widths = [(self.inverter_object).widths(), (self.←
            AND2_object).widths(), (self.OR2_object).widths(←
                )]
596         return widths
597
598     def lengths(self):
599         lengths = [(self.inverter_object).lengths(), (self.←
            .AND2_object).lengths(), (self.OR2_object).←
                lengths()]
600         return lengths
601
602     def nets(self):
603         nets = [(self.inverter_object).nets(), (self.←
            AND2_object).nets(), (self.OR2_object).nets()]
604         return nets
605
606     def lines(self):
607         lines = [(self.inverter_object).lines(), (self.←
            AND2_object).lines(), (self.OR2_object).lines()←
                ]
608         return lines
609
610 class OAI21b0b1_type:
611     'This class creates an instance of an OAI21b0b1 type'
612
613     def __init__(self, OAI21b0b1_object):
614         self.OAI21b0b1_object = OAI21b0b1_object
615         self.OAI21b0b1_type = 0
616
617     def __call__(self):
618         return self.OAI21b0b1_object
619

```

```

620     def __getitem__(self):
621         return self.OAI21b0b1_object
622
623     def OAI21b0b1_object(self):
624         return self.OAI21b0b1_object
625
626
627 #####FUNCTIONS#####
628
629 ##Initial global array and variable initializations
630 NMOSIx = []
631 PMOSIx = []
632 NMOSIx_original = []
633 PMOSIx_original = []
634 inverters = []
635 inverter_types = []
636 removed_inverters_for_type_finding = []
637 NAND2s = []
638 removed_NAND2s_for_type_finding = []
639 NAND2_types = []
640 AND2s = []
641 removed_AND2s_for_type_finding = []
642 AND2_types = []
643 NOR2s = []
644 removed_NOR2s_for_type_finding = []
645 NOR2_types = []
646 OR2s = []
647 removed_OR2s_for_type_finding = []
648 OR2_types = []
649 NAND2b0s = []
650 removed_NAND2b0s_for_type_finding = []
651 NAND2b0_types = []
652 OAI21s = []
653 removed_OAI21s_for_type_finding = []
654 OAI21_types = []
655 OAI21b1s = []
656 removed_OAI21b1s_for_type_finding = []
657 OAI21b1_types = []
658 OAI21b0b1s = []
659 removed_OAI21b0b1s_for_type_finding = []
660 OAI21b0b1_types = []
661 NMOSlines = []
662 PMOSlines = []

```

```

663 components_list = []
664 inverter_count = 0
665 NAND2_count = 0
666 NOR2_count = 0
667 OR2_count = 0
668 AND2_count = 0
669 NAND2b0_count = 0
670 OAI21_count = 0
671 OAI21b1_count = 0
672 OAI21b0b1_count = 0
673
674 #####Top level functions
675
676 def SCR(file_name):
677     ##This function initializes all global variables and arrays↵
        to zero
678     ##and calls second level functions to conduct SCR
679
680
681     find_tx(file_name)
682     find_cells(inverter_count, NAND2_count, NOR2_count, ↵
        AND2_count, OR2_count, NAND2b0_count, OAI21_count, ↵
        OAI21b1_count, OAI21b0b1_count)
683     find_cell_types()
684     replace_cells(file_name)
685
686
687
688
689 #####Second level functions
690
691 def find_tx(file_name):
692     ##This function reads the input transistor level netlist ↵
        and
693     ##searches for keywords to identify which lines describe ↵
        the
694     ##NMOS and PMOS transistors. Then, it calls third level ↵
        functions
695     ##to create the NMOS and PMOS objects.
696
697     shutil.copyfile(file_name, file_name[:file_name.find(↵
        ".txt")] + '_hierarchical.txt')
```

```

698         #copies input netlist to a new output file. ←
        Netlist changes will be made to output file.
699     NMOS_str = "ami06N" #NMOS keyword
700     PMOS_str = "ami06P" #PMOS keyword
701     width_str = "w="
702     length_str = "l="
703     NMOS_count = 0
704     PMOS_count = 0
705
706     myfile = open(file_name[:file_name.find(".txt")] + '_←
        _hierarchical.txt', 'r+')
707     for num, line in enumerate(myfile):
708
709         if (NMOS_str in line) and (width_str in line) and ←
            (length_str in line):
710             #finds the lines of the transistor level ←
            netlist that describe an NMOS transistor
711             NMOSlines.append(num)
712             NMOS_count = NMOS_count + 1
713
714         if (PMOS_str in line) and (width_str in line) and ←
            (length_str in line):
715             #finds the lines of the transistor level ←
            netlist that describe a PMOS transistor
716             PMOSlines.append(num)
717             PMOS_count = PMOS_count + 1
718
719     create_NMOS_objects(file_name)
720     create_PMOS_objects(file_name)
721
722
723 def find_cells(inverter_count, NAND2_count, NOR2_count, ←
    AND2_count, OR2_count, NAND2b0_count, OAI21_count, ←
    OAI21b1_count, OAI21b0b1_count):
724     ##Calls the functions to identify each type of gate.
725     ##The functions need to be called in order of increasing ←
    abstraction level,
726     ##but decreasing level of complexity within each level of ←
    abstraction.
727
728     find_inverter(inverter_count)
729     find_NAND2(NAND2_count)
730     find_NOR2(NOR2_count)

```

```

731     find_AND2(AND2_count)
732     find_OR2(OR2_count)
733     find_OAI21b0b1(OAI21b0b1_count)
734     find_OAI21b1(OAI21b1_count)
735     find_OAI21(OAI21_count)
736     find_NAND2b0(NAND2b0_count)
737
738 def find_cell_types():
739     ##Calls functions to identify varying types (sizes, widths←
740     , etc.) of each gate.
741     ##The order in which these functions are called does not ←
742     matter.
743
744     find_inverter_types()
745     find_NAND2_types()
746     find_NOR2_types()
747     find_AND2_types()
748     find_OR2_types()
749     find_NAND2b0_types()
750     find_OAI21_types()
751     find_OAI21b1_types()
752     find_OAI21b0b1_types()
753
754 def replace_cells(file_name):
755     ##This function identifies and saves the comments of the ←
756     transistor level netlist,
757     ##identifies the portion of the netlist that lists the ←
758     components,
759     ##maintains the netlist comments in the output netlist,
760     ##and writes the gates to the output netlist.
761
762     NMOS_str = "ami06N"
763     PMOS_str = "ami06P"
764     width_str = "w="
765     length_str = "l="
766     linenumbers = []
767     linenumbers2 = []
768     subcircuit_instance = []
769     before_components = []
770     after_components = []
771     to_delete = []

```

```

769     f = open(file_name[:file_name.find(".txt")] + '↵
       _hierarchical.txt', 'r+')
770     lines = f.readlines()
771
772     ##ID locations of key parts of text
773     for num, line in enumerate(lines):
774
775         if "View name" in line: #This line preceeds the ↵
            beginning of the tx level components list
776             linenum2.append(num)
777
778         if "simulatorOptions" in line: #This line occurs ↵
            after the end of the tx level components list
779             end_components_line = num
780
781     ##Save comments to their own lists
782     f.close()
783     f = open(file_name[:file_name.find(".txt")] + '↵
       _hierarchical.txt', 'r+')
784     lines = f.readlines()
785     for i in range(len(lines)):
786         before_components.append(lines[i]) #saves the ↵
            comments that occur before the components list
787         if "// View name:" in lines[i]: #marks the end of ↵
            the comments
788             ##before_components.append(lines[i])
789             break
790
791     f.close()
792     f = open(file_name[:file_name.find(".txt")] + '↵
       _hierarchical.txt', 'r+')
793     lines = f.readlines()
794     for i in range(len(lines)):
795         if "simulatorOptions" in lines[i]: #marks the ↵
            beginning of the comments
796             number = i
797     for i in range(number, len(lines)):
798         after_components.append(lines[i]) #saves the ↵
            comments that occur after the components list
799     f.close()
800
801     ##Find the components list of the text, save to array
802

```

```

803     i=0
804
805     for j in range(len(lines)-1):
806
807         if i == end_components_line:
808             break
809         if i >= (linenumber2[len(linenumber2)-1] + 1):
810             components_list.append(lines[j])
811
812         i = i + 1
813
814     remove_cell_transistors() #removes the transistors ←
815         that belong to gates from the components list
816     add_cells_to_components() #adds the gates to the ←
817         components list
818     write_cells_to_netlist(file_name, before_components, ←
819         subcircuit_instance, components_list, ←
820         after_components)
821     #creates the subcircuit definitions of each type ←
822         of gate
823
824 #####Third level functions
825
826 def create_NMOS_objects(file_name):
827     ##This function creates NMOS objects from the NMOS lines ←
828     found in the input netlist
829
830     for i in range(len(NMOSlines)):
831         myfile = open(file_name[:file_name.find(".txt")] + ←
832             '_hierarchical.txt', 'r+')
833         n = max(NMOSlines)
834         for j in range(0, n+1):
835             line = myfile.readline()
836             if j == NMOSlines[i]:
837                 NMOS_object = NMOS(line, myfile.readline() ←
838                     ) #There are two lines that contain ←
839                     info for the transistor
840                 NMOS_tx.append(NMOS_object)
841                 NMOS_tx_original.append(NMOS_object)
842             myfile.close()

```

```

837
838 def create_PMOS_objects(file_name):
839     ##This function creates PMOS objects from the NMOS lines ←
        found in the input netlist
840
841     for i in range(len(PMOSlines)):
842         myfile = open(file_name[:file_name.find(".txt")] + ←
            '_hierarchical.txt', 'r+')
843         n = max(PMOSlines)
844         for j in range(0, n+1):
845             line = myfile.readline()
846             if j == PMOSlines[i]:
847                 PMOS_object = PMOS(line, myfile.readline() ←
                    )
848                 PMOSStx.append(PMOS_object)
849                 PMOSStx_original.append(PMOS_object)
850             myfile.close()
851
852
853 #####Find gates#####
854
855 def find_inverter(inverter_count):
856     ##This function identifies patterns of transistors that ←
        match an inverter
857
858     NMOS_index_to_remove = [] #this will keep track of ←
        NMOS transistors to remove from global array
859     PMOS_index_to_remove = [] #this will keep track of ←
        PMOS transistors to remove from global array
860
861     if len(NMOSStx) <= len(PMOSStx): #if uneven # of ←
        transistors, search for NMOS/PMOS pairs using ←
        lowest # of iterations
862         for i in range(0, len(NMOSStx)):
863             for j in range(0, len(PMOSStx)):
864                 if (NMOSStx[i].net()[0] == PMOSStx[j].net() ←
                    [0]) and (NMOSStx[i].net()[1] == PMOSStx[ ←
                    j].net()[1]): #Share a common drain ←
                    pair?
865                     if (NMOSStx[i].net()[2] != PMOSStx[j]. ←
                        net()[2]) and (NMOSStx[i].net()[3] ←
                        != PMOSStx[j].net()[3]): #Share a ←
                        common input?

```

```

866         inverter_count += 1
867         inverter_object = inverter(NMOSStx[←
            i], PMOSStx[j]) #Make inverter ←
            object
868         inverters.append(inverter_object) ←
            #Add inverter object to global ←
            array
869         NMOS_index_to_remove.append(i)
870         PMOS_index_to_remove.append(j)
871         break
872     else:
873         for i in range(0, len(PMOSStx)): #same as previous ←
            if statement
874             for j in range(0, len(NMOSStx)):
875                 if (NMOSStx[j].net()[0] == PMOSStx[i].net()←
                    [0]) and (NMOSStx[j].net()[1] == PMOSStx[←
                    i].net()[1]):
876                     if (NMOSStx[j].net()[2] != PMOSStx[i].←
                        net()[2]) and (NMOSStx[j].net()[3] ←
                        != PMOSStx[i].net()[3]):
877                         inverter_count = inverter_count + ←
                            1
878                         inverter_object = inverter(NMOSStx[←
                            j], PMOSStx[i])
879                         inverters.append(inverter_object)
880                         NMOS_index_to_remove.append(j)
881                         PMOS_index_to_remove.append(i)
882                         break
883
884     remove_transistors(NMOS_index_to_remove, ←
        PMOS_index_to_remove) #removes the transistors from←
        global array
885
886
887 def find_NAND2(NAND2_count):
888     ##This function identifies patterns of transistors/gates ←
        that match a NAND2
889
890     NMOS_index_to_remove = [] #this will keep track of ←
        NMOS transistors to remove from global array
891     PMOS_index_to_remove = [] #this will keep track of ←
        PMOS transistors to remove from global array

```

```

892     inverters_to_remove = [] #this will keep track of ↵
      inverters to remove from global array
893
894     for i in range(len(inverters)): #NAND2 gate is built ↵
      from an inverter structure
895         PMOS_zero = inverters[i].PMOS_object #PMOS ↵
          transistor of inverter in question
896         NMOS_zero = inverters[i].NMOS_object #NMOS ↵
          transistor of inverter in question
897         for j in range(len(NMOSStx)): #comparing NMOS/PMOS ↵
          of inverter with all NMOS/PMOS transistors in ↵
          global array
898             if NMOS_zero.net()[2] == NMOSStx[j].net()[0]: #↵
              NMOS drain connected to NMOS_zero source?
899                 if NMOS_zero.net()[2] != NMOSStx[j].net()↵
                  [2]: #NMOS transistors not in parallel?
900                     for k in range(len(PMOSStx)):
901                         if ((PMOS_zero.net()[0] == PMOSStx[↵
                            k].net()[0]) and (PMOS_zero.net(↵
                            ) [2] == PMOSStx[k].net()[2])) ↵
                            and (PMOS_zero.net()[3] == ↵
                            PMOSStx[k].net()[3]):
902                             #PMOS transistors in parallel?
903                             if (PMOSStx[k].net()[1] == ↵
                                NMOSStx[j].net()[1]): #PMOS/↵
                                NMOS pair share same input?
904                                 NAND2_count = NAND2_count ↵
                                    + 1
905                                 NAND2_object = NAND2(↵
                                    NMOS_zero, NMOSStx[j], ↵
                                    PMOS_zero, PMOSStx[k]) #↵
                                    make NAND2 object
906                                 NAND2s.append(NAND2_object↵
                                    ) #add object to global↵
                                    array
907                                 NMOS_index_to_remove.↵
                                    append(j)
908                                 PMOS_index_to_remove.↵
                                    append(k)
909                                 inverters_to_remove.append↵
                                    (i)
910                                 break
911

```

```

912     inverters_to_remove_sorted = sorted(↵
        inverters_to_remove) #removes inverters that ↵
        compose NAND2 gates from global array
913     for k in range(len(inverters_to_remove_sorted)-1, -1, ↵
        -1):
914         del inverters[inverters_to_remove_sorted[k]]
915
916
917     remove_transistors(NMOS_index_to_remove, ↵
        PMOS_index_to_remove) #removes the transistors from↵
        global array
918
919
920 def find_NOR2(NOR2_count):
921     ##This function identifies patterns of transistors/gates ↵
        that match a NOR2
922
923     NMOS_index_to_remove = [] #this will keep track of ↵
        NMOS transistors to remove from global array
924     PMOS_index_to_remove = [] #this will keep track of ↵
        PMOS transistors to remove from global array
925     inverters_to_remove = [] #this will keep track of ↵
        inverters to remove from global array
926
927     for i in range(len(inverters)): #NAND2 gate is built ↵
        from an inverter structure
928         PMOS_one = inverters[i].PMOS_object #PMOS ↵
            transistor of inverter in question
929         NMOS_one = inverters[i].NMOS_object #NMOS ↵
            transistor of inverter in question
930         for j in range(len(PMOSIx)):
931             if PMOS_one.net()[2] == PMOSIx[j].net()[0]: #↵
                PMOS transistors in series?
932                 if PMOS_one.net()[2] != PMOSIx[j].net()↵
                    [2]: #PMOS transistors not in parallel?
933                     for k in range(len(NMOSIx)):
934                         if ((NMOS_one.net()[0] == NMOSIx[k]↵
                            ].net()[0]) and (NMOS_one.net()↵
                            [2] == NMOSIx[k].net()[2])): # ↵
                            and (NMOS_one.net()[3] == ↵
                            NMOSIx[k].net()[3]):
935                             #NMOS transistors in parallel?

```

```

936         if (NMOSnx[k].net()[1] == ←
PMOSnx[j].net()[1]): #NMOS/←
PMOS pair share same input?
937             NOR2_count = NOR2_count + ←
1
938             NOR2_object = NOR2(NMOSnx[←
k], NMOS_one, PMOSnx[j←
], PMOS_one)
939             NOR2s.append(NOR2_object)
940             PMOS_index_to_remove.←
append(j)
941             NMOS_index_to_remove.←
append(k)
942             inverters_to_remove.append←
(i)
943             break
944
945     inverters_to_remove_sorted = sorted(←
inverters_to_remove) #removes inverters that ←
compose NOR2 gates from global array
946     for k in range(len(inverters_to_remove_sorted)-1, -1, ←
-1):
947         del inverters[inverters_to_remove_sorted[k]]
948
949     remove_transistors(NMOS_index_to_remove, ←
PMOS_index_to_remove)#removes the transistors from ←
global array
950
951 def find_AND2(AND2_count):
952     ##This function identifies patterns of transistors/gates ←
that match an AND2
953
954     NAND2s_to_remove = [] #this will keep track of NAND2s ←
to remove from global array
955     inverters_to_remove = [] #this will keep track of ←
inverters to remove from global array
956     for i in range(len(NAND2s)):
957         NAND2_half = NAND2s[i]
958         for j in range(len(inverters)):
959             inverter_half = inverters[j]
960             if (NAND2_half.NMOS_object_0).net()[0] == (←
inverter_half.NMOS_object).net()[1]: #NAND2←
output is inverter input?

```

```

961         if (NAND2_half.PMOS_object_1).net()[0] == ←
            (inverter_half.NMOS_object).net()[1]: #←
            NAND2 output is inverter input?
962         NMOS_tx_original_copy = NMOS_tx_original
963         NMOS_tx_original_copy.remove(←
            inverter_half.NMOS_object) #This is←
            created so this NMOS object ←
            instance isn't included in search
964         for k in range(len(←
            NMOS_tx_original_copy)):
965             if (NAND2_half.NMOS_object_0).net←
                ([0] == NMOS_tx_original_copy[k←
                ].net()[1]: #makes sure NAND2 ←
                gate is only connected to ←
                inverter
966                 break
967             elif k == len(NMOS_tx_original_copy←
                )-1:
968                 try: #make sure that the ←
                    inverter hasn't already ←
                    been tagged to be removed
969                     inverters_to_remove.remove←
                        (j)
970                     inverters_to_remove.append←
                        (j)
971             except ValueError: #create ←
                AND2 object and tag NAND2 ←
                objects and inverter ←
                objects for removal
972                 AND2_count = AND2_count + ←
                    1
973                 AND2_object = AND2(←
                    NAND2_half, ←
                    inverter_half)
974                 AND2s.append(AND2_object)
975                 NAND2s_to_remove.append(i)
976                 inverters_to_remove.append←
                    (j)
977                 break
978
979 NAND2s_to_remove_sorted = sorted(NAND2s_to_remove) #←
    remove NAND2s that compose AND2s from global array

```

```

980     for k in range(len(NAND2s_to_remove_sorted)-1, -1, -1):
981         x = NAND2s[NAND2s_to_remove_sorted[k]]
982         removed_NAND2s_for_type_finding.append(x)
983         del NAND2s[NAND2s_to_remove_sorted[k]]
984
985     inverters_to_remove_sorted = sorted(
986         inverters_to_remove) #remove inverters that compose
987         AND2 gates from global array
988     for l in range(len(inverters_to_remove_sorted)-1, -1,
989         -1):
990         y = inverters[inverters_to_remove_sorted[l]]
991         removed_inverters_for_type_finding.append(y)
992         del inverters[inverters_to_remove_sorted[l]]
993
994 def find_OR2(OR2_count):
995     ##This function identifies patterns of transistors/gates
996     that match an OR2
997
998     NOR2s_to_remove = [] #this will keep track of NOR2s to
999     remove from global array
1000     inverters_to_remove = [] #this will keep track of
1001     inverters to remove from global array
1002     for i in range(len(NOR2s)):
1003         NOR2_half = NOR2s[i]
1004         for j in range(len(inverters)):
1005             inverter_half = inverters[j]
1006             if (NOR2_half.NMOS_object_1.net()[0] == (
1007                 inverter_half.NMOS_object).net()[1]: #NOR2
1008                 output is inverter input?
1009                 if (NOR2_half.PMOS_object_1.net()[0] == (
1010                     inverter_half.NMOS_object).net()[1]: #
1011                     NOR2 output is inverter input?
1012                     NMOS_tx_original_copy = NMOS_tx_original
1013                     NMOS_tx_original_copy.remove(
1014                         inverter_half.NMOS_object) #This is
1015                         created so this NMOS object
1016                         instance isn't included in search
1017                     for k in range(len(
1018                         NMOS_tx_original_copy)):

```

```

1007         if (NOR2_half.NMOS_object_0).net()↵
1008             [0] == NMOS_tx_original_copy[k].↵
1009             net()[1]:
1010                 break
1011         elif k == len(NMOS_tx_original_copy↵
1012             )-1:
1013             try: #make sure that the ↵
1014                 inverter hasn't already ↵
1015                 been tagged to be removed
1016                 inverters_to_remove.remove↵
1017                     (j)
1018                 inverters_to_remove.append↵
1019                     (j)
1020             except ValueError: #create ↵
1021                 AND2 object and tag NOR2 ↵
1022                 objects and inverter ↵
1023                 objects for removal
1024                 OR2_count = OR2_count + 1
1025                 OR2_object = OR2(NOR2_half↵
1026                     , inverter_half)
1027                 OR2s.append(OR2_object)
1028                 NOR2s_to_remove.append(i)
1029                 inverters_to_remove.append↵
1030                     (j)
1031                 break
1032
1033     NOR2s_to_remove_sorted = sorted(NOR2s_to_remove) #↵
1034         remove NOR2s that compose OR2s from global array
1035     for k in range(len(NOR2s_to_remove_sorted)-1, -1, -1):
1036         x = NOR2s[NOR2s_to_remove_sorted[k]]
1037         removed_NOR2s_for_type_finding.append(x)
1038         del NOR2s[NOR2s_to_remove_sorted[k]]
1039
1040     inverters_to_remove_sorted = sorted(↵
1041         inverters_to_remove) #remove inverters that compose↵
1042         OR2 gates from global array
1043     for l in range(len(inverters_to_remove_sorted)-1, -1, ↵
1044         -1):
1045         y = inverters[inverters_to_remove_sorted[l]]
1046         removed_inverters_for_type_finding.append(y)
1047         del inverters[inverters_to_remove_sorted[l]]

```

```

1034 def find_NAND2b0(NAND2b0_count):
1035     ##This function identifies patterns of transistors/gates ←
        that match a NAND2b0
1036
1037     OR2s_to_remove = [] #this will keep track of OR2s to ←
        remove from global array
1038     inverters_to_remove = [] #this will keep track of ←
        inverters to remove from global array
1039     for i in range(len(OR2s)):
1040         OR2_half = OR2s[i]
1041         for j in range(len(inverters)):
1042             inverter_half = inverters[j]
1043             if (((OR2_half.NOR2_object).NMOS_object_1).net()←
                [1] == (inverter_half.NMOS_object).net()←
                [0]) | (((OR2_half.NOR2_object).←
                NMOS_object_0).net()[1] == (inverter_half.←
                NMOS_object).net()[0]): # inverter output =←
                one of the OR2 inputs?
1044                 #create NAND2b0 object and tag OR2 object ←
                and inverter object for removal
1045                 NAND2b0_count = NAND2b0_count + 1
1046                 NAND2b0_object = NAND2b0(inverter_half, ←
                OR2_half)
1047                 NAND2b0s.append(NAND2b0_object)
1048                 OR2s_to_remove.append(i)
1049                 inverters_to_remove.append(j)
1050                 break
1051
1052     OR2s_to_remove_sorted = sorted(OR2s_to_remove) #remove←
        OR2s that compose NAND2b0s from global array
1053     for k in range(len(OR2s_to_remove_sorted)-1, -1, -1):
1054         x = OR2s[OR2s_to_remove_sorted[k]]
1055         removed_OR2s_for_type_finding.append(x)
1056         del OR2s[OR2s_to_remove_sorted[k]]
1057
1058     inverters_to_remove_sorted = sorted(←
        inverters_to_remove) #remove inverters that compose←
        NAND2b0s from global array
1059     for l in range(len(inverters_to_remove_sorted)-1, -1, ←
        -1):
1060         y = inverters[inverters_to_remove_sorted[l]]
1061         removed_inverters_for_type_finding.append(y)
1062         del inverters[inverters_to_remove_sorted[l]]

```

```

1063
1064 def find_OAI21(OAI21_count):
1065     ##This function identifies patterns of transistors/gates ←
        that match an OAI21
1066     ##Note: the AND2 operations will only be used if AND2 ←
        gates are found before NAND2 gates.
1067
1068     OR2s_to_remove = [] #this will keep track of OR2s to ←
        remove from global array
1069     NAND2s_to_remove = [] #this will keep track of NAND2s ←
        to remove from global array
1070     AND2s_to_remove = [] #this will keep track of AND2s to ←
        remove from global array
1071     for i in range(len(OR2s)):
1072         OR2_half = OR2s[i]
1073         for j in range(len(NAND2s)):
1074             NAND2_half = NAND2s[j]
1075             if (((OR2_half.inverter_object).NMOS_object).←
                net()[0] == (NAND2_half.NMOS_object_1).net←
                ()[1]) | (((OR2_half.inverter_object).←
                NMOS_object).net()[0] == (NAND2_half.←
                NMOS_object_0).net()[1]): #output of OR2 ←
                gate = input of NAND2 gate?
1076                 try: #Make sure NAND2 gate hasn't already ←
                    been tagged for removal
1077                     NAND2s_to_remove.remove(j)
1078                     NAND2s_to_remove.append(j)
1079                 except ValueError: #Create OAI21 object ←
                    and tag OR2 and NAND2 gates for removal
1080                     OAI21_count = OAI21_count + 1
1081                     OAI21_object = OAI21(OR2_half, ←
                        NAND2_half)
1082                     OAI21s.append(OAI21_object)
1083                     OR2s_to_remove.append(i)
1084                     NAND2s_to_remove.append(j)
1085                     break
1086
1087     AND2s_to_remove_sorted = sorted(AND2s_to_remove) #←
        remove AND2s that compose OAI21s from global array
1088     for r in range(len(AND2s_to_remove_sorted)-1, -1, -1):
1089         del AND2s[AND2s_to_remove_sorted[r]]
1090

```

```

1091 OR2s_to_remove_sorted = sorted(OR2s_to_remove) #remove↵
      OR2s that compose OAI212s from global array
1092 for k in range(len(OR2s_to_remove_sorted)-1, -1, -1):
1093     x = OR2s[OR2s_to_remove_sorted[k]]
1094     removed_OR2s_for_type_finding.append(x)
1095     del OR2s[OR2s_to_remove_sorted[k]]
1096
1097 NAND2s_to_remove_sorted = sorted(NAND2s_to_remove) #↵
      remove NAND2s that compose OAI212s from global ↵
      array
1098 for l in range(len(NAND2s_to_remove_sorted)-1, -1, -1)↵
      :
1099     y = NAND2s[NAND2s_to_remove_sorted[l]]
1100     removed_NAND2s_for_type_finding.append(y)
1101     del NAND2s[NAND2s_to_remove_sorted[l]]
1102
1103
1104
1105 def find_OAI21b1(OAI21b1_count):
1106
1107     NAND2b0s_to_remove = [] #this will keep track of ↵
      NAND2b0s to remove from global array
1108     NAND2s_to_remove = [] #this will keep track of NAND2s ↵
      to remove from global array
1109     inverters_to_remove = [] #this will keep track of ↵
      inverters to remove from global array
1110     OAI21s_to_remove = [] #this will keep track of OAI21s ↵
      to remove from global array
1111     OR2s_to_remove = [] #this will keep track of OR2s to ↵
      remove from global array
1112
1113     for a in range(len(OAI21s)): #Looks for inverter ↵
      output = OR2 input, OR2 output = NAND2 input
1114         for b in range (len(inverters)):
1115             if (inverters[b].NMOS_net()[0] == (((OAI21s[a↵
      ].OR2_object).NOR2_object).NMOS_object_0).↵
      net()[1]) | (inverters[b].NMOS_net()[0] == ↵
      (((OAI21s[a].OR2_object).NOR2_object).↵
      NMOS_object_1).net()[1]):
1116                 try:
1117                     inverters_to_remove.remove(b)
1118                     inverters_to_remove.append(b)
1119                 except ValueError:

```

```

1120         OAI21b1_count = OAI21b1_count + 1
1121         OAI21b1_object = OAI21b1(inverters[b], ←
            OAI21s[a].OR2_object, OAI21s[a]. ←
            NAND2_object)
1122         OAI21b1s.append(OAI21b1_object)
1123         OAI21s_to_remove.append(a)
1124         inverters_to_remove.append(b)
1125         break
1126
1127     for c in range(len(inverters)): #Looks for inverter ←
        output = OR2 input of OAI21 gate
1128         inverter_part = inverters[c]
1129         for d in range(len(OR2s)):
1130             OR2_part = OR2s[d]
1131             if ((inverter_part.NMOS_object).net()[0] == ((←
                OR2_part.NOR2_object).NMOS_object_0).net() ←
                [1]) | ((inverter_part.NMOS_object).net() ←
                [0] == ((OR2_part.NOR2_object). ←
                NMOS_object_1).net()[1]):
1132                 for e in range(len(NAND2s)):
1133                     NAND2_part = NAND2s[e]
1134                     if (((OR2_part.inverter_object). ←
                        NMOS_object).net()[0] == (←
                        NAND2_part.NMOS_object_0).net()[1]) ←
                        | (((OR2_part.inverter_object). ←
                        NMOS_object).net()[0] == (←
                        NAND2_part.NMOS_object_1).net()[1]) ←
                        :
1135                         try:
1136                             NAND2s_to_remove.remove(e)
1137                             NAND2s_to_remove.append(e)
1138             except ValueError:
1139                 OAI21b1_count = OAI21b1_count ←
                    + 1
1140                 OAI21b1_object = OAI21b1(←
                    inverter_part, OR2_part, ←
                    NAND2_part)
1141                 OAI21b1s.append(OAI21b1_object ←
                    )
1142                 inverters_to_remove.append(c)
1143                 OR2s_to_remove.append(d)
1144                 NAND2s_to_remove.append(e)
1145                 break

```

```

1146         break
1147
1148     for f in range(len(NAND2s)): #Looks for NAND2b0 output↵
        = NAND2 input
1149         for g in range(len(NAND2b0s)):
1150             if (((NAND2b0s[g].OR2_object).inverter_object↵
                ).NMOS_net()[0]) == (NAND2s[f].↵
                NMOS_object_0).net()[1]) | (((NAND2b0s[g].↵
                OR2_object).inverter_object).NMOS_net()[0])↵
                == (NAND2s[f].NMOS_object_1).net()[1]):
1151                 try:
1152                     NAND2b0s_to_remove.remove(g)
1153                     NAND2b0s_to_remove.append(g)
1154             except ValueError:
1155                 OAI21b1_count = OAI21b1_count + 1
1156                 OAI21b1_object = OAI21b1(NAND2b0s[g].↵
                    inverter_object, NAND2b0s[g].↵
                    OR2_object, NAND2s[f])
1157                 OAI21b1s.append(OAI21b1_object)
1158                 NAND2b0s_to_remove.append(g)
1159                 NAND2s_to_remove.append(f)
1160                 break
1161
1162     OAI21s_to_remove_sorted = sorted(OAI21s_to_remove)
1163     for h in range(len(OAI21s_to_remove_sorted)-1, -1, -1)↵
        :
1164         i = OAI21s[OAI21s_to_remove_sorted[h]]
1165         removed_OAI21s_for_type_finding.append(i)
1166         del OAI21s[OAI21s_to_remove_sorted[h]]
1167
1168     NAND2b0s_to_remove_sorted = sorted(NAND2b0s_to_remove)
1169     for j in range(len(NAND2b0s_to_remove_sorted)-1, -1, ↵
        -1):
1170         k = NAND2b0s[NAND2b0s_to_remove_sorted[j]]
1171         removed_NAND2b0s_for_type_finding.append(k)
1172         del NAND2b0s[NAND2b0s_to_remove_sorted[j]]
1173
1174     NAND2s_to_remove_sorted = sorted(NAND2s_to_remove)
1175     for l in range(len(NAND2s_to_remove_sorted)-1, -1, -1)↵
        :
1176         m = NAND2s[NAND2s_to_remove_sorted[l]]
1177         removed_NAND2s_for_type_finding.append(m)
1178         del NAND2s[NAND2s_to_remove_sorted[l]]

```

```

1179
1180     inverters_to_remove_sorted = sorted(↵
1181         inverters_to_remove)
1182     for n in range(len(inverters_to_remove_sorted)-1, -1, ↵
1183         -1):
1184         o = inverters[inverters_to_remove_sorted[n]]
1185         removed_inverters_for_type_finding.append(o)
1186         del inverters[inverters_to_remove_sorted[n]]
1187
1188     OR2s_to_remove_sorted = sorted(OR2s_to_remove)
1189     for p in range(len(OR2s_to_remove_sorted)-1, -1, -1):
1190         q = OR2s[OR2s_to_remove[p]]
1191         removed_OR2s_for_type_finding.append(q)
1192         del OR2s[OR2s_to_remove[p]]
1193
1194 def find_OAI21b0b1(OAI21b0b1_count):
1195     inverters_to_remove = []
1196     OR2s_to_remove = []
1197     AND2s_to_remove = []
1198     NAND2b0s_to_remove = []
1199
1200     for h in range(len(inverters)): # looks for inverter ↵
1201         output = OR2 input, AND2 output = other OR2 input
1202         inverter_part = inverters[h]
1203         for i in range(len(OR2s)):
1204             OR2_part = OR2s[i]
1205             if ((inverter_part.NMOS_object).net()[0] == ((↵
1206                 OR2_part.NOR2_object).NMOS_object_0).net()↵
1207                 [1]) | ((inverter_part.NMOS_object).net()↵
1208                 [0] == ((OR2_part.NOR2_object).↵
1209                 NMOS_object_1).net()[1])):
1210                 for j in range(len(AND2s)):
1211                     AND2_part = AND2s[j]
1212                     if (((AND2_part.inverter_object).↵
1213                         NMOS_object).net()[0] == ((OR2_part↵
1214                         .NOR2_object).NMOS_object_1).net()↵
1215                         [1]) | (((AND2_part.inverter_object↵
1216                         ).NMOS_object).net()[0] == ((↵
1217                         OR2_part.NOR2_object).NMOS_object_0↵
1218                         ).net()[1])):
1219                         try:

```

```

1209         AND2s_to_remove.remove(j)
1210         AND2s_to_remove.append(j)
1211     except ValueError:
1212         OAI21b0b1_count = ←
1213             OAI21b0b1_count + 1
1214         OAI21b0b1_object = OAI21b0b1(←
1215             inverter_part, AND2_part, ←
1216             OR2_part)
1217         OAI21b0b1s.append(←
1218             OAI21b0b1_object)
1219         inverters_to_remove.append(h)
1220         OR2s_to_remove.append(i)
1221         AND2s_to_remove.append(j)
1222         break
1223
1224 for a in range(len(NAND2b0s)): # looks for output AND2←
1225     gate = OR2 input of NAND2b0 gate
1226     for b in range(len(AND2s)):
1227         if (((NAND2b0s[a].OR2_object).NOR2_object).←
1228             NMOS_object_0).net()[1] == ((AND2s[b].←
1229             inverter_object).NMOS_object).net()[0]) | ←
1230             (((NAND2b0s[a].OR2_object).NOR2_object).←
1231             NMOS_object_1).net()[1] == ((AND2s[b].←
1232             inverter_object).NMOS_object).net()[0]):
1233             try:
1234                 AND2s_to_remove.remove(b)
1235                 AND2s_to_remove.append(b)
1236             except ValueError:
1237                 OAI21b0b1_count = OAI21b0b1_count + 1
1238                 OAI21b0b1_object = OAI21b0b1(NAND2b0s[←
1239                     a].inverter_object, AND2s[b], ←
1240                     NAND2b0s[a].OR2_object)
1241                 OAI21b0b1s.append(OAI21b0b1_object)
1242                 NAND2b0s_to_remove.append(a)
1243                 AND2s_to_remove.append(b)
1244                 break
1245
1246 for c in range(len(OR2s)): #separates an OR2 gate in ←
1247     to a NOR2 gate and inverter, then looks for ←
1248     inverter output = OR2 input, AND2 output = other ←
1249     OR2 input
1250     for d in range(len(AND2s)):
1251         for e in range(len(OR2s)):

```

```

1237         if c == e:
1238             break
1239         elif (((OR2s[c].inverter_object).↵
            NMOS_object).net()[0] == ((OR2s[e].↵
            NOR2_object).NMOS_object_0).net()[1]) |↵
            (((OR2s[c].inverter_object).↵
            NMOS_object).net()[0] == ((OR2s[e].↵
            NOR2_object).NMOS_object_1).net()[1]):
1240             if (((AND2s[d].inverter_object).↵
                NMOS_object).net()[0] == ((OR2s[e].↵
                NOR2_object).NMOS_object_0).net()↵
                [1]) | (((AND2s[d].inverter_object)↵
                .NMOS_object).net()[0] == ((OR2s[e]↵
                ].NOR2_object).NMOS_object_1).net()↵
                [1])):
1241                 try:
1242                     OR2s_to_remove.remove(e)
1243                     OR2s_to_remove.append(e)
1244             except ValueError:
1245                 OAI21b0b1_count = ↵
                    OAI21b0b1_count + 1
1246                 OAI21b0b1_object = OAI21b0b1(↵
                    OR2s[c].inverter_object, ↵
                    AND2s[d], OR2s[e])
1247                 OAI21b0b1s.append(↵
                    OAI21b0b1_object)
1248                 OR2s_to_remove.append(c)
1249                 OR2s_to_remove.append(e)
1250                 AND2s_to_remove.append(d)
1251                 NOR2s.append(OR2s[c].↵
                    NOR2_object)
1252                 break
1253
1254     for f in range(len(AND2s)): #separates an AND2 gate in↵
        to a NAND2 gate and inverter, then looks for ↵
        inverter output = OR2 input, AND2 output = other ↵
        OR2 input
1255         for g in range(len(OR2s)):
1256             if (((AND2s[f].inverter_object).NMOS_object).↵
                net()[0] == ((OR2s[g].NOR2_object).↵
                NMOS_object_0).net()[1]) | (((AND2s[f].↵
                inverter_object).NMOS_object).net()[0] == ↵

```

```

1257 ((OR2s[g].NOR2_object).NMOS_object_1).net()↵
1258 [1]):
1259     for h in range(len(AND2s)):
1260         if f == h:
1261             break
1262         elif (((AND2s[h].inverter_object).↵
1263             NMOS_object).net()[0] == ((OR2s[g].↵
1264             NOR2_object).NMOS_object_0).net()↵
1265             [1]) | (((AND2s[h].inverter_object)↵
1266             .NMOS_object).net()[0] == ((OR2s[g]↵
1267             ].NOR2_object).NMOS_object_1).net()↵
1268             [1]):
1269             try:
1270                 AND2s_to_remove.remove(h)
1271                 AND2s_to_remove.append(h)
1272             except ValueError:
1273                 flag = False
1274                 for i in range(len(OR2s)):
1275                     if ((AND2s[f].NAND2_object↵
1276                     ).NMOS_object_0).net()↵
1277                     [1] == ((OR2s[i].↵
1278                     inverter_object).↵
1279                     NMOS_object).net()[0]:
1280                         OAI21b0b1_count = ↵
1281                         OAI21b0b1_count + 1
1282                         OAI21b0b1_object = ↵
1283                         OAI21b0b1(AND2s[f].↵
1284                         inverter_object, ↵
1285                         AND2s[h], OR2s[g])
1286                         OAI21b0b1s.append(↵
1287                         OAI21b0b1_object)
1288                         AND2s_to_remove.append(↵
1289                         (f)
1290                         OR2s_to_remove.append(↵
1291                         g)
1292                         AND2s_to_remove.append(↵
1293                         (h)
1294                         NAND2s.append(AND2s[f↵
1295                         ].NAND2_object)
1296                         flag = True
1297                         break
1298                     elif ((AND2s[h].↵
1299                     NAND2_object).↵

```

```

1278 NMOS_object_0).net()[1]↵
1279 == ((OR2s[i].↵
inverter_object).↵
NMOS_object).net()[0]:
OAI21b0b1_count = ↵
OAI21b0b1_count + 1
OAI21b0b1_object = ↵
OAI21b0b1(AND2s[h].↵
inverter_object, ↵
AND2s[f], OR2s[g])
OAI21b0b1s.append(↵
OAI21b0b1_object)
AND2s_to_remove.append(↵
(f)
OR2s_to_remove.append(↵
g)
AND2s_to_remove.append(↵
(h)
NAND2s.append(AND2s[h↵
].NAND2_object)
flag = True
break
1287 if flag == False:
1288     OAI21b0b1_count = ↵
OAI21b0b1_count + 1
OAI21b0b1_object = ↵
OAI21b0b1(AND2s[f].↵
inverter_object, AND2s[↵
h], OR2s[g])
1290 OAI21b0b1s.append(↵
OAI21b0b1_object)
AND2s_to_remove.append(f)
1292 OR2s_to_remove.append(g)
AND2s_to_remove.append(h)
1294 NAND2s.append(AND2s[f].↵
NAND2_object)
break
1296 else:
1297     break
1298
1299
1300
1301 NAND2b0s_to_remove_sorted = sorted(NAND2b0s_to_remove)

```

```

1302     for c in range(len(NAND2b0s_to_remove_sorted)-1, -1, ←
1303         -1):
1304         d = NAND2b0s[NAND2b0s_to_remove_sorted[c]]
1305         removed_NAND2b0s_for_type_finding.append(d)
1306         del NAND2b0s[NAND2b0s_to_remove_sorted[c]]
1307
1308     inverters_to_remove_sorted = sorted(←
1309         inverters_to_remove)
1310     for m in range(len(inverters_to_remove_sorted)-1, -1, ←
1311         -1):
1312         z = inverters[inverters_to_remove_sorted[m]]
1313         removed_inverters_for_type_finding.append(z)
1314         del inverters[inverters_to_remove_sorted[m]]
1315
1316     OR2s_to_remove_sorted = sorted(OR2s_to_remove)
1317     for k in range(len(OR2s_to_remove_sorted)-1, -1, -1):
1318         x = OR2s[OR2s_to_remove_sorted[k]]
1319         removed_OR2s_for_type_finding.append(x)
1320         del OR2s[OR2s_to_remove_sorted[k]]
1321
1322     AND2s_to_remove_sorted = sorted(AND2s_to_remove)
1323     for l in range(len(AND2s_to_remove_sorted)-1, -1, -1):
1324         y = AND2s[AND2s_to_remove_sorted[l]]
1325         removed_AND2s_for_type_finding.append(y)
1326         del AND2s[AND2s_to_remove_sorted[l]]
1327
1328     #####Find gate types#####
1329
1330     def find_inverter_types():
1331         if len(inverters) != 0: #only proceed of this type of ←
1332             gate exists in circuit
1333             new_inverter_type = inverter_type(inverters[0]) #←
1334             make first gate in global array the first gate ←
1335             type
1336             inverter_types.append(new_inverter_type)
1337             type_number = 1
1338
1339             for i in range(1, len(inverters)): #compare the ←
1340                 other gates in global array
1341                 for j in range(0, len(inverter_types)):

```

```

1338         if inverters[i].widths() == (←
            inverter_types[j].inverter_object()).←
            widths():
1339             if inverters[i].lengths() == (←
                inverter_types[j].inverter_object()←
                ).lengths():
1340                 break #if same type as already in ←
                    array, move on to next inverter
1341         else:
1342             if j == len(inverter_types)-1: #if not←
                same type as already in array, ←
                create new type
1343                 new_inverter_type = inverter_type(←
                    inverters[i])
1344                 inverter_types.append(←
                    new_inverter_type)
1345                 inverters[i].inverter_type = ←
                    type_number
1346                 inverter_types[j+1].inverter_type ←
                    = type_number
1347                 type_number = type_number + 1
1348
1349     if len(removed_inverters_for_type_finding) != 0: #←
        repeat process for subgates that were deleted. ←
        Needed for subcircuit definitions
1350         if len(inverter_types) == 0:
1351             new_inverter_type = inverter_type(←
                removed_inverters_for_type_finding[0])
1352             inverter_types.append(new_inverter_type)
1353             type_number = 1
1354         for i in range(1, len(←
            removed_inverters_for_type_finding)):
1355             for j in range(0, len(inverter_types)):
1356                 if removed_inverters_for_type_finding[i].←
                    widths() == (inverter_types[j].←
                    inverter_object()).widths():
1357                     if removed_inverters_for_type_finding[←
                        i].lengths() == (inverter_types[j].←
                        inverter_object()).lengths():
1358                         break
1359                 else:
1360                     if j == len(inverter_types)-1:

```

```

1361         new_inverter_type = inverter_type(↵
1362             removed_inverters_for_type_finding↵
1363             [i])
1364         inverter_types.append(↵
1365             new_inverter_type)
1366         removed_inverters_for_type_finding↵
1367         [i].inverter_type = type_number
1368         inverter_types[j+1].inverter_type ↵
1369         = type_number
1370         type_number = type_number + 1
1371
1372 def find_NAND2_types():
1373     if len(NAND2s) != 0: #only proceed of this type of ↵
1374         gate exists in circuit
1375
1376         new_NAND2_type = NAND2_type(NAND2s[0]) #make first↵
1377         gate in global array the first gate type
1378         NAND2_types.append(new_NAND2_type)
1379         type_number = 1
1380
1381         for i in range(1, len(NAND2s)): #compare the other↵
1382         gates in global array
1383         for j in range(0, len(NAND2_types)):
1384             if NAND2s[i].widths() == (NAND2_types[j].↵
1385                 NAND2_object()).widths():
1386                 if NAND2s[i].lengths() == (NAND2_types↵
1387                     [j].NAND2_object()).lengths():
1388                     break
1389             else:
1390                 if j == len(NAND2_types)-1: #if not ↵
1391                 same type as already in array, ↵
1392                 create new type
1393                 new_NAND2_type = NAND2_type(NAND2s↵
1394                     [i])
1395                 NAND2_types.append(new_NAND2_type)
1396                 NAND2s[i].NAND2_type = type_number
1397                 NAND2_types[j+1].NAND2_type = ↵
1398                 type_number
1399                 type_number = type_number + 1

```

```

1390
1391     if len(removed_NAND2s_for_type_finding) != 0: #repeat ↵
1392         process for subgates that were deleted. Needed for ↵
1393         subcircuit definitions
1394         if len(NAND2_types) == 0:
1395             new_NAND2_type = NAND2_type(↵
1396                 removed_NAND2s_for_type_finding[0])
1397             NAND2_types.append(new_NAND2_type)
1398             type_number = 1
1399         for i in range(1, len(↵
1400             removed_NAND2s_for_type_finding)):
1401             for j in range(0, len(NAND2_types)):
1402                 if removed_NAND2s_for_type_finding[i].↵
1403                     widths() == (NAND2_types[j].↵
1404                         NAND2_object()).widths():
1405                     if removed_NAND2s_for_type_finding[i].↵
1406                         lengths() == (NAND2_types[j].↵
1407                             NAND2_object()).lengths():
1408                         break
1409             else:
1410                 if j == len(NAND2_types)-1:
1411                     new_NAND2_type = NAND2_type(↵
1412                         removed_NAND2s_for_type_finding↵
1413                             [i])
1414                     NAND2_types.append(new_NAND2_type)
1415                     removed_NAND2s_for_type_finding[i]↵
1416                         .NAND2_type = type_number
1417                     NAND2_types[j+1].NAND2_type = ↵
1418                         type_number
1419                     type_number = type_number + 1
1420
1421 def find_NOR2_types():
1422
1423     if len(NOR2s) != 0: #only proceed of this type of gate↵
1424         exists in circuit
1425
1426     new_NOR2_type = NOR2_type(NOR2s[0]) #make first ↵
1427     gate in global array the first gate type
1428     NOR2_types.append(new_NOR2_type)
1429     type_number = 1
1430
1431     for i in range(1, len(NOR2s)): #compare the other↵
1432         gates in global array

```

```

1418         for j in range(0, len(NOR2_types)):
1419             if NOR2s[i].widths() == (NOR2_types[j].NOR2_object().widths():
1420                 if NOR2s[i].lengths() == (NOR2_types[j].NOR2_object().lengths():
1421                     break
1422             else:
1423                 if j == len(NOR2_types)-1: #if not
1424                     same type as already in array,
1425                     create new type
1426                     new_NOR2_type = NOR2_type(NOR2s[i].NOR2_object())
1427                     NOR2_types.append(new_NOR2_type)
1428                     NOR2s[i].NOR2_type = type_number
1429                     NOR2_types[j+1].NOR2_type = type_number
1430                     type_number = type_number + 1
1431
1432     if len(removed_NOR2s_for_type_finding) != 0: #repeat
1433         process for subgates that were deleted. Needed for
1434         subcircuit definitions
1435         if len(NOR2_types) == 0:
1436             new_NOR2_type = NOR2_type(removed_NOR2s_for_type_finding[0])
1437             NOR2_types.append(new_NOR2_type)
1438             type_number = 1
1439         for i in range(1, len(removed_NOR2s_for_type_finding)):
1440             for j in range(0, len(NOR2_types)):
1441                 if removed_NOR2s_for_type_finding[i].widths() == (NOR2_types[j].NOR2_object().widths():
1442                     if removed_NOR2s_for_type_finding[i].lengths() == (NOR2_types[j].NOR2_object().lengths():
1443                         break
1444             else:
1445                 if j == len(NOR2_types)-1:
1446                     new_NOR2_type = NOR2_type(removed_NOR2s_for_type_finding[i])
1447                     NOR2_types.append(new_NOR2_type)

```

```

1444         removed_NOR2s_for_type_finding[i].NOR2_type = type_number
1445         NOR2_types[j+1].NOR2_type = type_number
1446         type_number = type_number + 1
1447
1448 def find_AND2_types():
1449
1450     if len(AND2s) != 0: #only proceed of this type of gate exists in circuit
1451
1452         new_AND2_type = AND2_type(AND2s[0]) #make first gate in global array the first gate type
1453         AND2_types.append(new_AND2_type)
1454         type_number = 1
1455
1456         for i in range(1, len(AND2s)): #compare the other gates in global array
1457             for j in range(0, len(AND2_types)):
1458                 if AND2s[i].widths() == (AND2_types[j].AND2_object()).widths():
1459                     if AND2s[i].lengths() == (AND2_types[j].AND2_object()).lengths():
1460                         break
1461                 else:
1462                     if j == len(AND2_types)-1: #if not same type as already in array, create new type
1463                         new_AND2_type = AND2_type(AND2s[i])
1464                         AND2_types.append(new_AND2_type)
1465                         AND2s[i].AND2_type = type_number
1466                         AND2_types[j+1].AND2_type = type_number
1467                         type_number = type_number + 1
1468
1469     if len(removed_AND2s_for_type_finding) != 0: #repeat process for subgates that were deleted. Needed for subcircuit definitions
1470         if len(AND2_types) == 0:
1471             new_AND2_type = AND2_type(removed_AND2s_for_type_finding[0])
1472             AND2_types.append(new_AND2_type)

```

```

1473         type_number = 1
1474     for i in range(1, len(removed_AND2s_for_type_finding)):
1475         for j in range(0, len(AND2_types)):
1476             if removed_AND2s_for_type_finding[i].widths() == (AND2_types[j].AND2_object()).widths():
1477                 if removed_AND2s_for_type_finding[i].lengths() == (AND2_types[j].AND2_object()).lengths():
1478                     break
1479             else:
1480                 if j == len(AND2_types)-1:
1481                     new_AND2_type = AND2_type(removed_AND2s_for_type_finding[i])
1482                     AND2_types.append(new_AND2_type)
1483                     removed_AND2s_for_type_finding[i].AND2_type = type_number
1484                     AND2_types[j+1].AND2_type = type_number
1485                     type_number = type_number + 1
1486
1487 def find_OR2_types():
1488
1489     if len(OR2s) != 0: #only proceed of this type of gate exists in circuit
1490         new_OR2_type = OR2_type(OR2s[0]) #make first gate in global array the first gate type
1491         OR2_types.append(new_OR2_type)
1492         type_number = 1
1493
1494         for i in range(1, len(OR2s)): #compare the other gates in global array
1495             for j in range(0, len(OR2_types)):
1496                 if OR2s[i].widths() == (OR2_types[j].OR2_object()).widths():
1497                     if OR2s[i].lengths() == (OR2_types[j].OR2_object()).lengths():
1498                         break
1499             else:

```

```

1500         if j == len(OR2_types)-1: #if not ←
1501             same type as already in array, ←
1502             create new type
1503             new_OR2_type = OR2_type(OR2s[i])
1504             OR2_types.append(new_OR2_type)
1505             OR2s[i].OR2_type = type_number
1506             OR2_types[j+1].OR2_type = ←
1507                 type_number
1508                 type_number = type_number + 1
1509
1510     if len(removed_OR2s_for_type_finding) != 0: #repeat ←
1511         process for subgates that were deleted. Needed for ←
1512         subcircuit definitions
1513         if len(OR2_types) == 0:
1514             new_OR2_type = OR2_type(←
1515                 removed_OR2s_for_type_finding[0])
1516             OR2_types.append(new_OR2_type)
1517             type_number = 1
1518
1519         for i in range(0, len(←
1520             removed_OR2s_for_type_finding)):
1521             for j in range(0, len(OR2_types)):
1522                 if removed_OR2s_for_type_finding[i].widths(←
1523                     ) == (OR2_types[j].OR2_object()).←
1524                     widths():
1525                     if removed_OR2s_for_type_finding[i].←
1526                         lengths() == (OR2_types[j].←
1527                             OR2_object()).lengths():
1528                         break
1529                 else:
1530                     if j == len(OR2_types)-1:
1531                         new_OR2_type = OR2_type(←
1532                             removed_OR2s_for_type_finding[i]←
1533                                 ])
1534                         OR2_types.append(new_OR2_type)
1535                         removed_OR2s_for_type_finding[i].←
1536                             OR2_type = type_number
1537                         OR2_types[j+1].OR2_type = ←
1538                             type_number
1539                         type_number = type_number + 1
1540
1541 def find_NAND2b0_types():
1542

```

```

1528     if len(NAND2b0s) != 0: #only proceed of this type of ↵
1529         gate exists in circuit
1530
1531         new_NAND2b0_type = NAND2b0_type(NAND2b0s[0]) #↵
1532         make first gate in global array the first gate ↵
1533         type
1534     NAND2b0_types.append(new_NAND2b0_type)
1535     type_number = 1
1536
1537     for i in range(1, len(NAND2b0s)): #compare the ↵
1538         other gates in global array
1539         for j in range(0, len(NAND2b0_types)):
1540             if NAND2b0s[i].widths() == (NAND2b0_types[↵
1541                 j].NAND2b0_object()).widths():
1542                 if NAND2b0s[i].lengths() == (↵
1543                     NAND2b0_types[j].NAND2b0_object()).↵
1544                     lengths():
1545                     break
1546             else:
1547                 if j == len(NAND2b0_types)-1: #if not↵
1548                     same type as already in array, ↵
1549                     create new type
1550                     new_NAND2b0_type = NAND2b0_type(↵
1551                         NAND2b0s[i])
1552                     NAND2b0_types.append(↵
1553                         new_NAND2b0_type)
1554                     NAND2b0s[i].NAND2b0_type = ↵
1555                     type_number
1556                     NAND2b0_types[j+1].NAND2b0_type = ↵
1557                     type_number
1558                     type_number = type_number + 1
1559
1560 if len(removed_NAND2b0s_for_type_finding) != 0: #↵
1561     repeat process for subgates that were deleted. ↵
1562     Needed for subcircuit definitions
1563     if len(NAND2b0_types) == 0:
1564         new_NAND2b0_type = NAND2b0_type(↵
1565             removed_NAND2b0s_for_type_finding[0])
1566         NAND2b0_types.append(new_NAND2b0_type)
1567         type_number = 1
1568     for i in range(1, len(↵
1569         removed_NAND2b0s_for_type_finding)):
1570         for j in range(0, len(NAND2b0_types)):

```

```

1554         if removed_NAND2b0s_for_type_finding[i].widths() == (NAND2b0_types[j].NAND2b0_object()).widths():
1555             if removed_NAND2b0s_for_type_finding[i].lengths() == (NAND2b0_types[j].NAND2b0_object()).lengths():
1556                 break
1557         else:
1558             if j == len(NAND2b0_types)-1:
1559                 new_NAND2b0_type = NAND2b0_type(removed_NAND2b0s_for_type_finding[i])
1560                 NAND2b0_types.append(new_NAND2b0_type)
1561                 removed_NAND2b0s_for_type_finding[i].NAND2b0_type = type_number
1562                 NAND2b0_types[j+1].NAND2b0_type = type_number
1563                 type_number = type_number + 1
1564
1565     def find_OAI21_types():
1566
1567         if len(OAI21s) != 0: #only proceed of this type of gate exists in circuit
1568
1569             new_OAI21_type = OAI21_type(OAI21s[0]) #make first gate in global array the first gate type
1570             OAI21_types.append(new_OAI21_type)
1571             type_number = 1
1572
1573             for i in range(1, len(OAI21s)): #compare the other gates in global array
1574                 for j in range(0, len(OAI21_types)):
1575                     if OAI21s[i].widths() == (OAI21_types[j].OAI21_object()).widths():
1576                         if OAI21s[i].lengths() == (OAI21_types[j].OAI21_object()).lengths():
1577                             break
1578                 else:
1579                     if j == len(OAI21_types)-1: #if not same type as already in array, create new type

```

```

1580         new_OAI21_type = OAI21_type(OAI21s←
1581         [i])
1582         OAI21_types.append(new_OAI21_type)
1583         OAI21s[i].OAI21_type = type_number
1584         OAI21_types[j+1].OAI21_type = ←
1585         type_number
1586         type_number = type_number + 1
1587
1588     if len(removed_OAI21s_for_type_finding) != 0: #repeat ←
1589     process for subgates that were deleted. Needed for ←
1590     subcircuit definitions
1591     if len(OAI21_types) == 0:
1592         new_OAI21_type = OAI21_type(←
1593         removed_OAI21s_for_type_finding[0])
1594         OAI21_types.append(new_OAI21_type)
1595         type_number = 1
1596     for i in range(1, len(←
1597     removed_OAI21s_for_type_finding)):
1598         for j in range(0, len(OAI21_types)):
1599             if removed_OAI21s_for_type_finding[i].←
1600             widths() == (OAI21_types[j].←
1601             OAI21_object()).widths():
1602                 if removed_OAI21s_for_type_finding[i].←
1603                 lengths() == (OAI21_types[j].←
1604                 OAI21_object()).lengths():
1605                     break
1606             else:
1607                 if j == len(OAI21_types)-1:
1608                     new_OAI21_type = OAI21_type(←
1609                     removed_OAI21s_for_type_finding←
1610                     [i])
1611                     OAI21_types.append(new_OAI21_type)
1612                     removed_OAI21s_for_type_finding[i]←
1613                     ].OAI21_type = type_number
1614                     OAI21_types[j+1].OAI21_type = ←
1615                     type_number
1616                     type_number = type_number + 1
1617
1618 def find_OAI21b1_types():
1619
1620     if len(OAI21b1s) != 0: #only proceed of this type of ←
1621     gate exists in circuit

```

```

1608     new_OAI21b1_type = OAI21b1_type(OAI21b1s[0]) #↵
        make first gate in global array the first gate ↵
        type
1609     OAI21b1_types.append(new_OAI21b1_type)
1610     type_number = 1
1611
1612     for i in range(1, len(OAI21b1s)): #compare the ↵
        other gates in global array
1613         for j in range(0, len(OAI21b1_types)):
1614             if OAI21b1s[i].widths() == (OAI21b1_types[↵
                j]).OAI21b1_object()).widths():
1615                 if OAI21b1s[i].lengths() == (↵
                    OAI21b1_types[j].OAI21b1_object()).↵
                    lengths():
1616                     break
1617             else:
1618                 if j == len(OAI21b1_types)-1: #if not↵
                    same type as already in array, ↵
                    create new type
1619                     new_OAI21b1_type = OAI21b1_type(↵
                        OAI21b1s[i])
1620                     OAI21b1_types.append(↵
                        new_OAI21b1_type)
1621                     OAI21b1s[i].OAI21b1_type = ↵
                        type_number
1622                     OAI21b1_types[j+1].OAI21b1_type = ↵
                        type_number
1623                     type_number = type_number + 1
1624
1625 if len(removed_OAI21b1s_for_type_finding) != 0: #↵
    repeat process for subgates that were deleted. ↵
    Needed for subcircuit definitions
1626     if len(OAI21b1_types) == 0:
1627         new_OAI21b1_type = OAI21b1_type(↵
            removed_OAI21b1s_for_type_finding[0])
1628         OAI21b1_types.append(new_OAI21b1_type)
1629         type_number = 1
1630     for i in range(1, len(↵
        removed_OAI21b1s_for_type_finding)):
1631         for j in range(0, len(OAI21b1_types)):
1632             if removed_OAI21b1s_for_type_finding[i].↵
                widths() == (OAI21b1_types[j].↵
                    OAI21b1_object()).widths():

```

```

1633         if removed_OAI21b1s_for_type_finding[i]
1634             ].lengths() == (OAI21b1_types[j].
1635                 OAI21b1_object()).lengths():
1636             break
1637         else:
1638             if j == len(OAI21b1_types)-1:
1639                 new_OAI21b1_type = OAI21b1_type(
1640                     removed_OAI21b1s_for_type_finding
1641                     [i])
1642                 OAI21b1_types.append(
1643                     new_OAI21b1_type)
1644                 removed_OAI21b1s_for_type_finding[
1645                     i].OAI21b1_type = type_number
1646                 OAI21b1_types[j+1].OAI21b1_type =
1647                     type_number
1648                 type_number = type_number + 1
1649
1650 def find_OAI21b0b1_types():
1651
1652     if len(OAI21b0b1s) != 0: #only proceed of this type of
1653         gate exists in circuit
1654
1655         new_OAI21b0b1_type = OAI21b0b1_type(OAI21b0b1s[0])
1656         #make first gate in global array the first
1657         gate type
1658         OAI21b0b1_types.append(new_OAI21b0b1_type)
1659         type_number = 1
1660
1661         for i in range(1, len(OAI21b0b1s)): #compare the
1662             other gates in global array
1663             for j in range(0, len(OAI21b0b1_types)):
1664                 if OAI21b0b1s[i].widths() == (
1665                     OAI21b0b1_types[j].OAI21b0b1_object()).
1666                     widths():
1667                     if OAI21b0b1s[i].lengths() == (
1668                         OAI21b0b1_types[j].OAI21b0b1_object
1669                         ()).lengths():
1670                         break
1671                 else:
1672                     if j == len(OAI21b0b1_types)-1: #if
1673                         not same type as already in array,
1674                         create new type

```

```

1659         new_OAI21b0b1_type = ←
1660             OAI21b0b1_type(OAI21b0b1s[i])
1661         OAI21b0b1_types.append(←
1662             new_OAI21b0b1_type)
1663         OAI21b0b1s[i].OAI21b0b1_type = ←
1664             type_number
1665         OAI21b0b1_types[j+1].←
1666             OAI21b0b1_type = type_number
1667         type_number = type_number + 1
1668
1669     if len(removed_OAI21b0b1s_for_type_finding) != 0: #←
1670         repeat process for subgates that were deleted. ←
1671         Needed for subcircuit definitions
1672         if len(OAI21b0b1_types) == 0:
1673             new_OAI21b0b1_type = OAI21b0b1_type(←
1674                 removed_OAI21b0b1s_for_type_finding[0])
1675             OAI21b0b1_types.append(new_OAI21b0b1_type)
1676             type_number = 1
1677         for i in range(1, len(←
1678             removed_OAI21b0b1s_for_type_finding)):
1679             for j in range(0, len(OAI21b0b1_types)):
1680                 if removed_OAI21b0b1s_for_type_finding[i].←
1681                     widths() == (OAI21b0b1_types[j].←
1682                         OAI21b0b1_object()).widths():
1683                     if removed_OAI21b0b1s_for_type_finding←
1684                         [i].lengths() == (OAI21b0b1_types[j]←
1685                             ].OAI21b0b1_object()).lengths():
1686                         break
1687                 else:
1688                     if j == len(OAI21b0b1_types)-1:
1689                         new_OAI21b0b1_type = ←
1690                             OAI21b0b1_type(←
1691                                 removed_OAI21b0b1s_for_type_finding←
1692                                     [i])
1693                         OAI21b0b1_types.append(←
1694                             new_OAI21b0b1_type)
1695                         removed_OAI21b0b1s_for_type_finding←
1696                             [i].OAI21b0b1_type = ←
1697                                 type_number
1698                         OAI21b0b1_types[j+1].←
1699                             OAI21b0b1_type = type_number
1700                         type_number = type_number + 1

```

```

1683
1684 def remove_cell_transistors():
1685     ##This function removes the transistors that compose gates↵
        from the components list of the input netlist
1686     remove_OAI21b0b1_transistors_from_components()
1687     remove_OAI21b1_transistors_from_components()
1688     remove_OAI21_transistors_from_components()
1689     remove_NAND2b0_transistors_from_components()
1690     remove_OR2_transistors_from_components()
1691     remove_AND2_transistors_from_components()
1692     remove_NOR2_transistors_from_components()
1693     remove_NAND2_transistors_from_components()
1694     remove_inverter_transistors_from_components()
1695
1696 def add_cells_to_components():
1697     ##This function inserts the gates into the components list
1698     component_number = 0
1699     component_number = add_OAI21b0b1s_to_components(↵
        component_number)
1700     component_number = add_OAI21b1s_to_components(↵
        component_number)
1701     component_number = add_OAI21s_to_components(↵
        component_number)
1702     component_number = add_NAND2b0s_to_components(↵
        component_number)
1703     component_number = add_OR2s_to_components(↵
        component_number)
1704     component_number = add_AND2s_to_components(↵
        component_number)
1705     component_number = add_NOR2s_to_components(↵
        component_number)
1706     component_number = add_NAND2s_to_components(↵
        component_number)
1707     component_number = add_inverters_to_components(↵
        component_number)
1708
1709 def write_cells_to_netlist(file_name, before_components, ↵
        subcircuit_instance, components_list, after_components)↵
        :
1710     ##This function creates the subcircuit definitions of the ↵
        gates
1711     write_OAI21b0b1_to_netlist(subcircuit_instance)
1712     write_OAI21b1_to_netlist(subcircuit_instance)

```

```

1713 write_OAI21_to_netlist(subcircuit_instance)
1714 write_NAND2b0_to_netlist(subcircuit_instance)
1715 write_OR2_to_netlist(subcircuit_instance)
1716 write_AND2_to_netlist(subcircuit_instance)
1717 write_NOR2_to_netlist(subcircuit_instance)
1718 write_NAND2_to_netlist(subcircuit_instance)
1719 write_inverter_to_netlist(subcircuit_instance)
1720 netlist = ("".join(before_components)) + ("".join(↵
    subcircuit_instance)) + ("".join(components_list)) ↵
    + ("".join(after_components))
1721 os.remove(file_name[:file_name.find(".txt")] + '↵
    _hierarchical.txt')
1722 f = open(file_name[:file_name.find(".txt")] + '↵
    _hierarchical.txt', 'w')
1723 f.write(netlist)
1724 f.close()
1725
1726
1727
1728
1729
1730
1731 #####Fourth level functions
1732
1733 def remove_transistors(NMOS_index_to_remove, ↵
    PMOS_index_to_remove):
1734 ##This function removes the transistors that are used in ↵
    gates from the global array
1735
1736 NMOS_index_to_remove_sorted = sorted(↵
    NMOS_index_to_remove)
1737 PMOS_index_to_remove_sorted = sorted(↵
    PMOS_index_to_remove)
1738
1739 for i in range(len(NMOS_index_to_remove_sorted)-1, -1,↵
    -1):
1740     del NMOS_tx[NMOS_index_to_remove_sorted[i]]
1741
1742 for i in range(len(PMOS_index_to_remove_sorted)-1, -1,↵
    -1):
1743     del PMOS_tx[PMOS_index_to_remove_sorted[i]]
1744
1745

```

```

1746 def remove_OAI21b0b1_transistors_from_components():
1747     ##Remove transistors belonging to OAI21b0b1 gates from ←
    components_list
1748
1749     to_delete = []
1750
1751     for j in range(len(components_list)-1):
1752         for k in range(len(OAI21b0b1s)):
1753             if components_list[j] == (OAI21b0b1s[k].←
    inverter_object).NMOS_lines()[0]:
1754                 if len(to_delete) == 0:
1755                     to_delete.append(j)
1756                     to_delete.append(j+1)
1757                 elif to_delete[len(to_delete)-1] != j:
1758                     to_delete.append(j)
1759                     to_delete.append(j+1)
1760             elif components_list[j] == (OAI21b0b1s[k].←
    inverter_object).PMOS_lines()[0]:
1761                 if len(to_delete) == 0:
1762                     to_delete.append(j)
1763                     to_delete.append(j+1)
1764                 elif to_delete[len(to_delete)-1] != j:
1765                     to_delete.append(j)
1766                     to_delete.append(j+1)
1767             elif components_list[j] == ((OAI21b0b1s[k]←
    ].OR2_object).NOR2_object).NMOS_0_lines←
    ()[0]:
1768                 if len(to_delete) == 0:
1769                     to_delete.append(j)
1770                     to_delete.append(j+1)
1771                 elif to_delete[len(to_delete)-1] != j:
1772                     to_delete.append(j)
1773                     to_delete.append(j+1)
1774             elif components_list[j] == ((OAI21b0b1s[k]←
    ].OR2_object).NOR2_object).NMOS_1_lines←
    ()[0]:
1775                 if len(to_delete) == 0:
1776                     to_delete.append(j)
1777                     to_delete.append(j+1)
1778                 elif to_delete[len(to_delete)-1] != j:
1779                     to_delete.append(j)
1780                     to_delete.append(j+1)

```

```

1781 elif components_list[j] == ((OAI21b0b1s[k←
    ].OR2_object).NOR2_object).PMOS_0_lines←
    ()[0]:
1782     if len(to_delete) == 0:
1783         to_delete.append(j)
1784         to_delete.append(j+1)
1785     elif to_delete[len(to_delete)-1] != j:
1786         to_delete.append(j)
1787         to_delete.append(j+1)
1788 elif components_list[j] == ((OAI21b0b1s[k←
    ].OR2_object).NOR2_object).PMOS_1_lines←
    ()[0]:
1789     if len(to_delete) == 0:
1790         to_delete.append(j)
1791         to_delete.append(j+1)
1792     elif to_delete[len(to_delete)-1] != j:
1793         to_delete.append(j)
1794         to_delete.append(j+1)
1795 elif components_list[j] == ((OAI21b0b1s[k←
    ].OR2_object).inverter_object).←
    NMOS_lines()[0]:
1796     if len(to_delete) == 0:
1797         to_delete.append(j)
1798         to_delete.append(j+1)
1799     elif to_delete[len(to_delete)-1] != j:
1800         to_delete.append(j)
1801         to_delete.append(j+1)
1802 elif components_list[j] == ((OAI21b0b1s[k←
    ].OR2_object).inverter_object).←
    PMOS_lines()[0]:
1803     if len(to_delete) == 0:
1804         to_delete.append(j)
1805         to_delete.append(j+1)
1806     elif to_delete[len(to_delete)-1] != j:
1807         to_delete.append(j)
1808         to_delete.append(j+1)
1809 elif components_list[j] == ((OAI21b0b1s[k←
    ].AND2_object).NAND2_object).←
    NMOS_0_lines()[0]:
1810     if len(to_delete) == 0:
1811         to_delete.append(j)
1812         to_delete.append(j+1)
1813     elif to_delete[len(to_delete)-1] != j:

```

```

1814         to_delete.append(j)
1815         to_delete.append(j+1)
1816     elif components_list[j] == ((OAI21b0b1s[k↵
        ].AND2_object).NAND2_object).↵
        NMOS_1_lines()[0]:
1817         if len(to_delete) == 0:
1818             to_delete.append(j)
1819             to_delete.append(j+1)
1820             elif to_delete[len(to_delete)-1] != j:
1821                 to_delete.append(j)
1822                 to_delete.append(j+1)
1823     elif components_list[j] == ((OAI21b0b1s[k↵
        ].AND2_object).NAND2_object).↵
        PMOS_0_lines()[0]:
1824         if len(to_delete) == 0:
1825             to_delete.append(j)
1826             to_delete.append(j+1)
1827             elif to_delete[len(to_delete)-1] != j:
1828                 to_delete.append(j)
1829                 to_delete.append(j+1)
1830     elif components_list[j] == ((OAI21b0b1s[k↵
        ].AND2_object).NAND2_object).↵
        PMOS_1_lines()[0]:
1831         if len(to_delete) == 0:
1832             to_delete.append(j)
1833             to_delete.append(j+1)
1834             elif to_delete[len(to_delete)-1] != j:
1835                 to_delete.append(j)
1836                 to_delete.append(j+1)
1837     elif components_list[j] == ((OAI21b0b1s[k↵
        ].AND2_object).inverter_object).↵
        NMOS_lines()[0]:
1838         if len(to_delete) == 0:
1839             to_delete.append(j)
1840             to_delete.append(j+1)
1841             elif to_delete[len(to_delete)-1] != j:
1842                 to_delete.append(j)
1843                 to_delete.append(j+1)
1844     elif components_list[j] == ((OAI21b0b1s[k↵
        ].AND2_object).inverter_object).↵
        PMOS_lines()[0]:
1845         if len(to_delete) == 0:
1846             to_delete.append(j)

```

```

1847         to_delete.append(j+1)
1848         elif to_delete[len(to_delete)-1] != j:
1849             to_delete.append(j)
1850             to_delete.append(j+1)
1851
1852     for i in range(len(to_delete)-1, -1, -1):
1853         del components_list[to_delete[i]]
1854
1855
1856 def remove_OAI21b1_transistors_from_components():
1857     ##Remove transistors belonging to OAI21b1 gates from ↵
1858     components list
1859
1860     to_delete = []
1861
1862     for j in range(len(components_list)-1):
1863         for k in range(len(OAI21b1s)):
1864             if components_list[j] == (OAI21b1s[k].↵
1865                 inverter_object).NMOS_lines()[0]:
1866                 if len(to_delete) == 0:
1867                     to_delete.append(j)
1868                     to_delete.append(j+1)
1869                     elif to_delete[len(to_delete)-1] != j:
1870                         to_delete.append(j)
1871                         to_delete.append(j+1)
1872             elif components_list[j] == (OAI21b1s[k].↵
1873                 inverter_object).PMOS_lines()[0]:
1874                 if len(to_delete) == 0:
1875                     to_delete.append(j)
1876                     to_delete.append(j+1)
1877                     elif to_delete[len(to_delete)-1] != j:
1878                         to_delete.append(j)
1879                         to_delete.append(j+1)
1880             elif components_list[j] == ((OAI21b1s[k].↵
1881                 OR2_object).NOR2_object).NMOS_0_lines()↵
1882                 [0]:
1883                 if len(to_delete) == 0:
1884                     to_delete.append(j)
1885                     to_delete.append(j+1)
1886                     elif to_delete[len(to_delete)-1] != j:
1887                         to_delete.append(j)
1888                         to_delete.append(j+1)

```

```

1884 elif components_list[j] == ((OAI21b1s[k].←
      OR2_object).NOR2_object).NMOS_1_lines()←
      [0]:
1885     if len(to_delete) == 0:
1886         to_delete.append(j)
1887         to_delete.append(j+1)
1888     elif to_delete[len(to_delete)-1] != j:
1889         to_delete.append(j)
1890         to_delete.append(j+1)
1891 elif components_list[j] == ((OAI21b1s[k].←
      OR2_object).NOR2_object).PMOS_0_lines()←
      [0]:
1892     if len(to_delete) == 0:
1893         to_delete.append(j)
1894         to_delete.append(j+1)
1895     elif to_delete[len(to_delete)-1] != j:
1896         to_delete.append(j)
1897         to_delete.append(j+1)
1898 elif components_list[j] == ((OAI21b1s[k].←
      OR2_object).NOR2_object).PMOS_1_lines()←
      [0]:
1899     if len(to_delete) == 0:
1900         to_delete.append(j)
1901         to_delete.append(j+1)
1902     elif to_delete[len(to_delete)-1] != j:
1903         to_delete.append(j)
1904         to_delete.append(j+1)
1905 elif components_list[j] == ((OAI21b1s[k].←
      OR2_object).inverter_object).NMOS_lines←
      ()[0]:
1906     if len(to_delete) == 0:
1907         to_delete.append(j)
1908         to_delete.append(j+1)
1909     elif to_delete[len(to_delete)-1] != j:
1910         to_delete.append(j)
1911         to_delete.append(j+1)
1912 elif components_list[j] == ((OAI21b1s[k].←
      OR2_object).inverter_object).PMOS_lines←
      ()[0]:
1913     if len(to_delete) == 0:
1914         to_delete.append(j)
1915         to_delete.append(j+1)
1916     elif to_delete[len(to_delete)-1] != j:

```

```

1917         to_delete.append(j)
1918         to_delete.append(j+1)
1919     elif components_list[j] == (OAI21b1s[k].↵
        NAND2_object).NMOS_0_lines()[0]:
1920         if len(to_delete) == 0:
1921             to_delete.append(j)
1922             to_delete.append(j+1)
1923         elif to_delete[len(to_delete)-1] != j:
1924             to_delete.append(j)
1925             to_delete.append(j+1)
1926     elif components_list[j] == (OAI21b1s[k].↵
        NAND2_object).NMOS_1_lines()[0]:
1927         if len(to_delete) == 0:
1928             to_delete.append(j)
1929             to_delete.append(j+1)
1930         elif to_delete[len(to_delete)-1] != j:
1931             to_delete.append(j)
1932             to_delete.append(j+1)
1933     elif components_list[j] == (OAI21b1s[k].↵
        NAND2_object).PMOS_0_lines()[0]:
1934         if len(to_delete) == 0:
1935             to_delete.append(j)
1936             to_delete.append(j+1)
1937         elif to_delete[len(to_delete)-1] != j:
1938             to_delete.append(j)
1939             to_delete.append(j+1)
1940     elif components_list[j] == (OAI21b1s[k].↵
        NAND2_object).PMOS_1_lines()[0]:
1941         if len(to_delete) == 0:
1942             to_delete.append(j)
1943             to_delete.append(j+1)
1944         elif to_delete[len(to_delete)-1] != j:
1945             to_delete.append(j)
1946             to_delete.append(j+1)
1947
1948     for i in range(len(to_delete)-1, -1, -1):
1949         del components_list[to_delete[i]]
1950
1951
1952 def remove_OAI21_transistors_from_components():
1953     ##Remove transistors belonging to OAI21 gates from ↵
        components list
1954

```

```

1955 to_delete = []
1956
1957 for j in range(len(components_list)-1):
1958     for k in range(len(OAI21s)):
1959         if components_list[j] == ((OAI21s[k].↵
OR2_object).NOR2_object).NMOS_0_lines()↵
[0]:
1960             if len(to_delete) == 0:
1961                 to_delete.append(j)
1962                 to_delete.append(j+1)
1963             elif to_delete[len(to_delete)-1] != j:
1964                 to_delete.append(j)
1965                 to_delete.append(j+1)
1966         elif components_list[j] == ((OAI21s[k].↵
OR2_object).NOR2_object).NMOS_1_lines()↵
[0]:
1967             if len(to_delete) == 0:
1968                 to_delete.append(j)
1969                 to_delete.append(j+1)
1970             elif to_delete[len(to_delete)-1] != j:
1971                 to_delete.append(j)
1972                 to_delete.append(j+1)
1973         elif components_list[j] == ((OAI21s[k].↵
OR2_object).NOR2_object).PMOS_0_lines()↵
[0]:
1974             if len(to_delete) == 0:
1975                 to_delete.append(j)
1976                 to_delete.append(j+1)
1977             elif to_delete[len(to_delete)-1] != j:
1978                 to_delete.append(j)
1979                 to_delete.append(j+1)
1980         elif components_list[j] == ((OAI21s[k].↵
OR2_object).NOR2_object).PMOS_1_lines()↵
[0]:
1981             if len(to_delete) == 0:
1982                 to_delete.append(j)
1983                 to_delete.append(j+1)
1984             elif to_delete[len(to_delete)-1] != j:
1985                 to_delete.append(j)
1986                 to_delete.append(j+1)
1987         elif components_list[j] == ((OAI21s[k].↵
OR2_object).inverter_object).NMOS_lines↵
() [0]:

```

```

1988         if len(to_delete) == 0:
1989             to_delete.append(j)
1990             to_delete.append(j+1)
1991         elif to_delete[len(to_delete)-1] != j:
1992             to_delete.append(j)
1993             to_delete.append(j+1)
1994     elif components_list[j] == ((OAI21s[k].↵
        OR2_object).inverter_object).PMOS_lines↵
       ())[0]:
1995         if len(to_delete) == 0:
1996             to_delete.append(j)
1997             to_delete.append(j+1)
1998         elif to_delete[len(to_delete)-1] != j:
1999             to_delete.append(j)
2000             to_delete.append(j+1)
2001     elif components_list[j] == (OAI21s[k].↵
        NAND2_object).NMOS_0_lines())[0]:
2002         if len(to_delete) == 0:
2003             to_delete.append(j)
2004             to_delete.append(j+1)
2005         elif to_delete[len(to_delete)-1] != j:
2006             to_delete.append(j)
2007             to_delete.append(j+1)
2008     elif components_list[j] == (OAI21s[k].↵
        NAND2_object).NMOS_1_lines())[0]:
2009         if len(to_delete) == 0:
2010             to_delete.append(j)
2011             to_delete.append(j+1)
2012         elif to_delete[len(to_delete)-1] != j:
2013             to_delete.append(j)
2014             to_delete.append(j+1)
2015     elif components_list[j] == (OAI21s[k].↵
        NAND2_object).PMOS_0_lines())[0]:
2016         if len(to_delete) == 0:
2017             to_delete.append(j)
2018             to_delete.append(j+1)
2019         elif to_delete[len(to_delete)-1] != j:
2020             to_delete.append(j)
2021             to_delete.append(j+1)
2022     elif components_list[j] == (OAI21s[k].↵
        NAND2_object).PMOS_1_lines())[0]:
2023         if len(to_delete) == 0:
2024             to_delete.append(j)

```

```

2025         to_delete.append(j+1)
2026         elif to_delete[len(to_delete)-1] != j:
2027             to_delete.append(j)
2028             to_delete.append(j+1)
2029
2030     for i in range(len(to_delete)-1, -1, -1):
2031         del components_list[to_delete[i]]
2032
2033
2034 def remove_NAND2b0_transistors_from_components():
2035     ##Remove transistors belonging to NAND2b0 gates from ↵
2036     components list
2037     to_delete = []
2038
2039     for j in range(len(components_list)-1):
2040         for k in range(len(NAND2b0s)):
2041             if components_list[j] == ((NAND2b0s[k].↵
2042                 OR2_object).NOR2_object).NMOS_0_lines()↵
2043                 [0]:
2044                 if len(to_delete) == 0:
2045                     to_delete.append(j)
2046                     to_delete.append(j+1)
2047                     elif to_delete[len(to_delete)-1] != j:
2048                         to_delete.append(j)
2049                         to_delete.append(j+1)
2050             elif components_list[j] == ((NAND2b0s[k].↵
2051                 OR2_object).NOR2_object).NMOS_1_lines()↵
2052                 [0]:
2053                 if len(to_delete) == 0:
2054                     to_delete.append(j)
2055                     to_delete.append(j+1)
2056                     elif to_delete[len(to_delete)-1] != j:
2057                         to_delete.append(j)
2058                         to_delete.append(j+1)
2059             elif components_list[j] == ((NAND2b0s[k].↵
2060                 OR2_object).NOR2_object).PMOS_0_lines()↵
                [0]:
                if len(to_delete) == 0:
                    to_delete.append(j)
                    to_delete.append(j+1)
                    elif to_delete[len(to_delete)-1] != j:
                        to_delete.append(j)
                        to_delete.append(j+1)

```

```

2061 elif components_list[j] == ((NAND2b0s[k].↵
    OR2_object).NOR2_object).PMOS_1_lines()↵
    [0]:
2062     if len(to_delete) == 0:
2063         to_delete.append(j)
2064         to_delete.append(j+1)
2065     elif to_delete[len(to_delete)-1] != j:
2066         to_delete.append(j)
2067         to_delete.append(j+1)
2068 elif components_list[j] == ((NAND2b0s[k].↵
    OR2_object).inverter_object).NMOS_lines↵
    ()[0]:
2069     if len(to_delete) == 0:
2070         to_delete.append(j)
2071         to_delete.append(j+1)
2072     elif to_delete[len(to_delete)-1] != j:
2073         to_delete.append(j)
2074         to_delete.append(j+1)
2075 elif components_list[j] == ((NAND2b0s[k].↵
    OR2_object).inverter_object).PMOS_lines↵
    ()[0]:
2076     if len(to_delete) == 0:
2077         to_delete.append(j)
2078         to_delete.append(j+1)
2079     elif to_delete[len(to_delete)-1] != j:
2080         to_delete.append(j)
2081         to_delete.append(j+1)
2082 elif components_list[j] == (NAND2b0s[k].↵
    inverter_object).NMOS_lines()[0]:
2083     if len(to_delete) == 0:
2084         to_delete.append(j)
2085         to_delete.append(j+1)
2086     elif to_delete[len(to_delete)-1] != j:
2087         to_delete.append(j)
2088         to_delete.append(j+1)
2089 elif components_list[j] == (NAND2b0s[k].↵
    inverter_object).PMOS_lines()[0]:
2090     if len(to_delete) == 0:
2091         to_delete.append(j)
2092         to_delete.append(j+1)
2093     elif to_delete[len(to_delete)-1] != j:
2094         to_delete.append(j)
2095         to_delete.append(j+1)

```

```

2096
2097     for i in range(len(to_delete)-1, -1, -1):
2098         del components_list[to_delete[i]]
2099
2100
2101 def remove_OR2_transistors_from_components():
2102     ##Remove transistors belonging to OR2 gates from ↵
2103     components list
2104
2105     to_delete = []
2106
2107     for j in range(len(components_list)-1):
2108         for k in range(len(OR2s)):
2109             if components_list[j] == (OR2s[k].↵
2110                 NOR2_object).NMOS_0_lines()[0]:
2111                 if len(to_delete) == 0:
2112                     to_delete.append(j)
2113                     to_delete.append(j+1)
2114                 elif to_delete[len(to_delete)-1] != j:
2115                     to_delete.append(j)
2116                     to_delete.append(j+1)
2117             elif components_list[j] == (OR2s[k].↵
2118                 NOR2_object).NMOS_1_lines()[0]:
2119                 if len(to_delete) == 0:
2120                     to_delete.append(j)
2121                     to_delete.append(j+1)
2122                 elif to_delete[len(to_delete)-1] != j:
2123                     to_delete.append(j)
2124                     to_delete.append(j+1)
2125             elif components_list[j] == (OR2s[k].↵
2126                 NOR2_object).PMOS_0_lines()[0]:
2127                 if len(to_delete) == 0:
2128                     to_delete.append(j)
2129                     to_delete.append(j+1)
2130                 elif to_delete[len(to_delete)-1] != j:
2131                     to_delete.append(j)
2132                     to_delete.append(j+1)
2133                 elif to_delete[len(to_delete)-1] != j:

```

```

2134         to_delete.append(j)
2135         to_delete.append(j+1)
2136     elif components_list[j] == (OR2s[k].↵
        inverter_object).NMOS_lines()[0]:
2137         if len(to_delete) == 0:
2138             to_delete.append(j)
2139             to_delete.append(j+1)
2140         elif to_delete[len(to_delete)-1] != j:
2141             to_delete.append(j)
2142             to_delete.append(j+1)
2143     elif components_list[j] == (OR2s[k].↵
        inverter_object).PMOS_lines()[0]:
2144         if len(to_delete) == 0:
2145             to_delete.append(j)
2146             to_delete.append(j+1)
2147         elif to_delete[len(to_delete)-1] != j:
2148             to_delete.append(j)
2149             to_delete.append(j+1)
2150
2151     for i in range(len(to_delete)-1, -1, -1):
2152         del components_list[to_delete[i]]
2153
2154 def remove_AND2_transistors_from_components():
2155     ##Remove transistors belonging to AND2 gates from ↵
        components list
2156
2157     to_delete = []
2158
2159     for j in range(len(components_list)-1):
2160         for k in range(len(AND2s)):
2161             if components_list[j] == (AND2s[k].↵
                NAND2_object).NMOS_0_lines()[0]:
2162                 if len(to_delete) == 0:
2163                     to_delete.append(j)
2164                     to_delete.append(j+1)
2165                 elif to_delete[len(to_delete)-1] != j:
2166                     to_delete.append(j)
2167                     to_delete.append(j+1)
2168             elif components_list[j] == (AND2s[k].↵
                NAND2_object).NMOS_1_lines()[0]:
2169                 if len(to_delete) == 0:
2170                     to_delete.append(j)
2171                     to_delete.append(j+1)

```

```

2172         elif to_delete[len(to_delete)-1] != j:
2173             to_delete.append(j)
2174             to_delete.append(j+1)
2175     elif components_list[j] == (AND2s[k].↵
        NAND2_object).PMOS_0_lines()[0]:
2176         if len(to_delete) == 0:
2177             to_delete.append(j)
2178             to_delete.append(j+1)
2179         elif to_delete[len(to_delete)-1] != j:
2180             to_delete.append(j)
2181             to_delete.append(j+1)
2182     elif components_list[j] == (AND2s[k].↵
        NAND2_object).PMOS_1_lines()[0]:
2183         if len(to_delete) == 0:
2184             to_delete.append(j)
2185             to_delete.append(j+1)
2186         elif to_delete[len(to_delete)-1] != j:
2187             to_delete.append(j)
2188             to_delete.append(j+1)
2189     elif components_list[j] == (AND2s[k].↵
        inverter_object).NMOS_lines()[0]:
2190         if len(to_delete) == 0:
2191             to_delete.append(j)
2192             to_delete.append(j+1)
2193         elif to_delete[len(to_delete)-1] != j:
2194             to_delete.append(j)
2195             to_delete.append(j+1)
2196     elif components_list[j] == (AND2s[k].↵
        inverter_object).PMOS_lines()[0]:
2197         if len(to_delete) == 0:
2198             to_delete.append(j)
2199             to_delete.append(j+1)
2200         elif to_delete[len(to_delete)-1] != j:
2201             to_delete.append(j)
2202             to_delete.append(j+1)
2203
2204     for i in range(len(to_delete)-1, -1, -1):
2205         del components_list[to_delete[i]]
2206
2207
2208 def remove_NOR2_transistors_from_components():
2209     ##Remove transistors belonging to NOR2 gates from ↵
        components list

```

```

2210
2211     to_delete = []
2212
2213     for j in range(len(components_list)-1):
2214         for k in range(len(NOR2s)):
2215             if components_list[j] == NOR2s[k].↵
2216                 NMOS_0_lines()[0]:
2217                     if len(to_delete) == 0:
2218                         to_delete.append(j)
2219                         to_delete.append(j+1)
2220                     elif to_delete[len(to_delete)-1] != j:
2221                         to_delete.append(j)
2222                         to_delete.append(j+1)
2223             elif components_list[j] == NOR2s[k].↵
2224                 NMOS_1_lines()[0]:
2225                     if len(to_delete) == 0:
2226                         to_delete.append(j)
2227                         to_delete.append(j+1)
2228                     elif to_delete[len(to_delete)-1] != j:
2229                         to_delete.append(j)
2230                         to_delete.append(j+1)
2231             elif components_list[j] == NOR2s[k].↵
2232                 PMOS_0_lines()[0]:
2233                     if len(to_delete) == 0:
2234                         to_delete.append(j)
2235                         to_delete.append(j+1)
2236                     elif to_delete[len(to_delete)-1] != j:
2237                         to_delete.append(j)
2238                         to_delete.append(j+1)
2239             elif components_list[j] == NOR2s[k].↵
2240                 PMOS_1_lines()[0]:
2241                     if len(to_delete) == 0:
2242                         to_delete.append(j)
2243                         to_delete.append(j+1)
2244                     elif to_delete[len(to_delete)-1] != j:
2245                         to_delete.append(j)
2246                         to_delete.append(j+1)
2247
2248         for i in range(len(to_delete)-1, -1, -1):
2249             del components_list[to_delete[i]]
2250
2251 def remove_NAND2_transistors_from_components():

```

```

2249 ##Remove transistors belonging to NAND2 gates from ↵
      components_list
2250
2251     to_delete = []
2252
2253     for j in range(len(components_list)-1):
2254         for k in range(len(NAND2s)):
2255             if components_list[j] == NAND2s[k].↵
                NMOS_0_lines()[0]:
2256                 if len(to_delete) == 0:
2257                     to_delete.append(j)
2258                     to_delete.append(j+1)
2259                 elif to_delete[len(to_delete)-1] != j:
2260                     to_delete.append(j)
2261                     to_delete.append(j+1)
2262             elif components_list[j] == NAND2s[k].↵
                NMOS_1_lines()[0]:
2263                 if len(to_delete) == 0:
2264                     to_delete.append(j)
2265                     to_delete.append(j+1)
2266                 elif to_delete[len(to_delete)-1] != j:
2267                     to_delete.append(j)
2268                     to_delete.append(j+1)
2269             elif components_list[j] == NAND2s[k].↵
                PMOS_0_lines()[0]:
2270                 if len(to_delete) == 0:
2271                     to_delete.append(j)
2272                     to_delete.append(j+1)
2273                 elif to_delete[len(to_delete)-1] != j:
2274                     to_delete.append(j)
2275                     to_delete.append(j+1)
2276             elif components_list[j] == NAND2s[k].↵
                PMOS_1_lines()[0]:
2277                 if len(to_delete) == 0:
2278                     to_delete.append(j)
2279                     to_delete.append(j+1)
2280                 elif to_delete[len(to_delete)-1] != j:
2281                     to_delete.append(j)
2282                     to_delete.append(j+1)
2283
2284     for i in range(len(to_delete)-1, -1, -1):
2285         del components_list[to_delete[i]]
2286

```

```

2287 def remove_inverter_transistors_from_components():
2288     ##Remove transistors belonging to inverters in from ↵
        components_list
2289
2290     to_delete = []
2291
2292     for j in range(len(components_list)):
2293         for k in range(len(inverters)):
2294             if components_list[j] == inverters[k].↵
                NMOS_lines()[0]:
2295                 if len(to_delete) == 0:
2296                     to_delete.append(j)
2297                     to_delete.append(j+1)
2298                 elif to_delete[len(to_delete)-1] != j:
2299                     to_delete.append(j)
2300                     to_delete.append(j+1)
2301             elif components_list[j] == inverters[k].↵
                PMOS_lines()[0]:
2302                 if len(to_delete) == 0:
2303                     to_delete.append(j)
2304                     to_delete.append(j+1)
2305                 elif to_delete[len(to_delete)-1] != j:
2306                     to_delete.append(j)
2307                     to_delete.append(j+1)
2308
2309
2310     for i in range(len(to_delete)-1, -1, -1):
2311         del components_list[to_delete[i]]
2312
2313
2314     to_delete = []
2315
2316 def add_OAI21b0b1s_to_components(input_component_number):
2317     ##add OAI21b0b1s to components list
2318     component_number = input_component_number
2319     for i in range(len(OAI21b0b1s)):
2320         components_list.insert(0, "I" + str(↵
            component_number) + " (" + str((OAI21b0b1s[i].↵
            inverter_object).NMOS_net()[2]) + " " + str((↵
            OAI21b0b1s[i].AND2_object).NAND2_object).↵
            NMOS_0_net()[1]) + " " + str((OAI21b0b1s[i].↵
            AND2_object).NAND2_object).NMOS_1_net()[1]) + "↵
            " + str((OAI21b0b1s[i].inverter_object).↵

```

```

2321         NMOS_net()[1]) + " " + str((OAI21b0b1s[i].↵
2322         inverter_object).PMOS_net()[2]) + " " + str((↵
2323         OAI21b0b1s[i].OR2_object).inverter_object).↵
2324         NMOS_net()[0]) + ") OAI21b0b1_type_%s \n" % str(↵
2325         (OAI21b0b1s[i].OAI21b0b1_type))
2326     component_number = component_number + 1
2327     return component_number
2328
2329 def add_OAI21b1s_to_components(input_component_number):
2330     ##add OAI21b1s to components list
2331     component_number = input_component_number
2332     for i in range(len(OAI21b1s)):
2333         components_list.insert(0, "I" + str(↵
2334         component_number) + " (" + str((OAI21b1s[i].↵
2335         inverter_object).NMOS_net()[2]) + " " + str((↵
2336         OAI21b1s[i].inverter_object).NMOS_net()[1]) + "↵
2337         " + str((OAI21b1s[i].NAND2_object).NMOS_1_net↵
2338         ()[1]) + " " + str(((OAI21b1s[i].OR2_object).↵
2339         NOR2_object).NMOS_1_net()[1]) + " " + str((↵
2340         OAI21b1s[i].inverter_object).PMOS_net()[2]) + "↵
2341         " + str((OAI21b1s[i].NAND2_object).NMOS_0_net↵
2342         ()[0]) + ") OAI21b1_type_%s \n" % str(OAI21b1s[↵
2343         i].OAI21b1_type))
2344     component_number = component_number + 1
2345     return component_number
2346
2347 def add_OAI21s_to_components(input_component_number):
2348     ##add OAI21s to components list
2349     component_number = input_component_number
2350     for i in range(len(OAI21s)):
2351         components_list.insert(0, "I" + str(↵
2352         component_number) + " (" + str((OAI21s[i].↵
2353         OR2_object).NOR2_object).NMOS_0_net()[2]) + " "↵
2354         + str((OAI21s[i].NAND2_object).NMOS_0_net()↵
2355         [1]) + " " + str(((OAI21s[i].OR2_object).↵
2356         NOR2_object).NMOS_0_net()[1]) + " " + str(((↵
2357         OAI21s[i].OR2_object).NOR2_object).NMOS_1_net()↵
2358         [1]) + " " + str(((OAI21s[i].OR2_object).↵
2359         inverter_object).PMOS_net()[2]) + " " + str((↵
2360         OAI21s[i].NAND2_object).NMOS_0_net()[0]) + ") ↵
2361         OAI21_type_%s \n" % str(OAI21s[i].OAI21_type))
2362     component_number = component_number + 1
2363     return component_number

```

```

2339
2340
2341 def add_NAND2b0s_to_components(input_component_number):
2342     ##add NAND2b0s to components list
2343     component_number = input_component_number
2344     for i in range(len(NAND2b0s)):
2345         components_list.insert(0, "I" + str(↵
            component_number) + " (" + str(((NAND2b0s[i].↵
            OR2_object).NOR2_object).NMOS_0_net()[1]) + " "↵
            + str((NAND2b0s[i].inverter_object).NMOS_net()↵
            [2]) + " " + str((NAND2b0s[i].inverter_object).↵
            NMOS_net()[1]) + " " + str((NAND2b0s[i].↵
            inverter_object).PMOS_net()[2]) + " " + str(((↵
            NAND2b0s[i].OR2_object).inverter_object).↵
            NMOS_net()[0]) + ") NAND2b0_type_%s \n" % str(↵
            NAND2b0s[i].NAND2b0_type))
2346         component_number = component_number + 1
2347     return component_number
2348
2349
2350 def add_OR2s_to_components(input_component_number):
2351     ##add OR2s to components list
2352     component_number = input_component_number
2353     for i in range(len(OR2s)):
2354         components_list.insert(0, "I" + str(↵
            component_number) + " (" + str((OR2s[i].↵
            NOR2_object).NMOS_0_net()[1]) + " " + str((OR2s↵
            [i].NOR2_object).NMOS_1_net()[1]) + " " + str((↵
            OR2s[i].NOR2_object).NMOS_1_net()[2]) + " " + ↵
            str((OR2s[i].NOR2_object).PMOS_0_net()[2]) + " ↵
            " + str((OR2s[i].inverter_object).NMOS_net()↵
            [0]) + ") OR2_type_%s \n" % str(OR2s[i].↵
            OR2_type))
2355         component_number = component_number + 1
2356     return component_number
2357
2358
2359 def add_AND2s_to_components(input_component_number):
2360     ##add AND2s to components list
2361     component_number = input_component_number
2362     for i in range(len(AND2s)):
2363         components_list.insert(0, "I" + str(↵
            component_number) + " (" + str((AND2s[i].↵

```

```

        NAND2_object).NMOS_0_net()[1]) + " " + str((←
        AND2s[i].NAND2_object).NMOS_1_net()[1]) + " " +←
        str((AND2s[i].NAND2_object).NMOS_1_net()[2]) +←
        " " + str((AND2s[i].NAND2_object).PMOS_0_net()←
        [2]) + " " + str((AND2s[i].inverter_object).←
        NMOS_net()[0]) + ") AND2_type_%s \n" % str(←
        AND2s[i].AND2_type))
2364         component_number = component_number + 1
2365     return component_number
2366
2367
2368 def add_NOR2s_to_components(input_component_number):
2369     ##add NOR2s to components list
2370     component_number = input_component_number
2371     for i in range(len(NOR2s)):
2372         components_list.insert(0, "I" + str(←
            component_number) + " (" + str(NOR2s[i].←
            NMOS_0_net()[1]) + " " + str(NOR2s[i].←
            NMOS_1_net()[1]) + " " + str(NOR2s[i].←
            NMOS_1_net()[2]) + " " + str(NOR2s[i].←
            PMOS_0_net()[2]) + " " + str(NOR2s[i].←
            PMOS_1_net()[0]) + ") NOR2_type_%s \n" % str(←
            NOR2s[i].NOR2_type))
2373         component_number = component_number + 1
2374     return component_number
2375
2376
2377 def add_NAND2s_to_components(input_component_number):
2378     ##add NAND2s to components list
2379     component_number = input_component_number
2380     for i in range(len(NAND2s)):
2381         components_list.insert(0, "I" + str(←
            component_number) + " (" + str(NAND2s[i].←
            NMOS_0_net()[1]) + " " + str(NAND2s[i].←
            NMOS_1_net()[1]) + " " + str(NAND2s[i].←
            NMOS_1_net()[2]) + " " + str(NAND2s[i].←
            PMOS_0_net()[2]) + " " + str(NAND2s[i].←
            PMOS_0_net()[0]) + ") NAND2_type_%s \n" % str(←
            NAND2s[i].NAND2_type))
2382         component_number = component_number + 1
2383     return component_number
2384
2385 def add_inverters_to_components(input_component_number):

```

```

2386 ##add inverters to components list
2387     component_number = input_component_number
2388     for i in range(len(inverters)):
2389         components_list.insert(0, "I" + str(↵
            component_number) + " (" + str(inverters[i].↵
            NMOS_net()[2]) + " " + str(inverters[i].↵
            NMOS_net()[1]) + " " + str(inverters[i].↵
            NMOS_net()[0]) + " " + str(inverters[i].↵
            PMOS_net()[2]) + ") inverter_type_%s \n" % str(↵
            inverters[i].inverter_type))
2390         component_number = component_number + 1
2391     return component_number
2392
2393
2394
2395 def write_OAI21b0b1_to_netlist(input_subcircuit_instance):
2396     ##append the subcircuit instance of each type of OAI21b0b1↵
    to the subcircuit instance list
2397
2398     subcircuit_instance = input_subcircuit_instance
2399
2400     for i in range(len(OAI21b0b1_types)):
2401
2402         string0 = "subckt OAI21b0b1_type_%s GND In_AND_1 ↵
            In_AND_2 In_INV VDD Z \n" % str(OAI21b0b1_types↵
            [i].OAI21b0b1_type)
2403         string1 = "I0 (GND In_INV %s VDD) inverter_type_%s↵
            \n" % (str(((OAI21b0b1_types[i].↵
            OAI21b0b1_object).inverter_object).NMOS_net()↵
            [0]) , str(((OAI21b0b1_types[i].↵
            OAI21b0b1_object).inverter_object).↵
            inverter_type))
2404         string2 = "I1 (In_AND_1 In_AND_2 GND VDD %s) ↵
            AND2_type_%s \n" % (str(((OAI21b0b1_types[i].↵
            OAI21b0b1_object).AND2_object).inverter_object)↵
            .NMOS_net()[0]), str(((OAI21b0b1_types[i].↵
            OAI21b0b1_object).AND2_object).AND2_type))
2405         string3 = "I2 (%s %s GND VDD Z) OR2_type_%s \n" % ↵
            (str(((OAI21b0b1_types[i].OAI21b0b1_object).↵
            OR2_object).NOR2_object).NMOS_0_net()[1]), str↵
            (((OAI21b0b1_types[i].OAI21b0b1_object).↵
            OR2_object).NOR2_object).NMOS_1_net()[1]), str↵

```

```

2406         (((OAI21b0b1_types[i].OAI21b0b1_object).OR2_object).OR2_type))
2407     string4 = "ends OAI21b0b1_type_%s \n// End of subcircuit definition. \n \n" % str(OAI21b0b1_types[i].OAI21b0b1_type)
2408     string = string0 + " " + string1 + " " + string2 + " " + string3 + string4
2409     subcircuit_instance.append(string)
2410
2411     return subcircuit_instance
2412
2413
2414 def write_OAI21b1_to_netlist(input_subcircuit_instance):
2415     ##append the subcircuit instance of each type of OAI21b1 to the subcircuit instance list
2416
2417     subcircuit_instance = input_subcircuit_instance
2418
2419     for i in range(len(OAI21b1_types)):
2420
2421         string0 = "subckt OAI21b1_type_%s GND In_INV In_NAND In_OR VDD Z \n" % str(OAI21b1_types[i].OAI21b1_type)
2422         string1 = "I1 (GND In %s VDD) inverter_type_%s \n" % (str(((OAI21b1_types[i].OAI21b1_object).inverter_object).NMOS_net()[0]), str(((OAI21b1_types[i].OAI21b1_object).inverter_object).inverter_type))
2423         string2 = "I2 (In_OR %s GND VDD %s) OR2_type_%s \n" % (str(((OAI21b1_types[i].OAI21b1_object).inverter_object).NMOS_net()[0]), str(((OAI21b1_types[i].OAI21b1_object).OR2_object).inverter_object).NMOS_net()[0]), str(((OAI21b1_types[i].OAI21b1_object).OR2_object).OR2_type))
2424         string3 = "I3 (In_NAND %s GND VDD Z) NAND2_type_%s \n" % (str(((OAI21b1_types[i].OAI21b1_object).OR2_object).inverter_object).NMOS_net()[0]), str(((OAI21b1_types[i].OAI21b1_object).NAND2_object).NAND2_type))

```

```

2425         string4 = "ends OAI21b1_type_%s \n// End of ↵
                subcircuit definition. \n \n" % str(↵
                OAI21b1_types[i].OAI21b1_type)
2426         string = string0 + "      " + string1 + "      " + ↵
                string2 + "      " + string3 + string4
2427         subcircuit_instance.append(string)
2428
2429     return subcircuit_instance
2430
2431
2432 def write_OAI21_to_netlist(input_subcircuit_instance):
2433     ##append the subcircuit instance of each type of OAI21 to ↵
        the subcircuit instance list
2434
2435     subcircuit_instance = input_subcircuit_instance
2436
2437     for i in range(len(OAI21_types)):
2438
2439         string0 = "subckt OAI21_type_%s GND In_NAND ↵
                In_OR_1 In_OR_2 VDD Z \n" % str(OAI21_types[i].↵
                OAI21_type)
2440         string1 = "I0 (In_OR_1 In_OR_2 GND VDD %s ↵
                OR2_type_%s \n" % (str((((OAI21_types[i].↵
                OAI21_object).OR2_object).inverter_object).↵
                NMOS_net()[0]), str(((OAI21_types[i].↵
                OAI21_object).OR2_object).OR2_type))
2441         string2 = "I1 (In_NAND %s GND VDD Z) NAND2_type_%s↵
                \n" % (str((((OAI21_types[i].OAI21_object).↵
                OR2_object).inverter_object).NMOS_net()[0]), ↵
                str(((OAI21_types[i].OAI21_object).NAND2_object↵
                ).NAND2_type))
2442         string3 = "ends OAI21_type_%s \n// End of ↵
                subcircuit definition. \n \n" % str(OAI21_types↵
                [i].OAI21_type)
2443         string = string0 + "      " + string1 + "      " + ↵
                string2 + string3
2444         subcircuit_instance.append(string)
2445     return subcircuit_instance
2446
2447
2448 def write_NAND2b0_to_netlist(input_subcircuit_instance):
2449     ##append the subcircuit instance of each type of NAND2b0 ↵
        to the subcircuit instance list

```

```

2450
2451 ##Write new subcircuit components to netlist
2452
2453     subcircuit_instance = input_subcircuit_instance
2454
2455     for i in range(len(NAND2b0_types)):
2456
2457         string0 = "subckt NAND2b0_type_%s A GND In VDD Z \↵
                n" % str(NAND2b0_types[i].NAND2b0_type)
2458         string1 = "I0 (A %s GND VDD Z) OR2_type_%s \n" % (↵
                str(((NAND2b0_types[i].NAND2b0_object).↵
                inverter_object).NMOS_net()[0]), str(((↵
                NAND2b0_types[i].NAND2b0_object).OR2_object).↵
                OR2_type))
2459         string2 = "I1 (GND In %s VDD) inverter_type_%s \n"↵
                % (str(((NAND2b0_types[i].NAND2b0_object).↵
                inverter_object).NMOS_net()[0]), str(((↵
                NAND2b0_types[i].NAND2b0_object).↵
                inverter_object).inverter_type))
2460         string3 = "ends NAND2b0_type_%s \n// End of ↵
                subcircuit definition. \n \n" % str(↵
                NAND2b0_types[i].NAND2b0_type)
2461         string = string0 + "      " + string1 + "      " + ↵
                string2 + string3
2462         subcircuit_instance.append(string)
2463
2464     return subcircuit_instance
2465
2466
2467
2468 def write_OR2_to_netlist(input_subcircuit_instance):
2469 ##append the subcircuit instance of each type of OR2 to ↵
    the subcircuit instance list
2470
2471 ##Write new subcircuit components to netlist
2472
2473     subcircuit_instance = input_subcircuit_instance
2474
2475     for i in range(len(OR2_types)):
2476
2477         string0 = "subckt OR2_type_%s A B GND VDD Z \n" % ↵
                str(OR2_types[i].OR2_type)

```

```

2478     string1 = "I0 (GND %s Z VDD) inverter_type_%s \n" % (
2479         str(((OR2_types[i].OR2_object).NMOS_net()[1]), str(((OR2_types[i].OR2_object).inverter_object).inverter_type))
2480     string2 = "I1 (A B GND VDD %s) NOR2_type_%s \n" % (
2481         str(((OR2_types[i].OR2_object).NOR2_object).NMOS_0_net()[0]), str(((OR2_types[i].OR2_object).NOR2_object).NOR2_type))
2482     string3 = "ends OR2_type_%s \n// End of subcircuit definition. \n \n" % str(OR2_types[i].OR2_type)
2483     string = string0 + " " + string1 + " " + string2 + string3
2484     subcircuit_instance.append(string)
2485
2486     return subcircuit_instance
2487
2488 def write_AND2_to_netlist(input_subcircuit_instance):
2489     ##append the subcircuit instance of each type of AND2 to the subcircuit instance list
2490
2491     ##Write new subcircuit components to netlist
2492
2493     subcircuit_instance = input_subcircuit_instance
2494
2495     for i in range(len(AND2_types)):
2496         string0 = "subckt AND2_type_%s A B GND VDD Z \n" % str(AND2_types[i].AND2_type)
2497         string1 = "I0 (GND %s Z VDD) inverter_type_%s \n" % (
2498             str(((AND2_types[i].AND2_object).inverter_object).NMOS_net()[1]), str(((AND2_types[i].AND2_object).inverter_object).inverter_type))
2499         string2 = "I1 (A B GND VDD %s) NAND2_type_%s \n" % (
2500             str(((AND2_types[i].AND2_object).NAND2_object).NMOS_0_net()[0]), str(((AND2_types[i].AND2_object).NAND2_object).NAND2_type))
2501         string3 = "ends AND2_type_%s \n// End of subcircuit definition. \n \n" % str(AND2_types[i].AND2_type)

```

```

2500         string = string0 + "      " + string1 + "      " + ↵
                string2 + string3
2501         subcircuit_instance.append(string)
2502
2503     return subcircuit_instance
2504
2505
2506 def write_NOR2_to_netlist(input_subcircuit_instance):
2507     ##append the subcircuit instance of each type of NOR2 to ↵
        the subcircuit instance list
2508
2509     ##Write new subcircuit components to netlist
2510     subcircuit_instance = input_subcircuit_instance
2511
2512     for i in range(len(NOR2_types)):
2513
2514         if (NOR2_types[i].NOR2_object()).NMOS_0_net()[2] ↵
            == '0':
2515             NMOS0net2 = "GND"
2516         else:
2517             NMOS0net2 = str((NOR2_types[i].NOR2_object()).↵
                NMOS_0_net()[2])
2518
2519         if (NOR2_types[i].NOR2_object()).NMOS_1_net()[2] ↵
            == '0':
2520             NMOS1net2 = "GND"
2521         else:
2522             NMOS1net2 = str((NOR2_types[i].NOR2_object()).↵
                NMOS_1_net()[2])
2523
2524         if (NOR2_types[i].NOR2_object()).NMOS_0_net()[3] ↵
            == '0':
2525             NMOS0net3 = "GND"
2526         else:
2527             NMOS0net3 = str((NOR2_types[i].NOR2_object()).↵
                NMOS_0_net()[3])
2528
2529         if (NOR2_types[i].NOR2_object()).NMOS_1_net()[3] ↵
            == '0':
2530             NMOS1net3 = "GND"
2531         else:
2532             NMOS1net3 = str((NOR2_types[i].NOR2_object()).↵
                NMOS_1_net()[3])

```

```

2533
2534 PMOS0net2 = str((NOR2_types[i].NOR2_object()).PMOS_0_net()[2])
2535 PMOS0net0 = str((NOR2_types[i].NOR2_object()).PMOS_0_net()[0])
2536
2537 if (NOR2_types[i].NOR2_object()).PMOS_0_net()[3] == PMOS0net2:
2538     PMOS0net3 = "VDD"
2539 else:
2540     PMOS0net3 = str((NOR2_types[i].NOR2_object()).PMOS_0_net()[3])
2541
2542 if (NOR2_types[i].NOR2_object()).PMOS_1_net()[2] == PMOS0net2:
2543     PMOS1net2 = "VDD"
2544 else:
2545     PMOS1net2 = str((NOR2_types[i].NOR2_object()).PMOS_1_net()[2])
2546
2547 if (NOR2_types[i].NOR2_object()).PMOS_1_net()[3] == PMOS0net2:
2548     PMOS1net3 = "VDD"
2549 else:
2550     PMOS1net3 = str((NOR2_types[i].NOR2_object()).PMOS_1_net()[3])
2551
2552 string0 = "subckt NOR2_type_%s A B GND VDD Z \n" % str(NOR2_types[i].NOR2_type)
2553 string1 = (NOR2_types[i].NOR2_object()).NMOS_1_lines()[0]
2554 string2 = "N1 (Z B %s %s)" % (NMOS1net2, NMOS1net3) + string1[(string1.find("(")+len("(")):]
2555 string3 = (NOR2_types[i].NOR2_object()).NMOS_1_lines()[1]
2556 string4 = (NOR2_types[i].NOR2_object()).NMOS_0_lines()[0]
2557 string5 = "N0 (Z A %s %s)" % (NMOS0net2, NMOS0net3) + string4[(string4.find("(")+len("(")):]
2558 string6 = (NOR2_types[i].NOR2_object()).NMOS_0_lines()[1]
2559 string7 = (NOR2_types[i].NOR2_object()).PMOS_1_lines()[0]

```

```

2560     string8 = "P1 (Z B %s %s)" % (PMOS1net2, PMOS1net3) + string7[(string7.find("(")+len("(")):]
2561     string9 = (NOR2_types[i].NOR2_object()).PMOS_1_lines()[1]
2562     string10 = (NOR2_types[i].NOR2_object()).PMOS_0_lines()[0]
2563     string11 = "P0 (%s A VDD %s)" % (PMOS0net0, PMOS0net3) + string10[(string10.find("(")+len("(")):]
2564     string12 = (NOR2_types[i].NOR2_object()).PMOS_0_lines()[1]
2565     string13 = "ends NOR2_type_%s \n// End of subcircuit definition. \n \n" % str(NOR2_types[i].NOR2_type)
2566     string = string0 + " " + string2 + string3 + " " + string5 + string6 + " " + string8 + string9 + " " + string11 + string12 + string13
2567     subcircuit_instance.append(string)
2568
2569     return subcircuit_instance
2570
2571
2572 def write_NAND2_to_netlist(input_subcircuit_instance):
2573     ##append the subcircuit instance of each type of NAND2 to the subcircuit instance list
2574
2575     ##Write new subcircuit components to netlist
2576     subcircuit_instance = input_subcircuit_instance
2577
2578     for i in range(len(NAND2_types)):
2579
2580         NMOS1net0 = str((NAND2_types[i].NAND2_object()).NMOS_1_net()[0])
2581
2582         if (NAND2_types[i].NAND2_object()).NMOS_1_net()[3] == '0':
2583             NMOS1net3 = "GND"
2584         else:
2585             NMOS1net3 = str((NAND2_types[i].NAND2_object()).NMOS_1_net()[3])
2586

```

```

2587     if (NAND2_types[i].NAND2_object()).NMOS_0_net()[2] ←
2588         == '0':
2589         NMOS0net2 = "GND"
2590     else:
2591         NMOS0net2 = str((NAND2_types[i].NAND2_object() ←
2592             ).NMOS_0_net()[2])
2593
2594     if (NAND2_types[i].NAND2_object()).NMOS_0_net()[3] ←
2595         == '0':
2596         NMOS0net3 = "GND"
2597     else:
2598         NMOS0net3 = str((NAND2_types[i].NAND2_object() ←
2599             ).NMOS_0_net()[3])
2600
2601     PMOS0net2 = str((NAND2_types[i].NAND2_object()). ←
2602         PMOS_0_net()[2])
2603
2604     if (NAND2_types[i].NAND2_object()).PMOS_0_net()[3] ←
2605         == PMOS0net2:
2606         PMOS0net3 = "VDD"
2607     else:
2608         PMOS0net3 = str((NAND2_types[i].NAND2_object() ←
2609             ).PMOS_0_net()[3])
2610
2611     if (NAND2_types[i].NAND2_object()).PMOS_1_net()[3] ←
2612         == PMOS0net2:
2613         PMOS1net3 = "VDD"
2614     else:
2615         PMOS1net3 = str((NAND2_types[i].NAND2_object() ←
2616             ).PMOS_1_net()[3])
2617
2618     string0 = "subckt NAND2_type_%s A B GND VDD Z \n" ←
2619         % str(NAND2_types[i].NAND2_type)
2620     string1 = (NAND2_types[i].NAND2_object()). ←
2621         NMOS_1_lines()[0]
2622     string2 = "N1 (%s B GND %s)" % (NMOS1net0, ←
2623         NMOS1net3) + string1[(string1.find("(")+len(") ←
2624         ")):]
2625     string3 = (NAND2_types[i].NAND2_object()). ←
2626         NMOS_1_lines()[1]
2627     string4 = (NAND2_types[i].NAND2_object()). ←
2628         NMOS_0_lines()[0]

```

```

2614     string5 = "N0 (Z A %s %s)" %(NMOS0net2, NMOS0net3)↵
        + string4[(string4.find("))+len(")")):]
2615     string6 = (NAND2_types[i].NAND2_object()).↵
        NMOS_0_lines()[1]
2616     string7 = (NAND2_types[i].NAND2_object()).↵
        PMOS_0_lines()[0]
2617     string8 = "P0 (Z A VDD %s)" % PMOS0net3 + string7↵
        [(string7.find("))+len(")")):]
2618     string9 = (NAND2_types[i].NAND2_object()).↵
        PMOS_0_lines()[1]
2619     string10 = (NAND2_types[i].NAND2_object()).↵
        PMOS_1_lines()[0]
2620     string11 = "P1 (Z B VDD %s)" % PMOS1net3 + ↵
        string10[(string10.find("))+len(")")):]
2621     string12 = (NAND2_types[i].NAND2_object()).↵
        PMOS_1_lines()[1]
2622     string13 = "ends NAND2_type_%s \n// End of ↵
        subcircuit definition. \n \n" % str(NAND2_types↵
        [i].NAND2_type)
2623     string = string0 + "      " + string2 + string3 + " ↵
        " + string5 + string6 + "      " + string8 + ↵
        string9 + "      " + string11 + string12 + ↵
        string13
2624     subcircuit_instance.append(string)
2625
2626     return subcircuit_instance
2627
2628
2629 def write_inverter_to_netlist(input_subcircuit_instance):
2630     ##append the subcircuit instance of each type of inverter ↵
        to the subcircuit instance list
2631
2632     ##Write new subcircuit components to netlist
2633
2634     subcircuit_instance = input_subcircuit_instance
2635
2636     for i in range(len(inverter_types)):
2637         string0 = "subckt inverter_type_%s GND In Out VDD ↵
            \n" % str(inverter_types[i].inverter_type)
2638         string1 = (inverter_types[i].inverter_object()).↵
            NMOS_lines()[0]
2639         string2 = "N0 (Out In GND GND)" + string1[(string1↵
            .find("))+len(")")):]

```

```

2640     string3 = (inverter_types[i].inverter_object()).↵
        NMOS_lines()[1]
2641     string4 = (inverter_types[i].inverter_object()).↵
        PMOS_lines()[0]
2642     string5 = "P0 (Out In VDD VDD)" + string4[(string4↵
        .find("))+len(")"):]
2643     string6 = (inverter_types[i].inverter_object()).↵
        PMOS_lines()[1]
2644     string7 = "ends inverter_type_%s \n // End of ↵
        subcircuit definition. \n \n" % str(↵
        inverter_types[i].inverter_type)
2645     string = string0 + "      " + string2 + string3 + " ↵
        " + string5 + string6 + string7
2646     subcircuit_instance.append(string)
2647
2648     return subcircuit_instance
2649
2650     #####MAIN PROGRAM↵
        #####
2651
2652     ##SCR('inverter_netlist.txt')
2653     ##SCR('inverter_chain_netlist.txt')
2654     ##SCR('inverter_chain_x4_x16_circuit_flat.txt')
2655     ##SCR('inverter_chain_circuit_and_tx_flat.txt')
2656     ##SCR('NAND2_circuit_flat.txt')
2657     ##SCR('NAND2_complex_circuit_flat.txt')
2658     ##SCR('NOR2_circuit_flat.txt')
2659     ##SCR('OR2_circuit_flat.txt')
2660     ##SCR('AND2_circuit_flat.txt')
2661     ##SCR('NAND2b0_circuit_flat.txt')
2662     ##SCR('OAI21_circuit_flat.txt')
2663     ##SCR('OAI21b1_circuit_flat.txt')
2664     ##SCR('OAI21b0b1_circuit_flat.txt')
2665     ##SCR('full_adder_circuit_flat.txt')
2666     ##SCR('full_adder_virtuoso.txt')
2667     ##SCR('XOR_flat_virtuoso.txt')
2668     ##SCR('master_slave_dff_flat_virtuoso.txt')
2669     ##SCR('2_to_1_MUX_flat_virtuoso.txt')
2670     ##SCR('digital_comparator_flat_virtuoso.txt')
2671     ##SCR('4_bit_full_adder_flat_virtuoso.txt')
2672     SCR('Test1_flat_virtuoso.txt')
2673     ##SCR('Test2_flat_virtuoso.txt')
2674     ##SCR('Test3_flat_virtuoso.txt')

```

```
2675 ##SCR('Test4_flat_virtuoso.txt')
```

Appendix D: Netlists

D.1 Test1 Output Netlist

```
// Generated for: spectre
// Generated on: Feb 17 11:35:03 2015
// Design library name: Thesis
// Design cell name: Test_1_20_gates_flat_virtuoso_2
// Design view name: schematic
simulator lang=spectre
global 0 vdd!

// Library name: Thesis
// Cell name: Test_1_20_gates_flat_virtuoso_2
// View name: schematic
subckt OAI21b0b1_type_0 GND In_AND_1 In_AND_2 In_INV VDD Z
    I0 (GND In_INV net1278 VDD) inverter_type_0
    I1 (In_AND_1 In_AND_2 GND VDD net1274) AND2_type_0
    I2 (net1278 net1274 GND VDD Z) OR2_type_0
ends OAI21b0b1_type_0
// End of subcircuit definition.

subckt OAI21b1_type_0 GND In_INV In_NAND In_OR VDD Z
    I1 (GND In net1297 VDD) inverter_type_0
    I2 (In_OR net1297 GND VDD net1310) OR2_type_0
    I3 (In_NAND net1310 GND VDD Z) NAND2_type_0
ends OAI21b1_type_0
```

```

// End of subcircuit definition.

subckt OAI21_type_0 GND In_NAND In_OR_1 IN_OR_2 VDD Z
    I0 (In_OR_1 In_OR_2 GND VDD net1319 OR2_type_0
    I1 (In_NAND net1319 GND VDD Z) NAND2_type_0
ends OAI21_type_0
// End of subcircuit definition.

subckt NAND2b0_type_0 A GND In VDD Z
    I0 (A net1261 GND VDD Z) OR2_type_0
    I1 (GND In net1261 VDD) inverter_type_0
ends NAND2b0_type_0
// End of subcircuit definition.

subckt OR2_type_0 A B GND VDD Z
    I0 (GND net1270 Z VDD) inverter_type_0
    I1 (A B GND VDD net1270) NOR2_type_0
ends OR2_type_0
// End of subcircuit definition.

subckt AND2_type_0 A B GND VDD Z
    I0 (GND net1197 Z VDD) inverter_type_0
    I1 (A B GND VDD net1197) NAND2_type_0
ends AND2_type_0
// End of subcircuit definition.

```

```

subckt NOR2_type_0 A B GND VDD Z

N1 (Z B GND GND) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u pd=15.0u \
    m=1 region=sat

N0 (Z A GND GND) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u pd=15.0u \
    m=1 region=sat

P1 (Z B net1317 net1317) ami06P w=24.0u l=600n as=3.6e-11 \
    ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat

P0 (net1317 A VDD VDD) ami06P w=24.0u l=600n as=3.6e-11 \
    ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat

ends NOR2_type_0

// End of subcircuit definition.

```

```

subckt NAND2_type_0 A B GND VDD Z

N1 (net1223 B GND net1223) ami06N w=48.0u l=600n as=7.2e-11 \
    ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat

N0 (Z A net1223 net1223) ami06N w=48.0u l=600n as=7.2e-11 \
    ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat

P0 (Z A VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 \
    ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat

P1 (Z B VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 \
    ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat

ends NAND2_type_0

// End of subcircuit definition.

```

```

subckt NAND2_type_1 A B GND VDD Z

N1 (net1245 B GND net1245) ami06N w=12.0u l=600n as=1.8e-11 \

```

```

        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
N0 (Z A net1245 net1245) ami06N w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P0 (Z A VDD VDD) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P1 (Z B VDD VDD) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
ends NAND2_type_1

// End of subcircuit definition.

subckt inverter_type_0 GND In Out VDD
    N0 (Out In GND GND) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
        pd=15.0u m=1 region=sat
    P0 (Out In VDD VDD) ami06P w=24.0u l=600n as=3.6e-11 \
        ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
ends inverter_type_0

// End of subcircuit definition.

subckt inverter_type_1 GND In Out VDD
    N0 (Out In GND GND) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
        pd=15.0u m=1 region=sat
    P0 (Out In VDD VDD) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
ends inverter_type_1

// End of subcircuit definition.

```

```

subckt inverter_type_2 GND In Out VDD

    N0 (Out In GND GND) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
        ps=99.0u pd=99.0u m=1 region=sat

    P0 (Out In VDD VDD) ami06P w=96.0u l=600n as=1.44e-10 \
        ad=1.44e-10 ps=195.0000000u pd=195.0000000u m=1 region=sat

ends inverter_type_2

// End of subcircuit definition.

```

```

I20 (0 C net1199 vdd!) inverter_type_0
I19 (0 net1213 net1228 vdd!) inverter_type_0
I18 (0 net0192 net0261 vdd!) inverter_type_2
I17 (0 net0325 net1275 vdd!) inverter_type_1
I16 (0 net1255 net1261 net1262) inverter_type_0
I15 (net0260 net0261 0 vdd! net0233) NAND2_type_0
I14 (net0230 net1261 0 vdd! net0195) NAND2_type_0
I13 (net1235 net1237 0 vdd! net1255) NAND2_type_1
I12 (net1214 net1218 0 vdd! net1235) NAND2_type_0
I11 (net0235 net0233 0 vdd! Z) NOR2_type_0
I10 (net0189 net0192 0 vdd! net0235) AND2_type_0
I9 (net1199 net1218 0 vdd! net1213) AND2_type_0
I8 (net1235 net1237 0 vdd! net1269) AND2_type_0
I7 (A B 0 vdd! net1214) AND2_type_0
I6 (net1269 net1255 0 vdd! net0325) OR2_type_0
I5 (net1237 0 net1218 net1220 net1249) NAND2b0_type_0
I4 (net1261 0 net1255 net1262 net0189) NAND2b0_type_0
I3 (0 A A C vdd! net1218) OAI21_type_0

```

```

I2 (0 net0195 net0192 net0195 vdd! net0260) OAI21_type_0
I1 (0 net0325 net0230 net0230 vdd! net0192) OAI21b1_type_0
I0 (0 net1269 net1249 net1255 vdd! net0230) OAI21b0b1_type_0
P22 (net1262 net1255 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
    ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
P25 (net1220 net1214 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
    ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
N25 (net1237 net1214 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
    pd=15.0u m=1 region=sat
simulatorOptions options reltol=1e-3 vabstol=1e-6 iabstol=1e-12 temp=27 \
    tnom=27 scalem=1.0 scale=1.0 gmin=1e-12 rforce=1 maxnotes=5 maxwarns=5 \
    digits=5 cols=80 pivrel=1e-3 sensfile="../psf/sens.output" \
    checklimitdest=psf
modelParameter info what=models where=rawfile
element info what=inst where=rawfile
outputParameter info what=output where=rawfile
designParamVals info what=parameters where=rawfile
primitives info what=primitives where=rawfile
subckts info what=subckts where=rawfile
saveOptions options save=allpub

```

D.2 Gate-Level Netlist for the Unmodified 4-bit Ripple Carry Adder

```
// Generated for: spectre
// Generated on: Dec 12 16:27:54 2014
// Design library name: Thesis
// Design cell name: 4_bit_full_adder_flat_virtuoso
// Design view name: schematic
simulator lang=spectre
global 0 vdd!

// Library name: Thesis
// Cell name: 4_bit_full_adder_flat_virtuoso
// View name: schematic
subckt OAI21b0b1_type_0 GND In_AND_1 In_AND_2 In_INV VDD Z
    I0 (GND In_INV net01536 VDD) inverter_type_0
    I1 (In_AND_1 In_AND_2 GND VDD net01532) AND2_type_0
    I2 (net01536 net01532 GND VDD Z) OR2_type_0
ends OAI21b0b1_type_0
// End of subcircuit definition.

subckt OAI21b1_type_0 GND In_INV In_NAND In_OR VDD Z
    I1 (GND In net01320 VDD) inverter_type_0
    I2 (In_OR net01320 GND VDD net01351) OR2_type_0
    I3 (In_NAND net01351 GND VDD Z) NAND2_type_0
ends OAI21b1_type_0
// End of subcircuit definition.
```

```

subckt OAI21_type_0 GND In_NAND In_OR_1 IN_OR_2 VDD Z
    I0 (In_OR_1 In_OR_2 GND VDD net01465 OR2_type_0
    I1 (In_NAND net01465 GND VDD Z) NAND2_type_4
ends OAI21_type_0
// End of subcircuit definition.

```

```

subckt NAND2b0_type_0 A GND In VDD Z
    I0 (A net01337 GND VDD Z) OR2_type_0
    I1 (GND In net01337 VDD) inverter_type_0
ends NAND2b0_type_0
// End of subcircuit definition.

```

```

subckt OR2_type_0 A B GND VDD Z
    I0 (GND net01537 Z VDD) inverter_type_0
    I1 (A B GND VDD net01537) NOR2_type_0
ends OR2_type_0
// End of subcircuit definition.

```

```

subckt AND2_type_0 A B GND VDD Z
    I0 (GND net01508 Z VDD) inverter_type_0
    I1 (A B GND VDD net01508) NAND2_type_0
ends AND2_type_0
// End of subcircuit definition.

```

```

subckt NOR2_type_0 A B GND VDD Z
    N1 (Z B GND GND) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \

```

```

        pd=15.0u m=1 region=sat
N0 (Z A GND GND) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
        pd=15.0u m=1 region=sat
P1 (Z B net01538 net01538) ami06P w=24.0u l=600n as=3.6e-11 \
        ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
P0 (net01538 A VDD VDD) ami06P w=24.0u l=600n as=3.6e-11 \
        ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
ends NOR2_type_0
// End of subcircuit definition.

subckt NAND2_type_0 A B GND VDD Z
    N1 (net01440 B GND net01440) ami06N w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
        ps=147.000000u pd=147.000000u m=1 region=sat
    N0 (Z A net01440 net01440) ami06N w=72.0u l=600n as=1.08e-10 \
        ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
    P0 (Z A VDD VDD) ami06P w=72.0u l=600n as=1.08e-10 \
        ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
    P1 (Z B VDD VDD) ami06P w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
        ps=147.000000u pd=147.000000u m=1 region=sat
ends NAND2_type_0
// End of subcircuit definition.

subckt NAND2_type_1 A B GND VDD Z
    N1 (net01480 B GND net01480) ami06N w=48.0u l=600n as=7.2e-11 \
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
    N0 (Z A net01480 net01480) ami06N w=48.0u l=600n as=7.2e-11 \

```

```

        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
P0 (Z A VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 \
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
P1 (Z B VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 \
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
ends NAND2_type_1
// End of subcircuit definition.

subckt NAND2_type_2 A B GND VDD Z
    N1 (net01302 B GND net01302) ami06N w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
        ps=147.000000u pd=147.000000u m=1 region=sat
    N0 (Z A net01302 net01302) ami06N w=72.0u l=600n \
        as=1.08e-10 ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
    P0 (Z A VDD VDD) ami06P w=72.0u l=600n as=1.08e-10 \
        ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
    P1 (Z B VDD VDD) ami06P w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
        ps=147.000000u pd=147.000000u m=1 region=sat
ends NAND2_type_2
// End of subcircuit definition.

subckt NAND2_type_3 A B GND VDD Z
    N1 (net01335 B GND net01335) ami06N w=48.0u l=600n as=7.2e-11 \
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
    N0 (Z A net01335 net01335) ami06N w=48.0u l=600n \
        as=7.2e-11 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
    P0 (Z A VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 \

```

```

        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
P1 (Z B VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 \
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
ends NAND2_type_3
// End of subcircuit definition.

subckt NAND2_type_4 A B GND VDD Z
    N1 (net01469 B GND net01469) ami06N w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
    N0 (Z A net01469 net01469) ami06N w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
    P0 (Z A VDD VDD) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
    P1 (Z B VDD VDD) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
ends NAND2_type_4
// End of subcircuit definition.

subckt inverter_type_0 GND In Out VDD
    N0 (Out In GND GND) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 ps=99.0u \
        pd=99.0u m=1 region=sat
    P0 (Out In VDD VDD) ami06P w=96.0u l=600n as=1.44e-10 ad=1.44e-10 \
        ps=195.000000u pd=195.000000u m=1 region=sat
ends inverter_type_0
// End of subcircuit definition.

```

```

subckt inverter_type_1 GND In Out VDD
    N0 (Out In GND GND) ami06N w=24.0u l=600n as=3.6e-11 ad=3.6e-11 ps=51.0u \
        pd=51.0u m=1 region=sat
    P0 (Out In VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
        ps=99.0u pd=99.0u m=1 region=sat
ends inverter_type_1

// End of subcircuit definition.

subckt inverter_type_2 GND In Out VDD
    N0 (Out In GND GND) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
        pd=15.0u m=1 region=sat
    P0 (Out In VDD VDD) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
ends inverter_type_2

// End of subcircuit definition.

I35 (0 B0 net01418 vdd!) inverter_type_0
I34 (0 A0 net01417 vdd!) inverter_type_0
I33 (0 B1 net01280 vdd!) inverter_type_0
I32 (0 A1 net01279 vdd!) inverter_type_0
I31 (0 B3 net01283 vdd!) inverter_type_0
I30 (0 A3 net01282 vdd!) inverter_type_0
I29 (0 B2 net01421 vdd!) inverter_type_1
I28 (0 A2 net01420 vdd!) inverter_type_0
I27 (net01418 A0 0 vdd! net01455) NAND2_type_0
I26 (net01455 net01459 0 vdd! net01588) NAND2_type_0

```

I25 (net01417 B0 0 vdd! net01459) NAND2_type_0
 I24 (net01280 A1 0 vdd! net01317) NAND2_type_0
 I23 (net01317 net01321 0 vdd! net01546) NAND2_type_3
 I22 (net01279 B1 0 vdd! net01321) NAND2_type_0
 I21 (net01283 A3 0 vdd! net01319) NAND2_type_0
 I20 (net01319 net01325 0 vdd! net01547) NAND2_type_0
 I19 (net01282 B3 0 vdd! net01325) NAND2_type_2
 I18 (net01421 A2 0 vdd! net01457) NAND2_type_0
 I17 (net01457 net01463 0 vdd! net01589) NAND2_type_1
 I16 (net01420 B2 0 vdd! net01463) NAND2_type_0
 I15 (net01588 0 net01405 vdd! net01528) NAND2b0_type_0
 I14 (net01547 0 Cin0 vdd! net01393) NAND2b0_type_0
 I13 (net01589 0 net01406 vdd! net01531) NAND2b0_type_0
 I12 (net01546 0 net01544 vdd! net01390) NAND2b0_type_0
 I11 (0 net01544 B1 A1 vdd! net01332) OAI21_type_0
 I10 (0 Cin0 B3 A3 vdd! net01339) OAI21_type_0
 I9 (0 net01406 B2 A2 vdd! net01477) OAI21_type_0
 I8 (0 net01405 B0 A0 vdd! net01470) OAI21_type_0
 I7 (0 net01588 net01528 net01405 vdd! S) OAI21b1_type_0
 I6 (0 net01547 net01393 Cin0 vdd! S3) OAI21b1_type_0
 I5 (0 net01589 net01531 net01406 vdd! S2) OAI21b1_type_0
 I4 (0 net01546 net01390 net01544 vdd! S1) OAI21b1_type_0
 I3 (0 B0 A0 net01470 vdd! Cout0) OAI21b0b1_type_0
 I2 (0 B1 A1 net01332 vdd! net01405) OAI21b0b1_type_0
 I1 (0 B3 A3 net01339 vdd! net01406) OAI21b0b1_type_0
 I0 (0 B2 A2 net01477 vdd! net01544) OAI21b0b1_type_0

```
simulatorOptions options reltol=1e-3 vabstol=1e-6 iabstol=1e-12 temp=27 \  
    tnom=27 scalem=1.0 scale=1.0 gmin=1e-12 rforce=1 maxnotes=5 maxwarns=5 \  
    digits=5 cols=80 pivrel=1e-3 sensfile="../psf/sens.output" \  
    checklimitdest=psf  
modelParameter info what=models where=rawfile  
element info what=inst where=rawfile  
outputParameter info what=output where=rawfile  
designParamVals info what=parameters where=rawfile  
primitives info what=primitives where=rawfile  
subckts info what=subckts where=rawfile  
saveOptions options save=allpub
```

D.3 Transistor-Level Netlist for the Modified (OAI21 Gate Composition) 4-bit Ripple Carry Adder

```
// Cell name: 4_bit_altered_full_adder_flat_virtuoso
// View name: schematic

P110 (net01588 net01455 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
      ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
P97 (net01459 net01417 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 \
      ad=1.08e-10 ps=147.0000000u pd=147.0000000u m=1 region=sat
P24 (net01539 net01532 net01540 net01540) ami06P w=24.0u l=600n as=3.6e-11 \
      ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
P26 (net01540 net01536 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
      ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
P85 (net01333 net01544 net01334 net01334) ami06P w=24.0u l=600n as=3.6e-11 \
      ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
P92 (net01334 net01320 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
      ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
P105 (net01373 net01337 net01374 net01374) ami06P w=24.0u l=600n \
      as=3.6e-11 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
P114 (net01374 net01546 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
      ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
P18 (net01438 A2 net01439 net01439) ami06P w=24.0u l=600n as=3.6e-11 \
      ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
P20 (net01439 B2 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 ad=3.6e-11 \
      ps=51.0u pd=51.0u m=1 region=sat
P13 (net01478 net01406 net01479 net01479) ami06P w=24.0u l=600n as=3.6e-11 \
      ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
```

P15 (net01479 net01462 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P8 (net01518 net01482 net01519 net01519) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P6 (net01519 net01589 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P33 (net01401 net01394 net01402 net01402) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P37 (net01402 net01398 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P34 (net01300 A3 net01301 net01301) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P38 (net01301 B3 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 ad=3.6e-11 \
 ps=51.0u pd=51.0u m=1 region=sat
 P42 (net01340 Cin0 net01341 net01341) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P44 (net01341 net01324 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P52 (net01380 net01344 net01381 net01381) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P54 (net01381 net01547 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P69 (net01293 A1 net01294 net01294) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P80 (net01294 B1 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 ad=3.6e-11 \
 ps=51.0u pd=51.0u m=1 region=sat

P68 (net01399 net01392 net01400 net01400) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P79 (net01400 net01396 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P84 (net01471 net01405 net01472 net01472) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P88 (net01472 net01458 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P104 (net01511 net01475 net01512 net01512) ami06P w=24.0u l=600n \
 as=3.6e-11 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P108 (net01512 net01588 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P121 (net0349 B0 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P29 (net01532 net01515 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P25 (net01544 net01539 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P23 (net01536 net01477 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P87 (net01351 net01333 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P95 (net01320 net01546 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P102 (net01337 net01544 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat

P107 (net01390 net01373 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P93 (S1 net01390 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P94 (S1 net01351 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P28 (net01515 A2 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P27 (net01515 B2 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P21 (net01477 net01467 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P19 (net01467 net01438 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P22 (net01477 net01406 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P14 (net01490 net01478 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P12 (net01462 net01589 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P11 (net01482 net01406 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P10 (net01463 B2 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
 ps=147.000000u pd=147.000000u m=1 region=sat
 P9 (net01463 net01420 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 \
 ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat

P7 (net01420 A2 vdd! vdd!) ami06P w=96.0u l=600n as=1.44e-10 ad=1.44e-10 \
 ps=195.000000u pd=195.000000u m=1 region=sat
 P4 (net01589 net01463 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P1 (net01589 net01457 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P3 (net01457 A2 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 P0 (net01457 net01421 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P2 (net01421 B2 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 P5 (net01531 net01518 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P17 (S2 net01531 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P16 (S2 net01490 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P35 (net01406 net01401 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P30 (net01394 net01377 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P41 (net01398 net01339 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P31 (net01377 A3 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat

P32 (net01377 B3 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P36 (net01329 net01300 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P40 (net01339 net01329 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P39 (net01339 Cin0 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P43 (net01352 net01340 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P47 (net01324 net01547 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P48 (net01325 B3 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
 ps=147.000000u pd=147.000000u m=1 region=sat
 P49 (net01325 net01282 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 \
 ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
 P51 (net01282 A3 vdd! vdd!) ami06P w=96.0u l=600n as=1.44e-10 ad=1.44e-10 \
 ps=195.000000u pd=195.000000u m=1 region=sat
 P55 (net01547 net01325 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P56 (net01547 net01319 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P58 (net01319 A3 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 P57 (net01319 net01283 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat

P59 (net01283 B3 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 P50 (net01344 Cin0 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P53 (net01393 net01380 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P45 (S3 net01393 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P46 (S3 net01352 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P81 (net01332 net01544 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P82 (net01332 net01327 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P73 (net01327 net01293 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P64 (net01370 A1 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P65 (net01370 B1 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P63 (net01392 net01370 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P83 (net01396 net01332 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P72 (net01405 net01399 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat

P86 (net01489 net01471 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P91 (net01458 net01588 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P100 (net01321 B1 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
 ps=147.000000u pd=147.000000u m=1 region=sat
 P101 (net01321 net01279 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 \
 ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
 P103 (net01279 A1 vdd! vdd!) ami06P w=96.0u l=600n as=1.44e-10 ad=1.44e-10 \
 ps=195.000000u pd=195.000000u m=1 region=sat
 P115 (net01546 net01321 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P116 (net01546 net01317 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P118 (net01317 A1 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 P117 (net01317 net01280 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P119 (net01280 B1 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 P98 (net01475 net01405 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P106 (net01528 net01511 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P89 (S net01528 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat

P90 (S net01489 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P109 (net01588 net01459 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P96 (net01459 B0 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
 ps=147.000000u pd=147.000000u m=1 region=sat
 P122 (net0384 net0349 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P125 (net01470 net01405 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P66 (net01537 net01530 net01538 net01538) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P74 (net01538 net01534 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P70 (Cout0 net01537 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P99 (net01417 A0 vdd! vdd!) ami06P w=96.0u l=600n as=1.44e-10 ad=1.44e-10 \
 ps=195.000000u pd=195.000000u m=1 region=sat
 P61 (net01508 A0 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P62 (net01508 B0 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P60 (net01530 net01508 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P120 (net0334 A0 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat

P78 (net01534 net01470 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P123 (net0384 net0334 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P124 (net0398 net0384 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P126 (net01470 net0398 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P113 (net01418 B0 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 P111 (net01455 net01418 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P112 (net01455 A0 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 N106 (net01473 net01459 0 net01473) ami06N w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N114 (net01588 net01455 net01473 net01473) ami06N w=48.0u l=600n \
 as=7.2e-11 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N96 (net01433 B0 0 net01433) ami06N w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
 ps=147.0000000u pd=147.0000000u m=1 region=sat
 N98 (net01459 net01417 net01433 net01433) ami06N w=72.0u l=600n \
 as=1.08e-10 ad=1.08e-10 ps=147.0000000u pd=147.0000000u m=1 region=sat \
 N89 (net01371 net01390 0 net01371) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N94 (S1 net01351 net01371 net01371) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat

N28 (net01514 A2 0 net01514) ami06N w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 N27 (net01515 B2 net01514 net01514) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N21 (net01476 net01467 0 net01476) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N22 (net01477 net01406 net01476 net01476) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N10 (net01440 B2 0 net01440) ami06N w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
 ps=147.000000u pd=147.000000u m=1 region=sat
 N9 (net01463 net01420 net01440 net01440) ami06N w=72.0u l=600n as=1.08e-10 \
 ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
 N4 (net01480 net01463 0 net01480) ami06N w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N1 (net01589 net01457 net01480 net01480) ami06N w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N3 (net01442 A2 0 net01442) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 N0 (net01457 net01421 net01442 net01442) ami06N w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N16 (net01516 net01531 0 net01516) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N17 (S2 net01490 net01516 net01516) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N30 (net01376 A3 0 net01376) ami06N w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat

N32 (net01377 B3 net01376 net01376) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N37 (net01338 net01329 0 net01338) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N40 (net01339 Cin0 net01338 net01338) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N48 (net01302 B3 0 net01302) ami06N w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
 ps=147.000000u pd=147.000000u m=1 region=sat
 N49 (net01325 net01282 net01302 net01302) ami06N w=72.0u l=600n \
 as=1.08e-10 ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat \
 N54 (net01342 net01325 0 net01342) ami06N w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N57 (net01547 net01319 net01342 net01342) ami06N w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N55 (net01304 A3 0 net01304) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 N58 (net01319 net01283 net01304 net01304) ami06N w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N44 (net01378 net01393 0 net01378) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N46 (S3 net01352 net01378 net01378) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N82 (net01332 net01544 net01331 net01331) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N75 (net01331 net01327 0 net01331) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat

N61 (net01369 A1 0 net01369) ami06N w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 N65 (net01370 B1 net01369 net01369) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N97 (net01295 B1 0 net01295) ami06N w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
 ps=147.000000u pd=147.000000u m=1 region=sat
 N101 (net01321 net01279 net01295 net01295) ami06N w=72.0u l=600n \
 as=1.08e-10 ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat \
 N110 (net01335 net01321 0 net01335) ami06N w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N117 (net01546 net01317 net01335 net01335) ami06N w=48.0u l=600n \
 as=7.2e-11 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N111 (net01297 A1 0 net01297) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 N118 (net01317 net01280 net01297 net01297) ami06N w=48.0u l=600n \
 as=7.2e-11 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N86 (net01509 net01528 0 net01509) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N92 (S net01489 net01509 net01509) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N121 (net0349 B0 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N29 (net01532 net01515 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N24 (net01539 net01532 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat

N25 (net01539 net01536 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N26 (net01544 net01539 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N23 (net01536 net01477 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N87 (net01333 net01544 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N88 (net01333 net01320 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N91 (net01351 net01333 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N95 (net01320 net01546 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N102 (net01337 net01544 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N108 (net01373 net01337 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N109 (net01373 net01546 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N113 (net01390 net01373 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N19 (net01438 A2 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N18 (net01438 B2 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat

N20 (net01467 net01438 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N14 (net01478 net01406 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N15 (net01490 net01478 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N13 (net01478 net01462 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N12 (net01462 net01589 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N11 (net01482 net01406 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N8 (net01420 A2 0 0) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 ps=99.0u \
 pd=99.0u m=1 region=sat
 N2 (net01421 B2 0 0) ami06N w=24.0u l=600n as=3.6e-11 ad=3.6e-11 ps=51.0u \
 pd=51.0u m=1 region=sat
 N7 (net01518 net01482 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N6 (net01518 net01589 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N5 (net01531 net01518 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N38 (net01406 net01401 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N33 (net01401 net01398 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat

N34 (net01401 net01394 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N31 (net01394 net01377 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N41 (net01398 net01339 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N35 (net01300 A3 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N36 (net01300 B3 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N39 (net01329 net01300 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N42 (net01340 Cin0 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N43 (net01340 net01324 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N45 (net01352 net01340 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N47 (net01324 net01547 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N51 (net01282 A3 0 0) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 ps=99.0u \
 pd=99.0u m=1 region=sat
 N59 (net01283 B3 0 0) ami06N w=24.0u l=600n as=3.6e-11 ad=3.6e-11 ps=51.0u \
 pd=51.0u m=1 region=sat
 N50 (net01344 Cin0 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat

N52 (net01380 net01344 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N53 (net01380 net01547 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N56 (net01393 net01380 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N79 (net01327 net01293 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N73 (net01293 A1 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N74 (net01293 B1 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N64 (net01392 net01370 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N72 (net01399 net01392 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N71 (net01399 net01396 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N83 (net01396 net01332 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N78 (net01405 net01399 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N84 (net01471 net01405 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N85 (net01471 net01458 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat

N90 (net01489 net01471 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N93 (net01458 net01588 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N103 (net01279 A1 0 0) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 N119 (net01280 B1 0 0) ami06N w=24.0u l=600n as=3.6e-11 ad=3.6e-11 \
 ps=51.0u pd=51.0u m=1 region=sat
 N99 (net01475 net01405 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N104 (net01511 net01475 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N105 (net01511 net01588 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N112 (net01528 net01511 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N123 (net0383 net0334 0 net0383) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N122 (net0384 net0349 net0383 net0383) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N126 (net0453 net0398 0 net0453) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N125 (net01470 net01405 net0453 net0453) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N60 (net01507 A0 0 net01507) ami06N w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat

N63 (net01508 B0 net01507 net01507) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N76 (Cout0 net01537 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N100 (net01417 A0 0 0) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 N107 (net01435 A0 0 net01435) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 N115 (net01455 net01418 net01435 net01435) ami06N w=48.0u l=600n \
 as=7.2e-11 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N62 (net01530 net01508 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N67 (net01537 net01530 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N66 (net01537 net01534 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N120 (net0334 A0 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N81 (net01534 net01470 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N124 (net0398 net0384 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N116 (net01418 B0 0 0) ami06N w=24.0u l=600n as=3.6e-11 ad=3.6e-11 \
 ps=51.0u pd=51.0u m=1 region=sat
 simulatorOptions options reltol=1e-3 vabstol=1e-6 iabstol=1e-12 temp=27 \
 tnom=27 scalem=1.0 scale=1.0 gmin=1e-12 rforce=1 maxnotes=5 maxwarns=5 \

```
digits=5 cols=80 pivrel=1e-3 sensfile="../psf/sens.output" \  
checklimitdest=psf  
modelParameter info what=models where=rawfile  
element info what=inst where=rawfile  
outputParameter info what=output where=rawfile  
designParamVals info what=parameters where=rawfile  
primitives info what=primitives where=rawfile  
subckts info what=subckts where=rawfile  
saveOptions options save=allpub
```

D.4 Gate-Level Netlist for the Modified (OAI21 Gate Composition) 4-bit Ripple

Carry Adder

```
// Cell name: 4_bit_altered_full_adder_flat_virtuoso
// View name: schematic
subckt OAI21b0b1_type_0 GND In_AND_1 In_AND_2 In_INV VDD Z
    I0 (GND In_INV net01536 VDD) inverter_type_0
    I1 (In_AND_1 In_AND_2 GND VDD net01532) AND2_type_0
    I2 (net01536 net01532 GND VDD Z) OR2_type_0
ends OAI21b0b1_type_0
// End of subcircuit definition.
```

```
subckt OAI21b1_type_0 GND In_INV In_NAND In_OR VDD Z
    I1 (GND In net01320 VDD) inverter_type_0
    I2 (In_OR net01320 GND VDD net01351) OR2_type_0
    I3 (In_NAND net01351 GND VDD Z) NAND2_type_0
ends OAI21b1_type_0
// End of subcircuit definition.
```

```
subckt OAI21_type_0 GND In_NAND In_OR_1 IN_OR_2 VDD Z
    I0 (In_OR_1 In_OR_2 GND VDD net01467 OR2_type_0
    I1 (In_NAND net01467 GND VDD Z) NAND2_type_0
ends OAI21_type_0
// End of subcircuit definition.
```

```
subckt NAND2b0_type_0 A GND In VDD Z
    I0 (A net01337 GND VDD Z) OR2_type_0
```

```

        I1 (GND In net01337 VDD) inverter_type_0
ends NAND2b0_type_0
// End of subcircuit definition.

subckt OR2_type_0 A B GND VDD Z
    I0 (GND net01537 Z VDD) inverter_type_0
    I1 (A B GND VDD net01537) NOR2_type_0
ends OR2_type_0
// End of subcircuit definition.

subckt AND2_type_0 A B GND VDD Z
    I0 (GND net0384 Z VDD) inverter_type_0
    I1 (A B GND VDD net0384) NAND2_type_0
ends AND2_type_0
// End of subcircuit definition.

subckt NOR2_type_0 A B GND VDD Z
    N1 (Z B GND GND) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
        pd=15.0u m=1 region=sat
    N0 (Z A GND GND) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
        pd=15.0u m=1 region=sat
    P1 (Z B net01538 net01538) ami06P w=24.0u l=600n as=3.6e-11 \
        ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
    P0 (net01538 A VDD VDD) ami06P w=24.0u l=600n as=3.6e-11 \
        ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
ends NOR2_type_0

```

```
// End of subcircuit definition.
```

```
subckt NAND2_type_0 A B GND VDD Z
```

```
    N1 (net01473 B GND net01473) ami06N w=48.0u l=600n as=7.2e-11 \  
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
```

```
    N0 (Z A net01473 net01473) ami06N w=48.0u l=600n \  
        as=7.2e-11 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
```

```
    P0 (Z A VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 \  
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
```

```
    P1 (Z B VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 \  
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
```

```
ends NAND2_type_0
```

```
// End of subcircuit definition.
```

```
subckt NAND2_type_1 A B GND VDD Z
```

```
    N1 (net01433 B GND net01433) ami06N w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \  
        ps=147.000000u pd=147.000000u m=1 region=sat
```

```
    N0 (Z A net01433 net01433) ami06N w=72.0u l=600n \  
        as=1.08e-10 ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat \  
        \
```

```
    P0 (Z A VDD VDD) ami06P w=72.0u l=600n as=1.08e-10 \  
        ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
```

```
    P1 (Z B VDD VDD) ami06P w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \  
        ps=147.000000u pd=147.000000u m=1 region=sat
```

```
ends NAND2_type_1
```

```
// End of subcircuit definition.
```

```

subckt NAND2_type_2 A B GND VDD Z

    N1 (net01480 B GND net01480) ami06N w=48.0u l=600n as=7.2e-11 \
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
    N0 (Z A net01480 net01480) ami06N w=48.0u l=600n as=7.2e-11 \
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
    P0 (Z A VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 \
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
    P1 (Z B VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 \
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
ends NAND2_type_2

// End of subcircuit definition.

subckt NAND2_type_3 A B GND VDD Z

    N1 (net0453 B GND net0453) ami06N w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
    N0 (Z A net0453 net0453) ami06N w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
    P0 (Z A VDD VDD) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
    P1 (Z B VDD VDD) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
ends NAND2_type_3

// End of subcircuit definition.

subckt inverter_type_0 GND In Out VDD

    N0 (Out In GND GND) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \

```

```

        pd=15.0u m=1 region=sat
P0 (Out In VDD VDD) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
        ps=27.0u pd=27.0u m=1 region=sat
ends inverter_type_0

// End of subcircuit definition.

subckt inverter_type_1 GND In Out VDD
    N0 (Out In GND GND) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 ps=99.0u \
        pd=99.0u m=1 region=sat
    P0 (Out In VDD VDD) ami06P w=96.0u l=600n as=1.44e-10 ad=1.44e-10 \
        ps=195.000000u pd=195.000000u m=1 region=sat
ends inverter_type_1

// End of subcircuit definition.

subckt inverter_type_2 GND In Out VDD
    N0 (Out In GND GND) ami06N w=24.0u l=600n as=3.6e-11 ad=3.6e-11 ps=51.0u \
        pd=51.0u m=1 region=sat
    P0 (Out In VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
        ps=99.0u pd=99.0u m=1 region=sat
ends inverter_type_2

// End of subcircuit definition.

I38 (0 B0 net01418 vdd!) inverter_type_0
I37 (0 A0 net0334 vdd!) inverter_type_0
I36 (0 A0 net01417 vdd!) inverter_type_0
I35 (0 B1 net01280 vdd!) inverter_type_0

```

I34 (0 A1 net01279 vdd!) inverter_type_0
 I33 (0 B3 net01283 vdd!) inverter_type_0
 I32 (0 A3 net01282 vdd!) inverter_type_0
 I31 (0 B2 net01421 vdd!) inverter_type_2
 I30 (0 A2 net01420 vdd!) inverter_type_1
 I29 (0 B0 net0349 vdd!) inverter_type_0
 I28 (net01405 net0398 0 vdd! net01470) NAND2_type_3
 I27 (net01418 A0 0 vdd! net01455) NAND2_type_0
 I26 (net01280 A1 0 vdd! net01317) NAND2_type_0
 I25 (net01317 net01321 0 vdd! net01546) NAND2_type_0
 I24 (net01279 B1 0 vdd! net01321) NAND2_type_0
 I23 (net01283 A3 0 vdd! net01319) NAND2_type_0
 I22 (net01319 net01325 0 vdd! net01547) NAND2_type_0
 I21 (net01282 B3 0 vdd! net01325) NAND2_type_0
 I20 (net01421 A2 0 vdd! net01457) NAND2_type_0
 I19 (net01457 net01463 0 vdd! net01589) NAND2_type_2
 I18 (net01420 B2 0 vdd! net01463) NAND2_type_0
 I17 (net01417 B0 0 vdd! net01459) NAND2_type_1
 I16 (net01455 net01459 0 vdd! net01588) NAND2_type_0
 I15 (net0349 net0334 0 vdd! net0398) AND2_type_0
 I14 (net01588 0 net01405 vdd! net01528) NAND2b0_type_0
 I13 (net01547 0 Cin0 vdd! net01393) NAND2b0_type_0
 I12 (net01589 0 net01406 vdd! net01531) NAND2b0_type_0
 I11 (net01546 0 net01544 vdd! net01390) NAND2b0_type_0
 I10 (0 net01544 B1 A1 vdd! net01332) OAI21_type_0
 I9 (0 Cin0 B3 A3 vdd! net01339) OAI21_type_0

```

I8 (0 net01406 B2 A2 vdd! net01477) OAI21_type_0
I7 (0 net01588 net01528 net01405 vdd! S) OAI21b1_type_0
I6 (0 net01547 net01393 Cin0 vdd! S3) OAI21b1_type_0
I5 (0 net01589 net01531 net01406 vdd! S2) OAI21b1_type_0
I4 (0 net01546 net01390 net01544 vdd! S1) OAI21b1_type_0
I3 (0 B0 A0 net01470 vdd! Cout0) OAI21b0b1_type_0
I2 (0 B1 A1 net01332 vdd! net01405) OAI21b0b1_type_0
I1 (0 B3 A3 net01339 vdd! net01406) OAI21b0b1_type_0
I0 (0 B2 A2 net01477 vdd! net01544) OAI21b0b1_type_0

simulatorOptions options reltol=1e-3 vabstol=1e-6 iabstol=1e-12 temp=27 \
    tnom=27 scalem=1.0 scale=1.0 gmin=1e-12 rforce=1 maxnotes=5 maxwarns=5 \
    digits=5 cols=80 pivrel=1e-3 sensfile="../psf/sens.output" \
    checklimitdest=psf

modelParameter info what=models where=rawfile
element info what=inst where=rawfile
outputParameter info what=output where=rawfile
designParamVals info what=parameters where=rawfile
primitives info what=primitives where=rawfile
subckts info what=subckts where=rawfile
saveOptions options save=allpub

```

D.5 Transistor-Level Netlist for the Modified (OAI21 Gate Input Switch) 4-bit Ripple Carry Adder

```
// Generated for: spectre
// Generated on: Mar  3 21:00:44 2015
// Design library name: Thesis
// Design cell name: 4_bit_inputswitch_full_adder_flat_virtuoso
// Design view name: schematic
simulator lang=spectre
global 0 vdd!

// Library name: Thesis
// Cell name: 4_bit_inputswitch_full_adder_flat_virtuoso
// View name: schematic
P75 (net01432 A0 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 ad=3.6e-11 \
    ps=51.0u pd=51.0u m=1 region=sat
P67 (net01431 B0 net01432 net01432) ami06P w=24.0u l=600n as=3.6e-11 \
    ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
P24 (net01539 net01532 net01540 net01540) ami06P w=24.0u l=600n as=3.6e-11 \
    ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
P26 (net01540 net01536 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
    ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
P85 (net01333 net01544 net01334 net01334) ami06P w=24.0u l=600n as=3.6e-11 \
    ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
P92 (net01334 net01320 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
    ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
P105 (net01373 net01337 net01374 net01374) ami06P w=24.0u l=600n \
```

as=3.6e-11 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P114 (net01374 net01546 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P18 (net01438 A2 net01439 net01439) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P20 (net01439 B2 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 ad=3.6e-11 \
 ps=51.0u pd=51.0u m=1 region=sat
 P13 (net01478 net01406 net01479 net01479) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P15 (net01479 net01462 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P8 (net01518 net01482 net01519 net01519) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P6 (net01519 net01589 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P33 (net01401 net01394 net01402 net01402) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P37 (net01402 net01398 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P34 (net01300 A3 net01301 net01301) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P38 (net01301 B3 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 ad=3.6e-11 \
 ps=51.0u pd=51.0u m=1 region=sat
 P42 (net01340 Cin0 net01341 net01341) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P44 (net01341 net01324 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \

ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P52 (net01380 net01344 net01381 net01381) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P54 (net01381 net01547 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P69 (net01293 A1 net01294 net01294) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P80 (net01294 B1 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 ad=3.6e-11 \
 ps=51.0u pd=51.0u m=1 region=sat
 P68 (net01399 net01392 net01400 net01400) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P79 (net01400 net01396 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P84 (net01471 net01405 net01472 net01472) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P88 (net01472 net01458 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P104 (net01511 net01475 net01512 net01512) ami06P w=24.0u l=600n \
 as=3.6e-11 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P108 (net01512 net01588 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P29 (net01532 net01515 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P25 (net01544 net01539 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P23 (net01536 net01477 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \

ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P87 (net01351 net01333 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P95 (net01320 net01546 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P102 (net01337 net01544 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P107 (net01390 net01373 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P93 (S1 net01390 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P94 (S1 net01351 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P28 (net01515 A2 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P27 (net01515 B2 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P21 (net01477 net01467 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P19 (net01467 net01438 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P22 (net01477 net01406 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P14 (net01490 net01478 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P12 (net01462 net01589 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \

```

        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P11 (net01482 net01406 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P10 (net01463 B2 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
        ps=147.000000u pd=147.000000u m=1 region=sat
P9 (net01463 net01420 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 \
        ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
P7 (net01420 A2 vdd! vdd!) ami06P w=96.0u l=600n as=1.44e-10 ad=1.44e-10 \
        ps=195.000000u pd=195.000000u m=1 region=sat
P4 (net01589 net01463 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
P1 (net01589 net01457 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
P3 (net01457 A2 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
        ps=99.0u pd=99.0u m=1 region=sat
P0 (net01457 net01421 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
P2 (net01421 B2 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
        ps=99.0u pd=99.0u m=1 region=sat
P5 (net01531 net01518 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P17 (S2 net01531 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
        ps=27.0u pd=27.0u m=1 region=sat
P16 (S2 net01490 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
        ps=27.0u pd=27.0u m=1 region=sat
P35 (net01406 net01401 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \

```

```

        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P30 (net01394 net01377 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P41 (net01398 net01339 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P31 (net01377 A3 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
        ps=27.0u pd=27.0u m=1 region=sat
P32 (net01377 B3 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
        ps=27.0u pd=27.0u m=1 region=sat
P36 (net01329 net01300 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P40 (net01339 net01329 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P39 (net01339 Cin0 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
        ps=27.0u pd=27.0u m=1 region=sat
P43 (net01352 net01340 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P47 (net01324 net01547 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P48 (net01325 B3 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
        ps=147.000000u pd=147.000000u m=1 region=sat
P49 (net01325 net01282 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 \
        ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
P51 (net01282 A3 vdd! vdd!) ami06P w=96.0u l=600n as=1.44e-10 ad=1.44e-10 \
        ps=195.000000u pd=195.000000u m=1 region=sat
P55 (net01547 net01325 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \

```

ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P56 (net01547 net01319 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P58 (net01319 A3 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 P57 (net01319 net01283 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P59 (net01283 B3 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 P50 (net01344 Cin0 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P53 (net01393 net01380 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P45 (S3 net01393 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P46 (S3 net01352 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P81 (net01332 net01544 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P82 (net01332 net01327 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P73 (net01327 net01293 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P64 (net01370 A1 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P65 (net01370 B1 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \

```

ps=27.0u pd=27.0u m=1 region=sat
P63 (net01392 net01370 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
    ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P83 (net01396 net01332 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
    ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P72 (net01405 net01399 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
    ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P86 (net01489 net01471 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
    ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P91 (net01458 net01588 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
    ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
P100 (net01321 B1 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
    ps=147.000000u pd=147.000000u m=1 region=sat
P101 (net01321 net01279 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 \
    ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
P103 (net01279 A1 vdd! vdd!) ami06P w=96.0u l=600n as=1.44e-10 ad=1.44e-10 \
    ps=195.000000u pd=195.000000u m=1 region=sat
P115 (net01546 net01321 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
    ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
P116 (net01546 net01317 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
    ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
P118 (net01317 A1 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
    ps=99.0u pd=99.0u m=1 region=sat
P117 (net01317 net01280 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
    ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
P119 (net01280 B1 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \

```

ps=99.0u pd=99.0u m=1 region=sat
 P98 (net01475 net01405 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P106 (net01528 net01511 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P89 (S net01528 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P90 (S net01489 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P76 (net01470 net01405 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P77 (net01470 net01465 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P71 (net01465 net01431 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P66 (net01537 net01530 net01538 net01538) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P74 (net01538 net01534 vdd! vdd!) ami06P w=24.0u l=600n as=3.6e-11 \
 ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
 P70 (Cout0 net01537 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P78 (net01534 net01470 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P61 (net01508 A0 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 P62 (net01508 B0 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \

ps=27.0u pd=27.0u m=1 region=sat
 P60 (net01530 net01508 vdd! vdd!) ami06P w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 P96 (net01459 B0 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
 ps=147.000000u pd=147.000000u m=1 region=sat
 P97 (net01459 net01417 vdd! vdd!) ami06P w=72.0u l=600n as=1.08e-10 \
 ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
 P99 (net01417 A0 vdd! vdd!) ami06P w=96.0u l=600n as=1.44e-10 ad=1.44e-10 \
 ps=195.000000u pd=195.000000u m=1 region=sat
 P109 (net01588 net01459 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P110 (net01588 net01455 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P112 (net01455 A0 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 P111 (net01455 net01418 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 P113 (net01418 B0 vdd! vdd!) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 N68 (net01431 B0 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N69 (net01431 A0 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N89 (net01371 net01390 0 net01371) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N94 (S1 net01351 net01371 net01371) ami06N w=12.0u l=600n as=1.8e-11 \

ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N28 (net01514 A2 0 net01514) ami06N w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 N27 (net01515 B2 net01514 net01514) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N21 (net01476 net01467 0 net01476) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N22 (net01477 net01406 net01476 net01476) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N10 (net01440 B2 0 net01440) ami06N w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
 ps=147.000000u pd=147.000000u m=1 region=sat
 N9 (net01463 net01420 net01440 net01440) ami06N w=72.0u l=600n as=1.08e-10 \
 ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
 N4 (net01480 net01463 0 net01480) ami06N w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N1 (net01589 net01457 net01480 net01480) ami06N w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N3 (net01442 A2 0 net01442) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 N0 (net01457 net01421 net01442 net01442) ami06N w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N16 (net01516 net01531 0 net01516) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N17 (S2 net01490 net01516 net01516) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N30 (net01376 A3 0 net01376) ami06N w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \

ps=27.0u pd=27.0u m=1 region=sat
 N32 (net01377 B3 net01376 net01376) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N37 (net01338 net01329 0 net01338) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N40 (net01339 Cin0 net01338 net01338) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N48 (net01302 B3 0 net01302) ami06N w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
 ps=147.000000u pd=147.000000u m=1 region=sat
 N49 (net01325 net01282 net01302 net01302) ami06N w=72.0u l=600n \
 as=1.08e-10 ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat \
 N54 (net01342 net01325 0 net01342) ami06N w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N57 (net01547 net01319 net01342 net01342) ami06N w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N55 (net01304 A3 0 net01304) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 N58 (net01319 net01283 net01304 net01304) ami06N w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N44 (net01378 net01393 0 net01378) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N46 (S3 net01352 net01378 net01378) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N82 (net01332 net01544 net01331 net01331) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N75 (net01331 net01327 0 net01331) ami06N w=12.0u l=600n as=1.8e-11 \

ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N61 (net01369 A1 0 net01369) ami06N w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 N65 (net01370 B1 net01369 net01369) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N97 (net01295 B1 0 net01295) ami06N w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
 ps=147.000000u pd=147.000000u m=1 region=sat
 N101 (net01321 net01279 net01295 net01295) ami06N w=72.0u l=600n \
 as=1.08e-10 ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat \
 N110 (net01335 net01321 0 net01335) ami06N w=48.0u l=600n as=7.2e-11 \
 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N117 (net01546 net01317 net01335 net01335) ami06N w=48.0u l=600n \
 as=7.2e-11 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N111 (net01297 A1 0 net01297) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 N118 (net01317 net01280 net01297 net01297) ami06N w=48.0u l=600n \
 as=7.2e-11 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
 N86 (net01509 net01528 0 net01509) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N92 (S net01489 net01509 net01509) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N80 (net01470 net01405 net01469 net01469) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N70 (net01469 net01465 0 net01469) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N29 (net01532 net01515 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \

pd=15.0u m=1 region=sat

N24 (net01539 net01532 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N25 (net01539 net01536 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N26 (net01544 net01539 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N23 (net01536 net01477 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N87 (net01333 net01544 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N88 (net01333 net01320 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N91 (net01351 net01333 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N95 (net01320 net01546 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N102 (net01337 net01544 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N108 (net01373 net01337 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N109 (net01373 net01546 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N113 (net01390 net01373 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N19 (net01438 A2 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \

pd=15.0u m=1 region=sat
 N18 (net01438 B2 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N20 (net01467 net01438 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N14 (net01478 net01406 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N15 (net01490 net01478 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N13 (net01478 net01462 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N12 (net01462 net01589 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N11 (net01482 net01406 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N8 (net01420 A2 0 0) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 ps=99.0u \
 pd=99.0u m=1 region=sat
 N2 (net01421 B2 0 0) ami06N w=24.0u l=600n as=3.6e-11 ad=3.6e-11 ps=51.0u \
 pd=51.0u m=1 region=sat
 N7 (net01518 net01482 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N6 (net01518 net01589 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N5 (net01531 net01518 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N38 (net01406 net01401 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \

pd=15.0u m=1 region=sat
 N33 (net01401 net01398 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N34 (net01401 net01394 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N31 (net01394 net01377 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N41 (net01398 net01339 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N35 (net01300 A3 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N36 (net01300 B3 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N39 (net01329 net01300 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N42 (net01340 Cin0 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N43 (net01340 net01324 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N45 (net01352 net01340 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N47 (net01324 net01547 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N51 (net01282 A3 0 0) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 ps=99.0u \
 pd=99.0u m=1 region=sat
 N59 (net01283 B3 0 0) ami06N w=24.0u l=600n as=3.6e-11 ad=3.6e-11 ps=51.0u \

pd=51.0u m=1 region=sat

N50 (net01344 Cin0 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N52 (net01380 net01344 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N53 (net01380 net01547 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N56 (net01393 net01380 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N79 (net01327 net01293 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N73 (net01293 A1 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N74 (net01293 B1 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N64 (net01392 net01370 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N72 (net01399 net01392 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N71 (net01399 net01396 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N83 (net01396 net01332 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N78 (net01405 net01399 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

N84 (net01471 net01405 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat

pd=15.0u m=1 region=sat
 N85 (net01471 net01458 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N90 (net01489 net01471 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N93 (net01458 net01588 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N103 (net01279 A1 0 0) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
 ps=99.0u pd=99.0u m=1 region=sat
 N119 (net01280 B1 0 0) ami06N w=24.0u l=600n as=3.6e-11 ad=3.6e-11 \
 ps=51.0u pd=51.0u m=1 region=sat
 N99 (net01475 net01405 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N104 (net01511 net01475 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N105 (net01511 net01588 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N112 (net01528 net01511 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N77 (net01465 net01431 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
 pd=15.0u m=1 region=sat
 N60 (net01507 A0 0 net01507) ami06N w=12.0u l=600n as=1.8e-11 ad=1.8e-11 \
 ps=27.0u pd=27.0u m=1 region=sat
 N63 (net01508 B0 net01507 net01507) ami06N w=12.0u l=600n as=1.8e-11 \
 ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
 N76 (Cout0 net01537 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \

```

pd=15.0u m=1 region=sat
N81 (net01534 net01470 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat
N96 (net01433 B0 0 net01433) ami06N w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
ps=147.000000u pd=147.000000u m=1 region=sat
N98 (net01459 net01417 net01433 net01433) ami06N w=72.0u l=600n \
as=1.08e-10 ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat \
N106 (net01473 net01459 0 net01473) ami06N w=48.0u l=600n as=7.2e-11 \
ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
N114 (net01588 net01455 net01473 net01473) ami06N w=48.0u l=600n \
as=7.2e-11 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
N107 (net01435 A0 0 net01435) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
ps=99.0u pd=99.0u m=1 region=sat
N115 (net01455 net01418 net01435 net01435) ami06N w=48.0u l=600n \
as=7.2e-11 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
N62 (net01530 net01508 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat
N67 (net01537 net01530 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat
N66 (net01537 net01534 0 0) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
pd=15.0u m=1 region=sat
N100 (net01417 A0 0 0) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
ps=99.0u pd=99.0u m=1 region=sat
N116 (net01418 B0 0 0) ami06N w=24.0u l=600n as=3.6e-11 ad=3.6e-11 \
ps=51.0u pd=51.0u m=1 region=sat
simulatorOptions options reltol=1e-3 vabstol=1e-6 iabstol=1e-12 temp=27 \

```

```
tnom=27 scalem=1.0 scale=1.0 gmin=1e-12 rforce=1 maxnotes=5 maxwarns=5 \  
digits=5 cols=80 pivrel=1e-3 sensfile="../psf/sens.output" \  
checklimitdest=psf  
modelParameter info what=models where=rawfile  
element info what=inst where=rawfile  
outputParameter info what=output where=rawfile  
designParamVals info what=parameters where=rawfile  
primitives info what=primitives where=rawfile  
subckts info what=subckts where=rawfile  
saveOptions options save=allpub
```

D.6 Gate-Level Netlist for the Modified (OAI21 Gate Input Switch) 4-bit Ripple Carry Adder

```
// Generated for: spectre
// Generated on: Mar  3 21:00:44 2015
// Design library name: Thesis
// Design cell name: 4_bit_inputswitch_full_adder_flat_virtuoso
// Design view name: schematic
simulator lang=spectre
global 0 vdd!

// Library name: Thesis
// Cell name: 4_bit_inputswitch_full_adder_flat_virtuoso
// View name: schematic
subckt OAI21b0b1_type_0 GND In_AND_1 In_AND_2 In_INV VDD Z
    I0 (GND In_INV net01536 VDD) inverter_type_0
    I1 (In_AND_1 In_AND_2 GND VDD net01532) AND2_type_0
    I2 (net01536 net01532 GND VDD Z) OR2_type_0
ends OAI21b0b1_type_0
// End of subcircuit definition.

subckt OAI21b1_type_0 GND In_INV In_NAND In_OR VDD Z
    I1 (GND In net01320 VDD) inverter_type_0
    I2 (In_OR net01320 GND VDD net01351) OR2_type_0
    I3 (In_NAND net01351 GND VDD Z) NAND2_type_0
ends OAI21b1_type_0
// End of subcircuit definition.
```

```

subckt OAI21_type_0 GND In_NAND In_OR_1 IN_OR_2 VDD Z
    I0 (In_OR_1 In_OR_2 GND VDD net01465 OR2_type_0
    I1 (In_NAND net01465 GND VDD Z) NAND2_type_4
ends OAI21_type_0
// End of subcircuit definition.

```

```

subckt NAND2b0_type_0 A GND In VDD Z
    I0 (A net01337 GND VDD Z) OR2_type_0
    I1 (GND In net01337 VDD) inverter_type_0
ends NAND2b0_type_0
// End of subcircuit definition.

```

```

subckt OR2_type_0 A B GND VDD Z
    I0 (GND net01537 Z VDD) inverter_type_0
    I1 (A B GND VDD net01537) NOR2_type_0
ends OR2_type_0
// End of subcircuit definition.

```

```

subckt AND2_type_0 A B GND VDD Z
    I0 (GND net01508 Z VDD) inverter_type_0
    I1 (A B GND VDD net01508) NAND2_type_0
ends AND2_type_0
// End of subcircuit definition.

```

```

subckt NOR2_type_0 A B GND VDD Z

```

```

N1 (Z B GND GND) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
    pd=15.0u m=1 region=sat
N0 (Z A GND GND) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
    pd=15.0u m=1 region=sat
P1 (Z B net01538 net01538) ami06P w=24.0u l=600n as=3.6e-11 \
    ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
P0 (net01538 A VDD VDD) ami06P w=24.0u l=600n as=3.6e-11 \
    ad=3.6e-11 ps=51.0u pd=51.0u m=1 region=sat
ends NOR2_type_0
// End of subcircuit definition.

subckt NAND2_type_0 A B GND VDD Z
    N1 (net01440 B GND net01440) ami06N w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
        ps=147.000000u pd=147.000000u m=1 region=sat
    N0 (Z A net01440 net01440) ami06N w=72.0u l=600n as=1.08e-10 \
        ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
    P0 (Z A VDD VDD) ami06P w=72.0u l=600n as=1.08e-10 \
        ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
    P1 (Z B VDD VDD) ami06P w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
        ps=147.000000u pd=147.000000u m=1 region=sat
ends NAND2_type_0
// End of subcircuit definition.

subckt NAND2_type_1 A B GND VDD Z
    N1 (net01480 B GND net01480) ami06N w=48.0u l=600n as=7.2e-11 \
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat

```

```

N0 (Z A net01480 net01480) ami06N w=48.0u l=600n as=7.2e-11 \
    ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
P0 (Z A VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 \
    ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
P1 (Z B VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 \
    ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
ends NAND2_type_1

// End of subcircuit definition.

subckt NAND2_type_2 A B GND VDD Z
    N1 (net01302 B GND net01302) ami06N w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
        ps=147.000000u pd=147.000000u m=1 region=sat
    N0 (Z A net01302 net01302) ami06N w=72.0u l=600n \
        as=1.08e-10 ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat \
    P0 (Z A VDD VDD) ami06P w=72.0u l=600n as=1.08e-10 \
        ad=1.08e-10 ps=147.000000u pd=147.000000u m=1 region=sat
    P1 (Z B VDD VDD) ami06P w=72.0u l=600n as=1.08e-10 ad=1.08e-10 \
        ps=147.000000u pd=147.000000u m=1 region=sat
ends NAND2_type_2

// End of subcircuit definition.

subckt NAND2_type_3 A B GND VDD Z
    N1 (net01335 B GND net01335) ami06N w=48.0u l=600n as=7.2e-11 \
        ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
    N0 (Z A net01335 net01335) ami06N w=48.0u l=600n \
        as=7.2e-11 ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat

```

```

P0 (Z A VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 \
    ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
P1 (Z B VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 \
    ad=7.2e-11 ps=99.0u pd=99.0u m=1 region=sat
ends NAND2_type_3

// End of subcircuit definition.

subckt NAND2_type_4 A B GND VDD Z
    N1 (net01469 B GND net01469) ami06N w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
    N0 (Z A net01469 net01469) ami06N w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
    P0 (Z A VDD VDD) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
    P1 (Z B VDD VDD) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
ends NAND2_type_4

// End of subcircuit definition.

subckt inverter_type_0 GND In Out VDD
    N0 (Out In GND GND) ami06N w=48.0u l=600n as=7.2e-11 ad=7.2e-11 ps=99.0u \
        pd=99.0u m=1 region=sat
    P0 (Out In VDD VDD) ami06P w=96.0u l=600n as=1.44e-10 ad=1.44e-10 \
        ps=195.000000u pd=195.000000u m=1 region=sat
ends inverter_type_0

// End of subcircuit definition.

```

```

subckt inverter_type_1 GND In Out VDD
    N0 (Out In GND GND) ami06N w=24.0u l=600n as=3.6e-11 ad=3.6e-11 ps=51.0u \
        pd=51.0u m=1 region=sat
    P0 (Out In VDD VDD) ami06P w=48.0u l=600n as=7.2e-11 ad=7.2e-11 \
        ps=99.0u pd=99.0u m=1 region=sat
ends inverter_type_1

// End of subcircuit definition.

subckt inverter_type_2 GND In Out VDD
    N0 (Out In GND GND) ami06N w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
        pd=15.0u m=1 region=sat
    P0 (Out In VDD VDD) ami06P w=12.0u l=600n as=1.8e-11 \
        ad=1.8e-11 ps=27.0u pd=27.0u m=1 region=sat
ends inverter_type_2

// End of subcircuit definition.

I35 (0 B0 net01418 vdd!) inverter_type_0
I34 (0 A0 net01417 vdd!) inverter_type_0
I33 (0 B1 net01280 vdd!) inverter_type_0
I32 (0 A1 net01279 vdd!) inverter_type_0
I31 (0 B3 net01283 vdd!) inverter_type_0
I30 (0 A3 net01282 vdd!) inverter_type_0
I29 (0 B2 net01421 vdd!) inverter_type_1
I28 (0 A2 net01420 vdd!) inverter_type_0
I27 (net01418 A0 0 vdd! net01455) NAND2_type_0

```

I26 (net01455 net01459 0 vdd! net01588) NAND2_type_0
 I25 (net01417 B0 0 vdd! net01459) NAND2_type_0
 I24 (net01280 A1 0 vdd! net01317) NAND2_type_0
 I23 (net01317 net01321 0 vdd! net01546) NAND2_type_3
 I22 (net01279 B1 0 vdd! net01321) NAND2_type_0
 I21 (net01283 A3 0 vdd! net01319) NAND2_type_0
 I20 (net01319 net01325 0 vdd! net01547) NAND2_type_0
 I19 (net01282 B3 0 vdd! net01325) NAND2_type_2
 I18 (net01421 A2 0 vdd! net01457) NAND2_type_0
 I17 (net01457 net01463 0 vdd! net01589) NAND2_type_1
 I16 (net01420 B2 0 vdd! net01463) NAND2_type_0
 I15 (net01588 0 net01405 vdd! net01528) NAND2b0_type_0
 I14 (net01547 0 Cin0 vdd! net01393) NAND2b0_type_0
 I13 (net01589 0 net01406 vdd! net01531) NAND2b0_type_0
 I12 (net01546 0 net01544 vdd! net01390) NAND2b0_type_0
 I11 (0 net01544 B1 A1 vdd! net01332) OAI21_type_0
 I10 (0 Cin0 B3 A3 vdd! net01339) OAI21_type_0
 I9 (0 net01406 B2 A2 vdd! net01477) OAI21_type_0
 I8 (0 net01405 A0 B0 vdd! net01470) OAI21_type_0
 I7 (0 net01588 net01528 net01405 vdd! S) OAI21b1_type_0
 I6 (0 net01547 net01393 Cin0 vdd! S3) OAI21b1_type_0
 I5 (0 net01589 net01531 net01406 vdd! S2) OAI21b1_type_0
 I4 (0 net01546 net01390 net01544 vdd! S1) OAI21b1_type_0
 I3 (0 B0 A0 net01470 vdd! Cout0) OAI21b0b1_type_0
 I2 (0 B1 A1 net01332 vdd! net01405) OAI21b0b1_type_0
 I1 (0 B3 A3 net01339 vdd! net01406) OAI21b0b1_type_0

```

I0 (0 B2 A2 net01477 vdd! net01544) OAI21b0b1_type_0
simulatorOptions options reltol=1e-3 vabstol=1e-6 iabstol=1e-12 temp=27 \
    tnom=27 scalem=1.0 scale=1.0 gmin=1e-12 rforce=1 maxnotes=5 maxwarns=5 \
    digits=5 cols=80 pivrel=1e-3 sensfile="../psf/sens.output" \
    checklimitdest=psf
modelParameter info what=models where=rawfile
element info what=inst where=rawfile
outputParameter info what=output where=rawfile
designParamVals info what=parameters where=rawfile
primitives info what=primitives where=rawfile
subckts info what=subckts where=rawfile
saveOptions options save=allpub

```

Bibliography

- [1] “Digital Comparator”. *Electronics Tutorials*. Available:
http://www.electronics-tutorials.ws/combinational/comb_8.html [Last accessed: Dec 2014].
- [2] “How D-gate Master/Slave Flip-flops Work...” *BYU CS224 Computer Systems*. Available:
<https://students.cs.byu.edu/cs224ta/labs/L02-fsm/HowToUseMasterSlave.php> [Last accessed: Dec 2014].
- [3] “The Multiplexer”. *Electronics Tutorials*. Available:
http://www.electronics-tutorials.ws/combinational/comb_2.html [Last accessed: Dec 2014].
- [4] “Trusted Integrated Circuits (TRUST)”. *DARPA Microsystems Technology Office*. Available:
[http://www.darpa.mil/Our_Work/MTO/Programs/Trusted_Integrated_Circuits_\(TRUST\).aspx](http://www.darpa.mil/Our_Work/MTO/Programs/Trusted_Integrated_Circuits_(TRUST).aspx) [Last accessed: May 2014].
- [5] *Knowledge of Software Suppliers Needed to Manage Risks*. Technical report, United States Government Accountability Office, May 2004. URL
<http://www.gao.gov/new.items/d04678.pdf>.
- [6] *Intellectual Property: Observations on Efforts to Quantify the Economic Effects of Counterfeit and Pirated Goods*. Technical report, United States Government Accountability Office, April 2010. URL <http://www.gao.gov/assets/310/303057.pdf>.
- [7] *Inquiry into Counterfeit Electronic Parts in the Department of Defense Supply Chain*. Technical report, United States Senate Committee on Armed Services, May 2012. URL <http://www.armed-services.senate.gov/imo/media/doc/Counterfeit-Electronic-Parts.pdf>.
- [8] Centers, NAVSEA Warfare. “Want to See Some “Fake” Microelectronics?”, 2014. DARPA MTO Exposition.
- [9] Collins, Dean. “DARPA “TRUST in IC’s” Effort”, 2007. URL
<http://www.dtic.mil/docs/citations/ADA503809>. DARPA Microsystems Technology Symposium.
- [10] Evans, David. *Understanding and Mitigating Supply Chain Risks for Computing and Communications (or: Who’s Driving Your Missiles?)*. Technical report, DARPA Defense Science Study Group.

- [11] Grow, Brian, Chi-Chu Tschang, Cliff Edwards, and Brian Burnsed. “Dangerous Fakes”. *BusinessWeek*, 1 Oct 2008. Available: <http://www.BusinessWeek.com/stories/2008-10-01/dangerous-fakes> [Last accessed: May 2014].
- [12] Kim, W. and H. Shin. “Hierarchical LVS based on hierarchy rebuilding”. *Design and Test Workshop (IDT), 2009 4th International*, 379–384. 1998.
- [13] Komaroff, Mitchell and Kristen Baldwin. “DoD Software Assurance Initiative”. URL <https://acc.dau.mil/adl/en-US/25749/file/3178/DoD%20SW%20Assurance%20Initiative.pdf>.
- [14] Levin, Carl. *S.Amdt.1092 to S.1867*. Technical report, U.S. Senate, Nov 2011. URL <http://beta.congress.gov/amendment/112th-congress/senate-amendment/1092>.
- [15] Mike Rogers and C.A. Dutch Ruppersberger. *Investigative Report on the U.S. National Security Issues Posed by Chinese Telecommunications Companies Huawei and ZTE*. Technical report, U.S. House of Representatives, Oct 2012. URL [https://intelligence.house.gov/sites/intelligence.house.gov/files/documents/Huawei-ZTE%20Investigative%20Report%20\(FINAL\).pdf](https://intelligence.house.gov/sites/intelligence.house.gov/files/documents/Huawei-ZTE%20Investigative%20Report%20(FINAL).pdf).
- [16] Pecht, Michael and Sanjay Tiku. “Bogus!” *IEEE Spectrum*, 1 May 2006. Available: <http://spectrum.ieee.org/computing/hardware/bogus> [Last accessed: May 2014].
- [17] Savitz, Eric. “The Serious Risks From Counterfeit Electronic Parts”. *Forbes*, 11 July 2012. Available: <http://www.forbes.com/sites/ciocentral/2012/07/11/the-serious-risks-from-counterfeit-electronic-parts/> [Last accessed: May 2014].
- [18] Sciences, Applied DNA. “President Signs FY’12 National Defense Authorization Act Into Law”. *Yahoo! Finance*, 3 January 2012. Available: <http://finance.yahoo.com/news/President-Signs-FY-12-iw-3225445251.html> [Last accessed: May 2014].
- [19] Seery, Michael K. *Complex VLSI Feature Comparison For Commercial Microelectronics Verification*. Master’s thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, 2014.
- [20] Villasenor, John and Mohammad Tehranipoor. “The Hidden Dangers of Chop-Shop Electronics”. *IEEE Spectrum*, 20 September 2013. Available: <http://spectrum.ieee.org/semiconductors/processors/the-hidden-dangers-of-chopshop-electronics> [Last accessed: May 2014].
- [21] Weste, Neil H. E. and David Money Harris. *CMOS VLSI Design, A Circuits and Systems Perspective, Fourth Edition*.

- [22] Wolf, Jim. “U.S. lawmakers seek to block China Huawei, ZTE U.S. inroads”. *Reuters*, 8 October 2012. Available: <http://www.reuters.com/article/2012/10/08/us-usa-china-huawei-zte-idUSBRE8960NH20121008> [Last accessed: May 2014].

REPORT DOCUMENTATION PAGE				<i>Form Approved</i> OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 26-03-2015		2. REPORT TYPE Master's Thesis		3. DATES COVERED (From — To) Oct 2013–Mar 2015	
4. TITLE AND SUBTITLE Gate-Level Commercial Microelectronics Verification with Standard Cell Recognition				5a. CONTRACT NUMBER in house	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Hsia, Leleia A., Second Lieutenant, USAF				5d. PROJECT NUMBER JON14G150	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB, OH 45433-7765				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT-ENG-MS-15-M-069	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Bradley Paul Mixed Signal Design Center, Sensors Directorate Air Force Research Laboratory 2241 Avionics Circle, Bldg 600 Wright-Patterson AFB, OH 45433-7318 (937)528-8706, Bradley.Paul@us.af.mil				10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/Rydi	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED					
13. SUPPLEMENTARY NOTES This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.					
14. ABSTRACT Within the past two decades, the problem of counterfeit hardware has gained significant attention within the Department of Defense (DoD). Counterfeit electronics compromise national security systems as they may fail to meet durability requirements and/or contain malicious circuits [6, 16, 17]. This necessitates the development of methods to detect counterfeit electronics and prevent the counterfeit electronics from entering DoD systems. The DARPA TRUST program was established to address the need to verify integrated circuit (IC) electronics. This research describes the development of standard cell recognition (SCR) software intended to resolve conflicts in prior TRUST related applications of commercial software to verify IC designs. SCR software applications to circuits composed of up to 650 transistors are presented, and the resulting 90% SCR application success rate is discussed.					
15. SUBJECT TERMS SCR, TRUST, verification, VLSI					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 347	19a. NAME OF RESPONSIBLE PERSON Maj Derrick Langley (ENG)
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (include area code) (937) 255-3636 x6165 derrick.langley@afit.edu