5-2011

Reducing energy usage of NULL Convention Logic circuits using NULL Cycle Reduction combined with supply voltage scaling

Brett Sparkman
University of Arkansas, Fayetteville

REDUCING ENERGY USAGE OF NULL CONVENTION LOGIC CIRCUITS USING NULL CYCLE REDUCTION COMBINED WITH SUPPLY VOLTAGE SCALING

REDUCING ENERGY USAGE OF NULL CONVENTION LOGIC CIRCUITS USING NULL CYCLE REDUCTION COMBINED WITH SUPPLY VOLTAGE SCALING

A thesis submitted to the Honors College in partial
fulfillment of the requirements for the degree of
Honors Bachelors of Science
in Electrical Engineering

By

Brett Sparkman

May 2011
University of Arkansas

ABSTRACT

The NULL Cycle Reduction (NCR) technique can be used to improve the performance of a NULL Convention Logic (NCL) circuit at the expense of power and area. However, by decreasing the supply voltage of certain components, the power of the NCR circuit can be reduced. Since NCR has increased performance, it could be possible to decrease the power while maintaining the original performance of the circuit.

To verify this, the NCR circuit will be implemented using a 4-bit by 4-bit dual-rail multiplier as the test circuit. This circuit will be simulated in ModelSim to ensure functionality, synthesized into a Verilog netlist using Leonardo, and imported into Cadence to perform transistor-level simulations for power calculations. The supply voltage of the duplicated circuits will be decreased until the performance matches the design of the original multiplier, resulting in overall lower energy usage.

This thesis is approved for recommendation to the Honors College.

Thesis Director:

Scott C. Smith

Dr. Scott C. Smith

THESIS DUPLICATION RELEASE

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

Agreed Brett Spearman

Refused _____

ACKNOWLEDGMENTS

I thank Dr. Scott Smith, my thesis advisor, for the project topic and opportunity to conduct this undergraduate research. His continued assistance with verification and optimistic outlook truly made this an enjoyable experience.

In addition, I thank Liang Zhou, a graduate student, for his continued help with all of the software used during the project. This included synthesizing VHDL files in Leonardo, importing the files into Cadence, and running simulations in Cadence using UltraSim.

I would also like to thank my fiancée, Alexandra Gammill, for the continued support throughout this project and my mother, Michele Walker, for the proofreading assistance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Problem

As circuits are continually produced with increasing numbers of transistors and switching frequencies, circuit power also increases. Although these improvements can drastically raise the performance of circuits, they also have a downside: the circuits consume larger amounts of power. This increase in power consumption has several downsides: the circuits will heat up more due to higher power dissipation, the circuits will last shorter amounts of time on a single battery charge, and the circuits have a higher cost of operation for the same amount of time.

1.2 Thesis Statement

The goal of this research is to investigate applying the NULL Cycle Reduction (NCR) technique to a circuit and reducing the supply voltage of the duplicated portion in an effort to reduce the overall energy usage of the circuit while maintaining equivalent performance.

1.3 Approach

In order to determine if reducing the supply voltage of a circuit can reduce its power, a series of simulations was performed. First, a simulation of the VHDL design was performed in ModelSim to ensure that the circuit performed as desired. Next, the files were synthesized using Leonardo in order to generate a Verilog netlist, which was then imported into cadence. The final steps involved running numerous transistor-level simulations in Cadence to determine the effects of reducing the supply voltage in terms of power and performance.

1.4 Potential Impact

This research has the potential to impact power reduction methods used by Dr. Smith and his graduate students. If a standard circuit performs as desired but consumes too much power, this technique could be applied to lower the power of the circuit while maintaining the performance.

2. BACKGROUND

2.1 Power Reduction

The power of a circuit is given by the equation:

$$P = \propto C_L V_{DD}^2 f + \propto t_{sc} V_{DD} I_{peak} + V_{DD} I_{leakage}$$

where $\propto$ is the activity factor, C_L is the capacitance of the circuit, V_{DD} is the supply voltage, f is the clock frequency, t_{sc} is the short-circuit time, I_{peak} is the short-circuit current spike amplitude, and $I_{leakage}$ is the leakage current of the transistors. Since V_{DD} is present in all three terms, a large reduction in power consumption can be achieved by reducing the supply voltage. However, reducing the voltage can have a negative impact on the circuit, decreasing the performance and potentially causing the circuit to perform incorrectly. This resulting dilemma is one of the large tradeoffs in digital design: performance vs. power.

2.2 NULL Convention Logic (NCL) Overview [1]

The following two sections are the work of Dr. Scott C. Smith. NCL offers a self-timed logic paradigm where control is inherent with each datum. NCL follows the so-called ''weak conditions'' of Seitz's delay-insensitive signaling scheme [2]. As with other self-timed logic methods discussed herein, the NCL paradigm assumes that forks in wires are isochronic [3]. The origins of various aspects of the paradigm, including the NULL (or spacer) logic state from which NCL derives its name, can be traced back to Muller's work on speed-independent circuits in the 1950s and 1960s [4].

2.2.1 Delay-insensitivity

NCL uses symbolic completeness of expression [5] to achieve delay-insensitive behavior. A symbolically complete expression is defined as an expression that only depends on the relationships of the symbols present in the expression without a reference to their time of evaluation. In particular, dual-rail signals or other *Mutually Exclusive Assertion Groups* (MEAGs) can be used to incorporate data and control information into one mixed signal path to eliminate time reference [6]. A dual-rail signal, D, consists of two wires, D^0 and D^1, which may assume any value from the set {DATA0, DATA1, NULL}. The DATA0 state ($D^0 = 1$, $D^1 = 0$) corresponds to a Boolean logic 0, the DATA1 state ($D^0 = 0$, $D^1 = 1$) corresponds to a Boolean logic 1, and the NULL state ($D^0 = 0$, $D^1 = 0$) corresponds to the empty set meaning that the value of D is not yet available. The two rails are mutually exclusive so that both rails can never be asserted simultaneously; this state is defined as an illegal state. Dual-rail signals are space optimal 1-out-of-N delay-insensitive codes requiring two wires per bit. Other higher order MEAGs are not wire count optimal; however, they can be more power efficient due to the decreased number of transitions per cycle.

Most multi-rail delay-insensitive systems [2,5,7], including NCL, have at least two register stages, one at both the input and at the output. Two adjacent register stages interact through their request and acknowledge lines, K_i and K_o, respectively, to prevent the current DATA wavefront from overwriting the previous DATA wavefront, by ensuring that the two DATA wavefronts are always separated by a NULL wavefront.

2.2.2 Logic gates

NCL, like [3], differs from the other delay-insensitive paradigms [2,7] in that these other paradigms only utilize one type of state-holding gate, the *C-element* [4]. A C-element behaves as follows: when all inputs assume the same value, then the output assumes this value; otherwise the output does not change. On the other hand, all NCL gates are state-holding. Thus, NCL optimization methods can be considered as a subclass of the techniques for developing delay-insensitive circuits using a pre-defined set of more complex components, with built-in *hysteresis* behavior.

NCL uses *threshold gates* for its basic logic elements [8]. The primary type of threshold gate is the TH*mn* gate, where $1 \leq m \leq n$, as depicted in Figure 1. TH*mn* gates have n inputs. At least m of the n inputs must be asserted before the output will become asserted. Because NCL threshold gates are designed with hysteresis, all asserted inputs must be de-asserted before the output will be de-asserted. Hysteresis ensures a complete transition of inputs back to NULL before asserting the output associated with the next wavefront of input data. Therefore, a TH*nn* gate is equivalent to an n-input C-element and a TH1n gate is equivalent to an n-input OR gate. In a TH*mn* gate, each of the n inputs is connected to the rounded portion of the gate; the output emanates from the pointed end of the gate; and the gate's threshold value, m, is written inside of the gate. NCL threshold gates may also include a *reset* input to initialize the output. Resettable gates are denoted by either a D or an N appearing inside the gate, along with the gate's threshold, referring to the gate being reset to logic 1 or logic 0, respectively.

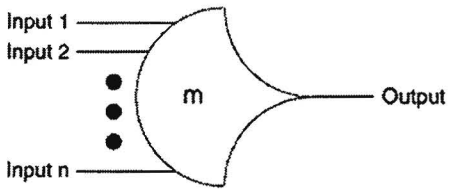

Figure 1. Th*mn* threshold gate. [1]

By employing threshold gates for each logic rail, NCL is able to determine the output status without referencing time. Inputs are partitioned into two separate wavefronts, the NULL wavefront and the DATA wavefront. The NULL wavefront consists of all inputs to a circuit being NULL, while the DATA wavefront refers to all inputs being DATA, some combination of DATA0 and DATA1 for dual-rail inputs. Initially, all circuit elements are reset to the NULL state. First, a DATA wavefront is presented to the circuit. Once all of the outputs of the circuit transition to DATA, the NULL wavefront is presented to the circuit. After all of the outputs of the circuit transition to NULL, the next DATA wavefront is presented to the circuit. This DATA/NULL cycle continues repeatedly. As soon as all outputs of the circuit are DATA, the circuit's result is valid. The NULL wavefront then transitions all of these DATA outputs back to NULL. When the outputs transition back to DATA again, the next output is available. This period is referred to as the DATA-to-DATA cycle time, denoted as T_{DD}, and has an analogous role to the clock period in a synchronous system.

2.2.3 Completeness

The completeness of input criterion [5], which NCL combinational circuits and circuits developed from other delay-insensitive paradigms [2,7] must maintain in order to be delay-insensitive, requires the following criteria: 1. all the outputs of a combinational circuit may not transition from NULL to DATA until all inputs have transitioned from NULL to DATA, and 2.

all the outputs of a combinational circuit may not transition from DATA to NULL until all inputs have transitioned from DATA to NULL. In circuits with multiple outputs, it is acceptable, according to Seitz's weak conditions [2], for some of the outputs to transition without having a complete input set present, as long as all outputs cannot transition before all inputs arrive.

Furthermore, circuits must also adhere to the completion-completeness criterion [9], which requires that completion signals only be generated such that no two adjacent DATA wavefronts can interact within any combinational component. This condition is only necessary when the bit-wise completion strategy is used with selective input-incomplete components, since it is inherent when using the full-word completion strategy and when using the bit-wise completion strategy with no input-incomplete components [9].

2.2.4. Observability

One more condition must be met to ensure delay-insensitivity for NCL and other delay-insensitive circuits [2,7]. No *orphans* may propagate through a gate [10]. An orphan is defined as a wire that transitions during the current DATA wavefront, but is not used in the determination of the output. Orphans are caused by wire forks and can be neglected through the isochronic fork assumption [3] as long as they are not allowed to cross a gate boundary. This *observability* condition, also referred to as indicatability or stability, ensures that every gate transition is observable at the output, which means that every gate that transitions is necessary to transition at least one of the outputs.

2.3 NULL Cycle Reduction (NCR) Overview [1]

The technique for reducing the NULL cycle, thus increasing throughput for any delay-insensitive circuit developed according to the paradigms [2,5,7], is shown in Figure 2. The NCR

architecture in Figure 2 is specifically designed for dual-rail circuits utilizing full-word completion, where all bits at the output of a registration stage are conjoined to form one completion signal. Bit-wise completion only sends the completion signal from bit *b* in register back to the bits in register$_{i-1}$ that took part in the calculation of bit *b*. This method may therefore require fewer logic levels in the completion circuitry than that of full-word completion, thus increasing throughput.

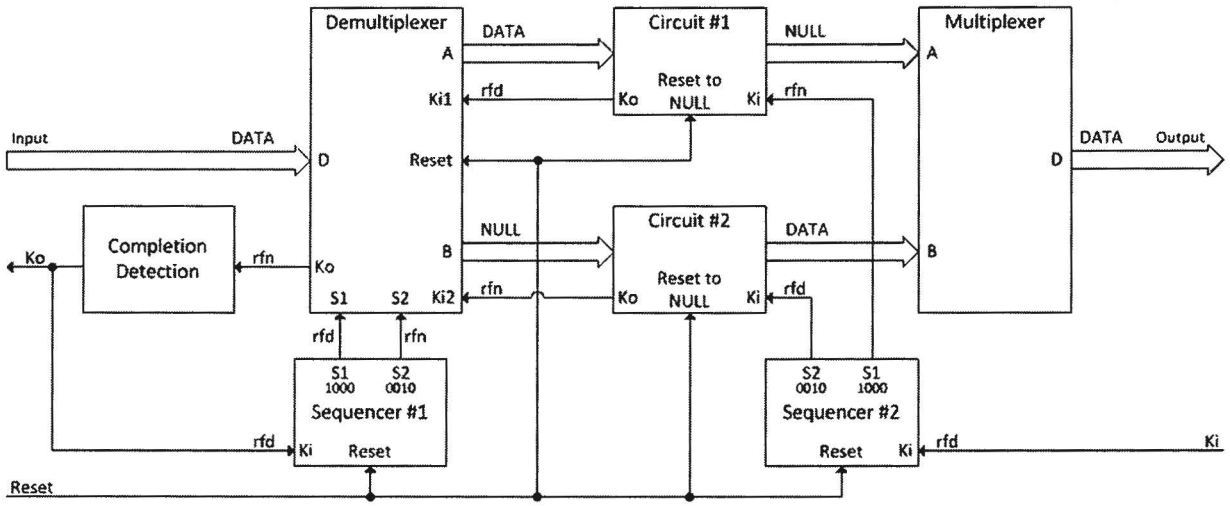

Figure 2. NCR architecture.

Circuit #1 and *Circuit #2* are both dual-rail delay-insensitive combinational circuits utilizing full-word completion, developed from one of the following delay-insensitive paradigms [2,5,7], with at least an input and output registration stage. (Additional registration stages may be present, thus further partitioning the combinational circuitry.) Both circuits have identical functionality and are both initialized to output NULL and request DATA upon reset. In the case of the NCL paradigm, the combinational functionality can be designed using the Threshold Combinational Reduction method described in [11]; and the resulting circuit can also be pipelined, as described in [12], to further increase throughput. The *Demultiplexer* partitions the input, *D*, into two outputs, *A* and *B*, such that *A* receives the first DATA/NULL cycle and *B*

receives the second DATA/NULL cycle. The input continuously alternates between *A* and *B*. The *Completion Detection* circuitry detects when either a complete DATA or NULL wavefront has propagated through the Demultiplexer and requests the next NULL or DATA wavefront, respectively. *Sequencer #1* is controlled by the output of the Completion Detection circuitry and is used to select either output *A* or *B* of the Demultiplexer. Output *A* of the Demultiplexer is input to Circuit #1, when requested by K_{i1}; and output *B* of the Demultiplexer is input to Circuit #2, when requested by K_{i2}. The outputs of Circuit #1 and Circuit #2 are allowed to pass through their respective output registers, as determined by *Sequencer #2*, which is controlled by the external request, K_i. The Multiplexer rejoins the partitioned data path by passing a DATA input on either *A* or *B* to the output, or asserting NULL on the output when both *A* and *B* are NULL. Figure 2 shows the state of the system when a DATA wavefront is being input before its acknowledge flows through the Completion Detection circuitry, and when a DATA wavefront is being output before it is acknowledged by the receiver.

3. APPROACH AND IMPLEMENTATION

3.1 VHDL with ModelSim

Previously used VHDL files were supplied by Dr. Smith to allow the project to start. These files included a 4-bit by 4-bit multiplier to use as the test circuit and a testbench to test the circuit. Also included were the files necessary for implementing the NCR architecture: a dual-rail signal declaration, mappings of common NCL gates, a demultiplexer, a multiplexer, a sequence generator, and the completion detection circuitry.

3.1.1 Modifications to VHDL

Unfortunately, when the previous VHDL code was written, it was not intended for use in Cadence. Several of the K_i, K_o, and reset signals were declared as dual-rail input or outputs. Only one of the rails was used in the design, and the unused rail could cause potential problems when running simulations in Cadence. All of the design files using these dual-rail signals were modified to be standard logic input signals.

There were further modifications that needed to be done to allow the code to work properly. Since the design was old, it was checked with the Cadence libraries for NCL component discrepancies. Unfortunately, the mappings for TH12bx0, TH24compx0, and THand0x0 were different. To correct this error, the NCL map file was altered to account for the different input and output mappings.

3.1.2 Simulating the Original Design

Initially, a single multiplier, shown in Figure 3, was compiled with its testbench and simulated in ModelSim to ensure that functionality. Several output vectors were compared to

their desired values based on the input. These outputs were correct, so the design was functioning properly. The testbench also included an "incorrect" signal that would transition to logic high if an output was incorrect. This always stayed logic low, so the design was functioning as desired. This signal was used in future simulations to ease the functionality checking.

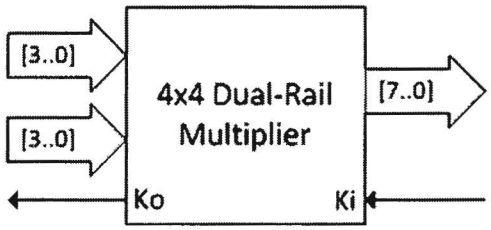

Figure 3. One 4-bit by 4-bit multiplier.

3.1.3 Simulating the NCR Design

Using the NCR architecture shown in Figure 2, Circuit #1 and Circuit #2 each consisted of a 4-bit by 4-bit multiplier. The design files, found in Appendix A and the following files from [13], NCL_signals.vhd, NCL_gates.vhd, NCL_components.vhd, and NCL_functions.vhd, were compiled, and the modified design was simulated to guarantee that it also was functional. The incorrect signal always remained low, so the design was functioning properly.

3.2 Verilog Synthesis in Leonardo

To perform power simulations using Cadence, it was necessary to have a Verilog netlist of the circuit. Leonardo, a VHDL to Verilog synthesis tool, was used to generate this file. A series of steps had to be followed in order to secure a proper Verilog file generation.

3.2.1 Importing VHDL

To import the VHDL, a sample library was used in Leonardo. The ASIC/Sample/SCL05u library was loaded. The design files were then read into Leonardo in a top-down order to ensure that all entities were mapped properly. The design was then optimized through flattening. Once this was done, the correct netlist was generated by selecting the output type to be Verilog.

3.2.2 Running Scripts

Unfortunately, the generated file was not ready to be imported into Cadence: it lacked the fanout, buffering, and supply voltage and ground signals required to perform power simulation. To fix this, a series of scripts were run on the generated Verilog file. Before running any scripts, however, the current Verilog file needed to be modified. The comments created by Leonardo were removed at the beginning, and any additional module definitions other than the multiplier design were deleted. The file could now be read properly by the scripts. The first script, fan.py, inserted the fanout. The second script, buffer.py added in the necessary buffering. The final script, AddPowGndFlatten.py, added in the supply voltage and ground signals.

3.3 Cadence Spectre

Utilizing Cadence Spectre was the final step in simulating the circuits. Cadence was used to perform the numerous simulations of the circuits with altered supply voltages. The power was measured from these simulations, and the results were compiled.

3.3.1 Importing Verilog

The Verilog netlist, found in Appendix B, was easily imported using the "Import Verilog" feature in Cadence. The Target Library Name was NCL_lvt_Li_zhen_brett, a low

threshold voltage library copied so modifications could be done without affecting any other designs or cluttering up a commonly used library. Under Schematic Generation Options, Full Place and Route was disabled. Doing this reduced the amount of wires on the schematic; all of the component input and output pins were tied together using net names. Both the single multiplier and the NCR design were imported. Once imported, the specific portions of the NCR design were given their own voltage sources whose values could easily be modified by changing parameters. The parameter names assigned to the multiple supply voltages were as follows: V_{global} for the demultiplexer, sequencer #1, and completion detection circuitry; V_{local} for the circuit #1 and circuit #2; V_{mux} for the mux; and V_{sel} for the sequencer #2.

3.3.2 Generating Controller

In order to simulate the design, a control circuit that would generate the input patterns was necessary. This design was contributed by Liang Zhou, who had a controller written in VerilogA for another circuit that he had worked on previously. This controller was imported into Cadence using the method mentioned in Section 3.3.1. The generated symbol was put into a schematic along with the multiplier design. Initially, a single input vector of all 1's was included to verify the circuit functionality after being imported into Cadence. Once all of the designs performed as expected, the controller's output vector was modified to generate a random set of inputs using the VerilogA random() function. A parameter was included within the parenthesis to generate the same inputs every time the controller was simulated. The random value was checked to see if it was even or odd, and then the controller assigned the dual-rail signals to be either a 1 or 0, respectively.

3.3.3 Power Simulations

To simulate the designs, the Analog Design Environment within Cadence was used. UltraSim was chosen as the simulator to produce fairly accurate results quickly. A transient analysis was performed from 0ns to 150ns. To easily modify the supply voltages, the parameters mentioned in Section 3.3.1 were incorporated and modified as the simulation required. The supply voltage currents, reset, K_i, K_o, and input and output signals were set to be plotted and saved. Once these settings were correct, initial simulations were run for the single multiplier and NCR designs.

After the first simulation of each design was finished, the output plots were checked to safeguard that the design was properly imported. The controller was then modified as described in Section 3.3.2 to generate a random input. The designs were then re-simulated using a range of supply voltages with the new input patterns, and the outputs were plotted.

For each simulation using the NCR design, the V_{local}, V_{mux}, and V_{sel} were reduced in certain sets. The first voltage reduced was V_{local} because circuit #1 and circuit #2 were larger than the multiplexor or sequence generator #2. Reducing only V_{local} would reduce the overall power the most effectively. The next voltage reduced was V_{mux} because the multiplexer was larger than sequence generator #2. The last voltage reduced was V_{sel} because the output select was the smallest out of the three components that the reduced voltage could be applied to. These were reduced until the period and power of the NCR design were lower than that of the single multiplier design, if possible, with a smallest resolution of 10mV.

On the simulation plots, it was noted that the outputs took a short amount of time before they began appearing. This delay occurred because the pipeline took a small amount of time to fill up before the correct output could be observed. In order to calculate the period of the circuit,

the period of the main K_o was averaged between the 10th rising edge and the 20th rising edge. Taking the average in this manner ensured that the circuit had reached a steady state. Similarly, the currents of all the voltage supplies were integrated from 50ns to 150ns to determine the energy used by the circuit. From this data, the energy per operation was calculated using the following equation:

$$\frac{Energy}{Operation} = \frac{\sum_{x=1}^{n}\left[V_x * \int_{50ns}^{150ns} I_x dt\right]}{\frac{100ns}{T}}$$

The results for the single multiplier design and the NCR design are shown below in Table 1.

Energy Calculation and Delay										
Design	V_{global} (V)	I_{global} (μA)	V_{local} (V)	I_{local} (μA)	V_{mux} (V)	I_{mux} (μA)	V_{sel} (V)	I_{sel} (μA)	Period (ns)	Energy / Op (μJ)
Single Multiplier	1.20	121.5							4.18	6.10
NCR Design	1.20	52.38	1.20	153.1	1.20	5.14	1.20	7.19	3.33	8.71
	1.20	45.60	1.10	122.4	1.20	4.62	1.20	6.33	3.76	7.61
	1.20	45.39	1.10	121.6	1.10	4.11	1.20	6.21	3.78	7.56
	1.20	45.48	1.10	120.4	1.10	4.08	1.10	5.48	3.80	7.51
	1.20	39.49	1.00	95.0	1.20	4.30	1.20	5.46	4.36	6.72
	1.20	39.52	1.00	94.7	1.00	3.18	1.20	5.39	4.39	6.67
	1.20	39.26	1.00	92.8	1.00	3.13	1.00	4.23	4.46	6.56

Table 1. One-multiplier design results.

Unfortunately, there was no possibility of reducing the supply voltages of the NCR design so that the period and power would be less than the single multiplier circuit. The reduction of power with a comparable delay was closest when V_{local} was reduced to 1.00V and everything else remained at 1.2V. The period and power could not be reduced where both would be better than the single multiplier because circuit #1 and circuit #2, the multiplier copies, were fairly small; hence, the overhead of the added DEMUX, MUX, and Sequencers outweighed the power savings. If the circuit that was duplicated was larger, a larger power reduction would be

seen when compared to the period increase, potentially allowing the circuit to be lower power and faster.

3.3.4 Two-Multiplier Circuit Modification

To produce a circuit with a lower power and period would require enlarging the duplicated circuit so that reducing the supply voltage would lessen power by a larger factor. Two additional circuits were designed and simulated to test this hypothesis. Although these circuits used the same multiplier, there were more copies of the multiplier which formed the larger circuit.

The first additional circuit that was designed and simulated was two of the multipliers in series, as shown in Figure 4. The new circuit no longer performed the same function as the original circuit, but it served as a simple example of a larger circuit. The two-multiplier circuit would now take the place of circuit #1 and circuit #2 in the NCR architecture. The 8-bit output was split apart and sent to the two 4-bit inputs of the multiplier. In order to properly simulate the design, the entire process from simulations in ModelSim to Verilog Synthesis to Cadence simulation needed to be performed. The multiplier VHDL file was copied and altered to contain two multipliers, tying the output of one to the input of another as shown in Appendix C.1. The incorrect signal in the testbench was also modified to produce the correct output of the multipliers strung together. The two-multiplier circuit and the new NCR design were simulated, and the incorrect signal remained low, indicating a functional circuit.

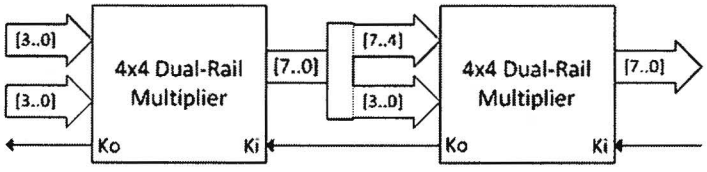

Figure 4. Two-multiplier design.

The VHDL files were synthesized into Verilog netlists and imported into Cadence as mentioned in Section 3.3.1. The schematics had to be modified to include the multiple supply voltages, and the controller had to be included into the schematics as well. An all-1's input vector was simulated in UltraSim to ensure that the outputs of the imported two-multiplier circuit and two-multiplier NCR design were correct, and the results matched the expected values. The controller's outputs were modified again to match the same random input values of the multiplier circuits as before. A series of simulations was performed, as in Section 3.3.3, and the results are shown below in Table 2.

Energy Calculation and Delay										
Design	V_{global} (V)	I_{global} (μA)	V_{local} (V)	I_{local} (μA)	V_{mux} (V)	I_{mux} (μA)	V_{sel} (V)	I_{sel} (μA)	Period (ns)	Energy / Op (μJ)
Single Multiplier	1.20	241.0							4.22	12.21
NCR Design	1.20	51.20	1.20	303.0	1.20	5.11	1.20	6.69	3.39	14.87
	1.20	45.44	1.10	242.4	1.20	4.58	1.20	6.03	3.83	12.77
	1.20	45.59	1.10	241.7	1.10	4.06	1.20	6.03	3.84	12.75
	1.20	45.38	1.10	238.9	1.10	4.01	1.10	5.23	3.86	12.65
	1.20	**41.47**	**1.05**	**212.4**	**1.05**	**3.50**	**1.05**	**4.54**	**4.16**	**11.70**
	1.20	**41.22**	**1.04**	**210.0**	**1.20**	**4.31**	**1.20**	**5.42**	**4.16**	**11.64**
	1.20	**41.20**	**1.04**	**208.2**	**1.04**	**3.44**	**1.20**	**5.44**	**4.19**	**11.56**
	1.20	41.04	1.04	205.3	1.04	3.43	1.04	4.41	4.23	11.46
	1.20	41.26	1.03	203.1	1.20	4.27	1.20	5.36	4.23	11.43
	1.20	41.03	1.03	202.6	1.03	3.33	1.20	5.29	4.25	11.39
	1.20	39.80	1.03	200.9	1.03	3.33	1.03	4.37	4.30	11.30
	1.20	38.85	1.00	187.1	1.20	4.23	1.20	5.16	4.43	10.86
	1.20	39.10	1.00	185.2	1.00	3.11	1.20	5.15	4.46	10.77
	1.20	38.91	1.00	181.7	1.00	3.07	1.00	3.89	4.52	10.65

Table 2. Two-multiplier design results.

Using the two-multiplier NCR design, it was possible to achieve lower power and a smaller period. The supply voltage parameter settings that accomplished this have been made bold in Table 2. Using a 1.04V V_{local} and V_{mux} while maintaining the 1.2V supply on all other

circuit elements decreased the period by 0.03ns and the energy per operation by 0.65μJ. The decreases correspond to a performance increase of approximately 0.7% and an energy decrease of approximately 5.3%. Although these results were positive, the result was not as beneficial as desired, so another circuit was designed.

3.3.5 Four-Multiplier Circuit Modification

To further show that supply voltage can have a large impact on power consumption when using the NCR design, an even larger third circuit was designed. This circuit simply strung together four of the multipliers, as shown in Figure 5. As with the first modification, the VHDL files had to be edited to account for the additional multiplier circuits, as shown in Appendix C.2. The new four-multiplier circuit and the four-multiplier NCR design were simulated to ensure functionality. The incorrect signal stayed low during the simulation, so the circuit performed as expected.

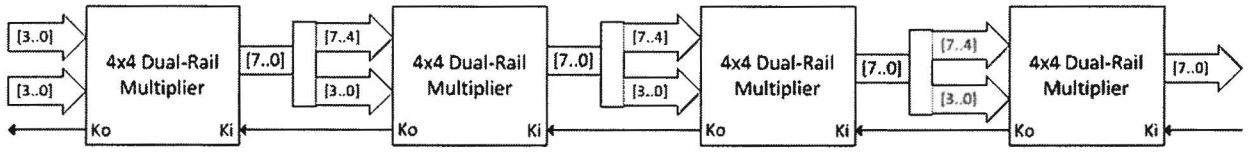

Figure 5. Four-multiplier design.

Once again, the VHDL files were synthesized and imported into Cadence. An all-1's input vector was simulated using the controller, and the results of the four-multiplier circuit and NCR design matched the expected values. Simulations using the same random inputs were performed, and the results are shown below in Table 3.

Simulating the four-multiplier NCR design further showed the benefits of reducing supply voltages in terms of power. It was possible for the NCR design to consume far less power and maintain performance. The supply voltage parameter settings that accomplished this have

been made bold in Table 3. Using a 1.00V V_{local} and V_{mux} while maintaining the 1.2V supply on all other circuit elements decreased the period by 0.01ns and the energy per operation by 6.02μJ. The decreases correspond to a performance increase of approximately 0.02% and an energy decrease of approximately 24.8%. Savings such as this could greatly benefit situations where circuits require lower power to operate. It was observed that the lower MUX supply voltage produced the same lower voltage at the output compared to the original design.

Power Calculation and Delay										
Design	V_{global} (V)	I_{global} (μA)	V_{local} (V)	I_{local} (μA)	V_{mux} (V)	I_{mux} (μA)	V_{sel} (V)	I_{sel} (μA)	Period (ns)	Energy / Op (μJ)
Single Multiplier	1.20	475.1							4.25	24.25
NCR Design	1.20	51.65	1.20	606.4	1.20	5.17	1.20	6.69	3.01	24.23
	1.20	46.12	1.10	485.5	1.20	4.63	1.20	6.04	3.60	21.69
	1.20	45.76	1.10	483.8	1.10	4.12	1.20	6.03	3.61	21.65
	1.20	45.52	1.10	479.8	1.10	4.10	1.10	5.27	3.64	21.60
	1.20	**39.80**	**1.01**	**378.1**	**1.01**	**3.18**	**1.01**	**4.10**	**4.24**	**18.51**
	1.20	**40.19**	**1.00**	**374.8**	**1.20**	**4.26**	**1.20**	**5.16**	**4.21**	**18.29**
	1.20	**39.48**	**1.00**	**372.8**	**1.00**	**3.13**	**1.20**	**5.17**	**4.24**	**18.23**
	1.20	38.94	1.00	366.0	1.00	3.11	1.00	3.93	4.31	18.11
	1.20	39.32	0.99	363.7	1.20	4.32	1.20	5.05	4.29	17.94
	1.20	38.99	0.99	360.9	0.99	3.05	1.20	4.95	4.32	17.85

Table 3. Four-multiplier design results.

4. CONCLUSIONS

Although it was impossible to reduce the power and maintain the performance of the initial one-multiplier NCR design, it was possible to greatly reduce the power while maintaining performance of additional circuits by scaling the supply voltage. This power reduction was demonstrated by enlarging the duplicated circuit. By stringing together two-multiplier and four-multiplier NCR designs and performing transistor-level simulations in Cadence to calculate power, it was clearly seen that the power reduction possible greatly increases as the duplicated circuit size increases. The decrease in power consumption occurred because the lesser supply voltage was distributed over a larger portion of the entire NCR design. The preferred design has a reduced supply voltage connected to only the duplicated circuit; this connection will ensure that the outputs are at the nominal supply voltage level and are therefore equivalent to the original design.

The technique of applying the NCR architecture to a circuit and then reducing the supply voltage to the duplicated circuits could be extremely useful in reducing the power of large circuits. As circuit size increases, the benefits of this technique increase rapidly. The supply voltage levels and components thereby supplied can be fine-tuned to produce a circuit with the exact same performance as the individual circuit with far less power usage.

This method of lowering power could tremendously increase the benefits seen by circuits designed at the University of Arkansas especially if power consumption is the primary concern. Since the technique is easy to implement in asynchronous designs, it could also be applied to any previously designed circuits to reduce power provided that the circuit is large enough to benefit.

For future work, this technique could be applied to different-sized circuits more extensively to determine the exact benefits of size. Other methods of reducing power could also

be investigated in parallel. These include altering the threshold voltages of the transistors, applying the global supply voltage to the critical path of the duplicated circuit while further lowering the local supply voltage, or transistor reordering.

REFERENCES

[1] S. C. Smith, "Speedup of NULL convention digital circuits using NULL cycle reduction," *Journal of Systems Architecture*, vol. 52, pp. 411-422, 2006.

[2] C.L. Seitz, *System timing.*: Addison-Wesley, 1980.

[3] A.J. Martin, "Programming in VLSI," in *Development in Concurrency and Communication.*: Addison-Wesley, 1990, pp. 1–64.

[4] D.E. Muller, "Asynchronous logics and application to information processing," in *Switching Theory in Space Technology.*: Stanford University Press, 1963, pp. 289–297.

[5] K.M. Fant and S.A. Brandt, "NULL convention logic: a complete and consistent logic for asynchronous digital circuit synthesis," in *International Conference on Application Specific Systems, Architectures, and Processors*, 1996, pp. 261-273.

[6] T. Verhoff, "Delay-insensitive codes—an overview," *Distributed Computing*, vol. 3, pp. 1-8, 1988.

[7] I. David, R. Ginosaur, and M. Yoeli, "An efficient implementation of boolean functions as self-timed circuits," *IEEE Transactions on Computers*, vol. 41, no. 1, pp. 2-10, 1996.

[8] G.E. Sobelman and K.M. Fant, "CMOS circuit design of threshold gates with hysteresis," in *IEEE International Symposium on Circuits and Systems*, vol. II, 1998, pp. 61-65.

[9] S.C. Smith, "Completion-completeness for NULL convention digital circuits utilizing the bit-wise completion strategy," in *The 2003 International Conference on VLSI*, 2003, pp. 143-149.

[10] A. Kondratyev, L. Neukom, O. Roig, A. Taubin, and K. Fant, "Checking delay-insensitivity:

10^4 gates and beyond," in *Eighth International Symposium on Asynchronous Circuits and Systems*, 2002, pp. 137-145.

[11] S.C. Smith, R.F. DeMara, J.S. Yuan, D. Ferguson, and D. Lamb, "Optimization of NULL convention self-timed circuits," *Integration, The VLSI Journal*, vol. 37, no. 3, pp. 135-165, 2004.

[12] S.C. Smith, R.F. DeMara, M. Hagedorn, and D. Ferguson, "Delay-insensitive gate-level pipelining," *Integration, The VLSI Journal*, vol. 30, no. 2, pp. 103-131, 2001.

[13] Scott C. Smith. (2011, March) Dr. Scott C. Smith: Projects. [Online]. http://comp.uark.edu/~smithsco/CCLI_async.html

A. SINGLE MULTIPLIER VHDL FILES

A.1 demux.vhd

```
library IEEE;
use ieee.std_logic_1164.all;
use work.ncl_signals.all;

entity demux is
  port (a: IN dual_rail_logic;
        rst, ki1, ki2, s1, s2: IN std_logic;
      z1, z2: OUT dual_rail_logic;
        ko: OUT std_logic);
end demux;

architecture arch of demux is
      signal t1, t2: dual_rail_logic;

      component th33nx0
        port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            rst: in std_logic;
            z: out std_logic);
      end component;

      component th14bx0
        port(a: in std_logic;
            b: in std_logic;
            c: in std_logic;
            d: in std_logic;
            zb: out std_logic);
      end component;

begin
  i11: th33nx0
      port map(a.rail1, s1, ki1, rst, t1.rail1);

  i10: th33nx0
      port map(a.rail0, s1, ki1, rst, t1.rail0);
      z1 <= t1;

  i21: th33nx0
      port map(a.rail1, s2, ki2, rst, t2.rail1);
```

```
  i20: th33nx0
     port map(a.rail0, s2, ki2, rst, t2.rail0);
     z2 <= t2;

  k0: th14bx0
     port map(t1.rail1, t1.rail0, t2.rail1, t2.rail0, ko);
--     ko.rail0 <= '0';

end arch;
```

A.2 demux_gen.vhd

```
library IEEE;
use ieee.std_logic_1164.all;
use work.ncl_signals.all;

entity dmux is
  generic(width: in integer := 1);
  port(a: IN dual_rail_logic_vector(width-1 downto 0);
       rst, ki1, ki2, s1, s2: IN std_logic;
     z1, z2: OUT dual_rail_logic_vector(width-1 downto 0);
       ko: OUT std_logic_vector(width-1 downto 0));
end dmux;

architecture arch of dmux is

  component demux
       port (a: IN dual_rail_logic;
       rst, ki1, ki2, s1, s2: IN std_logic;
       z1, z2: OUT dual_rail_logic;
              ko: OUT std_logic);
  end component;

begin
       struct: for i in a'range generate
              comp: demux
                     port map(a(i), rst, ki1, ki2, s1, s2, z1(i), z2(i), ko(i));
       end generate struct;

end arch;
```

A.3 mult4x4_1stage.vhd

```
library ieee;
use ieee.std_logic_1164.all;
```

```
use work.ncl_signals.all;
use work.dual_rail.all;

entity mult4x4_1n is
   port(x, y: in dual_rail_logic_VECTOR(3 downto 0);
        ki, reset: in std_logic;
        s: out dual_rail_logic_VECTOR(7 downto 0);
        ko: out std_logic);
end;

architecture BEHAVIOR of mult4x4_1n is
   signal pp1, pp2, pp3, pp4, pp5, pp6, pp7, pp8, pp9: dual_rail_logic;
   signal pp10, pp11, pp12, pp13, pp14, pp15: dual_rail_logic;
   signal c1_1, c1_2, c1_3, c1_4, c1_5: dual_rail_logic;
   signal s1_0, s1_1, s1_2, s1_3, s1_4, s1_5: dual_rail_logic;
   signal c2_3, c2_4, c2_5, c2_6, c2_7: dual_rail_logic;
   signal s2_2, s2_3, s2_4, s2_5, s2_6, s3_3: dual_rail_logic;
   signal c3_4, s3_4, c3_5: dual_rail_logic;
   signal x_o, y_o: dual_rail_logic_VECTOR(3 downto 0);
   signal temp0, temp0_in, temp0_out: dual_rail_logic_VECTOR(7 downto 0);
   signal ki_0, ko_0: std_logic_VECTOR(7 downto 0);
   signal temp1, temp1_in, temp1_out: dual_rail_logic_VECTOR(15 downto 0);
   signal ki_1, ko_1: std_logic_VECTOR(15 downto 0);
   signal temp2, temp2_in, temp2_out: dual_rail_logic_VECTOR(12 downto 0);
   signal ki_2, ko_2: std_logic_VECTOR(12 downto 0);
   signal temp3, temp3_in, temp3_out: dual_rail_logic_VECTOR(11 downto 0);
   signal ki_3, ko_3: std_logic_VECTOR(11 downto 0);
   signal temp4, temp4_in, temp4_out: dual_rail_logic_VECTOR(11 downto 0);
   signal ki_4, ko_4: std_logic_VECTOR(11 downto 0);
   signal temp5, temp5_in, temp5_out: dual_rail_logic_VECTOR(10 downto 0);
   signal ki_5, ko_5: std_logic_VECTOR(10 downto 0);
   signal temp6, temp6_in, temp6_out: dual_rail_logic_VECTOR(9 downto 0);
   signal ki_6, ko_6: std_logic_VECTOR(9 downto 0);
   signal temp7, temp7_in: dual_rail_logic_VECTOR(7 downto 0);
   signal ki_7, ko_7: std_logic_VECTOR(7 downto 0);
   signal ki0, ki1, ki2, ki3, ki4, ki5, ki6: std_logic;
   signal pp15_o, pp14_o, pp13_o, pp12_o, pp11_o, pp10_o: dual_rail_logic;
   signal pp9_o, pp8_o, pp7_o, pp6_o, pp5_o, pp4_o, pp3_o, pp2_o, pp1_o, s1_0_o:
dual_rail_logic;
   signal pp15o, c1_5o, s1_5o, c1_4o, s1_4o, c1_3o, pp12o, s1_3o, c1_2o, s1_2o, c1_1o,
s1_1o, s1_0o: dual_rail_logic;
   signal c2_7o, s2_6o, c2_6o, s2_5o, c2_5o, s2_4o, c2_4o, c2_3o, s2_3o, s2_2o, s2_1o,
s2_0o: dual_rail_logic;
   signal c2_7_o, s2_6_o, c2_6_o, s2_5_o, c2_5_o, s2_4_o, c2_4_o, c3_4_o, s3_3_o,
s2_2_o, s2_1_o, s2_0_o: dual_rail_logic;
```

```
        signal c2_7_o1, s2_6_o1, c2_6_o1, s2_5_o1, c2_5_o1, c3_5o, s4_4, s4_3, s4_2, s4_1,
s4_0: dual_rail_logic;
        signal c2_7_o2, s2_6_o2, c2_6_o2, c4_6o, s5_5, s5_4, s5_3, s5_2, s5_1, s5_0:
dual_rail_logic;
        signal s4_7, c4_7, s4_6, c4_6, s4_5: dual_rail_logic;

        component full_add
            port(c_in, x, y: in dual_rail_logic;
              c_out, s: out dual_rail_logic);
        end component;

        component half_add
            port(x, y: in dual_rail_logic;
              c_out, s: out dual_rail_logic);
        end component;

        component ncl_register
            generic(width: in integer;
                    initial_value: in integer);
            port(data_in: in dual_rail_logic_VECTOR(width - 1 downto 0);
              ki: in std_logic_VECTOR(width - 1 downto 0);
              rst: in std_logic;
              data_out: out dual_rail_logic_VECTOR(width - 1 downto 0);
              ko: out std_logic_VECTOR(width - 1 downto 0));
        end component;

        component comp8a
            port(a: in std_logic_VECTOR(7 downto 0);
              z: out std_logic);
        end component;

        component and2
            port(a, b: in dual_rail_logic;
              z: out dual_rail_logic);
        end component;

        component and2i
          port(a, b: in dual_rail_logic;
            z: out dual_rail_logic);
        end component;

        component gens7
            port(c, x, y, z: in dual_rail_logic;
              s: out dual_rail_logic);
        end component;
```

```
begin
  temp0_in <= x & y;

  COMP0: comp8a
       port map(ko_0, ko);

  REG0: ncl_register
       generic map(8, 2)
       port map(temp0_in, ki_0, reset, temp0_out, ko_0);
  ki_0(7) <= ki0;
  ki_0(6) <= ki0;
  ki_0(5) <= ki0;
  ki_0(4) <= ki0;
  ki_0(3) <= ki0;
  ki_0(2) <= ki0;
  ki_0(1) <= ki0;
  ki_0(0) <= ki0;
  x_o <= temp0_out(7 downto 4);
  y_o <= temp0_out(3 downto 0);


  GEN_S0: and2
       port map(y_o(0), x_o(0), s1_0);

  GEN_PP1: and2i
       port map(y_o(0), x_o(1), pp1);

  GEN_PP2: and2i
       port map(y_o(0), x_o(2), pp2);

  GEN_PP3: and2i
       port map(y_o(0), x_o(3), pp3);

  GEN_PP4: and2i
       port map(y_o(1), x_o(0), pp4);

  GEN_PP5: and2
       port map(y_o(1), x_o(1), pp5);

  GEN_PP6: and2i
       port map(y_o(1), x_o(2), pp6);

  GEN_PP7: and2i
       port map(y_o(1), x_o(3), pp7);

  GEN_PP8: and2i
```

```
        port map(y_o(2), x_o(0), pp8);

    GEN_PP9: and2i
        port map(y_o(2), x_o(1), pp9);

    GEN_PP10: and2
        port map(y_o(2), x_o(2), pp10);

    GEN_PP11: and2i
        port map(y_o(2), x_o(3), pp11);

    GEN_PP12: and2i
        port map(y_o(3), x_o(0), pp12);

    GEN_PP13: and2i
        port map(y_o(3), x_o(1), pp13);

    GEN_PP14: and2i
        port map(y_o(3), x_o(2), pp14);

    GEN_PP15: and2
        port map(y_o(3), x_o(3), pp15);


    temp1_out <= pp15 & pp14 & pp13 & pp12 & pp11 & pp10 & pp9 & pp8 & pp7 &
pp6 & pp5 & pp4 & pp3 & pp2 & pp1 & s1_0;


    pp15_o <= temp1_out(15);
    pp14_o <= temp1_out(14);
    pp13_o <= temp1_out(13);
    pp12_o <= temp1_out(12);
    pp11_o <= temp1_out(11);
    pp10_o <= temp1_out(10);
    pp9_o <= temp1_out(9);
    pp8_o <= temp1_out(8);
    pp7_o <= temp1_out(7);
    pp6_o <= temp1_out(6);
    pp5_o <= temp1_out(5);
    pp4_o <= temp1_out(4);
    pp3_o <= temp1_out(3);
    pp2_o <= temp1_out(2);
    pp1_o <= temp1_out(1);
    s1_0_o <= temp1_out(0);
```

```
HA1_1: half_add
    port map(pp1_o, pp4_o, c1_1, s1_1);

FA1_2: full_add
    port map(pp2_o, pp5_o, pp8_o, c1_2, s1_2);

FA1_3: full_add
    port map(pp3_o, pp6_o, pp9_o, c1_3, s1_3);

FA1_4: full_add
    port map(pp7_o, pp10_o, pp13_o, c1_4, s1_4);

HA1_5: half_add
    port map(pp11_o, pp14_o, c1_5, s1_5);

temp2_out <= pp15_o & c1_5 & s1_5 & c1_4 & s1_4 & c1_3 & pp12_o & s1_3 &
c1_2 & s1_2 & c1_1 & s1_1 & s1_0_o;


pp15o <= temp2_out(12);
c1_5o <= temp2_out(11);
s1_5o <= temp2_out(10);
c1_4o <= temp2_out(9);
s1_4o <= temp2_out(8);
c1_3o <= temp2_out(7);
pp12o <= temp2_out(6);
s1_3o <= temp2_out(5);
c1_2o <= temp2_out(4);
s1_2o <= temp2_out(3);
c1_1o <= temp2_out(2);
s1_1o <= temp2_out(1);
s1_0o <= temp2_out(0);


HA2_2: half_add
    port map(c1_1o, s1_2o, c2_3, s2_2);

FA2_3: full_add
    port map(pp12o, c1_2o, s1_3o, c2_4, s2_3);

HA2_4: half_add
    port map(c1_3o, s1_4o, c2_5, s2_4);

HA2_5: half_add
    port map(c1_4o, s1_5o, c2_6, s2_5);
```

```
HA2_6: half_add
    port map(pp15o, c1_5o, c2_7, s2_6);


temp3_out <= c2_7 & s2_6 & c2_6 & s2_5 & c2_5 & s2_4 & c2_4 & c2_3 & s2_3 &
s2_2 & s1_1o & s1_0o;

c2_7o <= temp3_out(11);
s2_6o <= temp3_out(10);
c2_6o <= temp3_out(9);
s2_5o <= temp3_out(8);
c2_5o <= temp3_out(7);
s2_4o <= temp3_out(6);
c2_4o <= temp3_out(5);
c2_3o <= temp3_out(4);
s2_3o <= temp3_out(3);
s2_2o <= temp3_out(2);
s2_1o <= temp3_out(1);
s2_0o <= temp3_out(0);


HA3_3: half_add
    port map(c2_3o, s2_3o, c3_4, s3_3);

temp4_out <= c2_7o & s2_6o & c2_6o & s2_5o & c2_5o & s2_4o & c2_4o & c3_4 &
s3_3 & s2_2o & s2_1o & s2_0o;


c2_7_o <= temp4_out(11);
s2_6_o <= temp4_out(10);
c2_6_o <= temp4_out(9);
s2_5_o <= temp4_out(8);
c2_5_o <= temp4_out(7);
s2_4_o <= temp4_out(6);
c2_4_o <= temp4_out(5);
c3_4_o <= temp4_out(4);
s3_3_o <= temp4_out(3);
s2_2_o <= temp4_out(2);
s2_1_o <= temp4_out(1);
s2_0_o <= temp4_out(0);


FA4_4: full_add
  port map(s2_4_o, c2_4_o, c3_4_o, c3_5, s3_4);
```

```
temp5_out <= c2_7_o & s2_6_o & c2_6_o & s2_5_o & c2_5_o & c3_5 & s3_4 &
s3_3_o & s2_2_o & s2_1_o & s2_0_o;


c2_7_o1 <= temp5_out(10);
s2_6_o1 <= temp5_out(9);
c2_6_o1 <= temp5_out(8);
s2_5_o1 <= temp5_out(7);
c2_5_o1 <= temp5_out(6);
c3_5o <= temp5_out(5);
s4_4 <= temp5_out(4);
s4_3 <= temp5_out(3);
s4_2 <= temp5_out(2);
s4_1 <= temp5_out(1);
s4_0 <= temp5_out(0);


FA4_5: full_add
    port map(c3_5o, c2_5_o1, s2_5_o1, c4_6, s4_5);


temp6_out <= c2_7_o1 & s2_6_o1 & c2_6_o1 & c4_6 & s4_5 & s4_4 & s4_3 & s4_2
& s4_1 & s4_0;


c2_7_o2 <= temp6_out(9);
s2_6_o2 <= temp6_out(8);
c2_6_o2 <= temp6_out(7);
c4_6o <= temp6_out(6);
s5_5 <= temp6_out(5);
s5_4 <= temp6_out(4);
s5_3 <= temp6_out(3);
s5_2 <= temp6_out(2);
s5_1 <= temp6_out(1);
s5_0 <= temp6_out(0);


FA4_6: full_add
    port map(c4_6o, c2_6_o2, s2_6_o2, open, s4_6);

g4_1_S7: gens7
    port map(c2_7_o2, s2_6_o2, c2_6_o2, c4_6o, s4_7);

temp7_in <= s4_7 & s4_6 & s5_5 & s5_4 & s5_3 & s5_2 & s5_1 & s5_0;
```

```
    COMP7: comp8a
        port map(ko_7, ki0);

    REG7: ncl_register
        generic map(8, 2)
        port map(temp7_in, ki_7, reset, s, ko_7);
    ki_7(7) <= ki;
    ki_7(6) <= ki;
    ki_7(5) <= ki;
    ki_7(4) <= ki;
    ki_7(3) <= ki;
    ki_7(2) <= ki;
    ki_7(1) <= ki;
    ki_7(0) <= ki;
end BEHAVIOR;
```

A.4 mult4x4_rnc.vhd

```
        s: out dual_rail_logic_vector (7 downto 0);
        ko: out std_logic);
end mult4x4;

architecture BEHAVIOR of mult4x4 is
        signal data_in, di1, di2, do1, do2: dual_rail_logic_vector(7 downto 0);
        signal kod: std_logic_vector(7 downto 0);
        signal ko1, ko2, ki1, ki2, kot, s1, s2: std_logic;

    component mult4x4_1n
        port(x, y: IN dual_rail_logic_vector (3 downto 0);
                ki, reset: IN std_logic;
                s: OUT dual_rail_logic_vector (7 downto 0);
                ko: OUT std_logic);
    end component;

    component dmux
      generic(width: in integer := 1);
      port(a: IN dual_rail_logic_vector(width-1 downto 0);
        rst, ki1, ki2: IN std_logic;
        s1, s2: IN std_logic;
        z1, z2: OUT dual_rail_logic_vector(width-1 downto 0);
                ko: OUT std_logic_vector(width-1 downto 0));
    end component;

    component mux
        generic(width: in integer := 1);
```

```
        port(a1, a2: IN dual_rail_logic_vector(width-1 downto 0);
        z: OUT dual_rail_logic_vector(width-1 downto 0));
    end component;

    component comp8a
      port(a: IN std_logic_vector(7 downto 0);
           z: OUT std_logic);
    end component;

    component selct
        port (ki, rst: IN std_logic;
        s1, s2: OUT std_logic);
    end component;

begin
        data_in <= x & y;

        DEMUX_INPUT: dmux
                generic map(8)
                port map(data_in, reset, ko1, ko2, s1, s2, di1, di2, kod);

        COMP: comp8a
                port map(kod, kot);
                ko <= kot;

        SELECT_INPUT: selct
                port map(kot, reset, s1, s2);

        COMB1: mult4x4_1n
                port map(di1(7 downto 4), di1(3 downto 0), ki1, reset, do1, ko1);

    COMB2: mult4x4_1n
      port map(di2(7 downto 4), di2(3 downto 0), ki2, reset, do2, ko2);

        MUX_OUTPUT: mux
                generic map(8)
                port map(do1, do2, s);

        SELECT_OUTPUT: selct
          port map(ki, reset, ki1, ki2);
--              ki1.rail0 <= '0';
--              ki2.rail0 <= '0';

end BEHAVIOR;
```

A.5 mux_gen.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use work.ncl_signals.all;

entity mux is
  generic(width: in integer := 1);
  port(a1, a2: IN dual_rail_logic_vector(width-1 downto 0);
      z: OUT dual_rail_logic_vector(width-1 downto 0));
end mux;

architecture arch of mux is

  component th12x0
        port (a: IN std_logic;
        b: IN std_logic;
        z: OUT std_logic);
  end component;

begin
        struct: for i in a1'range generate
                comp0: th12x0
                        port map(a1(i).rail0, a2(i).rail0, z(i).rail0);

                comp1: th12x0
                port map(a1(i).rail1, a2(i).rail1, z(i).rail1);
        end generate struct;

end arch;
```

A.6 select.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity selct is
  port (ki, rst: IN std_logic;
       s1, s2: OUT std_logic);
end selct;

architecture arch of selct is
        signal d0, d1, d2, d3, r0, r1, r2, r3: std_logic;

      component th33nx0
```

```
        port(a: in std_logic;
             b: in std_logic;
             c: in std_logic;
             rst: in std_logic;
             z: out std_logic);
      end component;

      component th33dx0
        port(a: in std_logic;
             b: in std_logic;
             c: in std_logic;
             rst: in std_logic;
             z: out std_logic);
      end component;

        component invx0
       port(i: in std_logic;
             zb: out std_logic);
      end component;
begin

        g0: th33nx0
                port map(ki, d3, r1, rst, d0);

      g1: th33dx0
            port map(ki, d0, r2, rst, d1);

      g2: th33nx0
            port map(ki, d1, r3, rst, d2);

      g3: th33nx0
            port map(ki, d2, r0, rst, d3);

      i0: invx0
            port map(d0, r0);

      i1: invx0
            port map(d1, r1);

      i2: invx0
            port map(d2, r2);

      i3: invx0
            port map(d3, r3);

        s1 <= d2;
```

```
            s2 <= d0;
        end arch;
```

A.7 tb_mult4x4_full.vhd

```
        Library IEEE;
        use IEEE.std_logic_1164.all;
        use IEEE.std_logic_unsigned.all;
        use work.ncl_signals.all;
        use work.dual_rail.all;

        use work.functions.all;
        use std.textio.all;

        entity TB_MULT4x4 is
        end;


        architecture TESTBENCH of TB_MULT4x4 is
                signal x, y, x_temp, x_next, y_temp, y_next: DUAL_RAIL_LOGIC_VECTOR(3
downto 0);
              signal s: DUAL_RAIL_LOGIC_VECTOR(7 downto 0);
              signal xy_calc: std_logic_vector(7 downto 0) := "00000000";
                signal ki, ko, reset: STD_LOGIC;
                signal cx, cy: DUAL_RAIL_LOGIC_VECTOR(3 downto 2);
                signal incorrect: std_logic := '0';

                type output_array is array(0 to 256) of std_logic_vector(7 downto 0);
              signal s_calc_array: output_array;

                component mult4x4 -- 1n
                        port(x, y: in DUAL_RAIL_LOGIC_VECTOR(3 downto 0);
                                ki, reset: in STD_LOGIC;
                                s: out DUAL_RAIL_LOGIC_VECTOR(7 downto 0);
                                ko: out STD_LOGIC);
                end component;

        begin
                UUT: MULT4x4 -- 1n
                        port map(x, y, ki, reset, s, ko);

                CALC_ANSWER: process
                begin
                        for i in 0 to 256 loop
                                s_calc_array(i) <= xy_calc(7 downto 4) * xy_calc(3 downto 0);
                                xy_calc <= xy_calc + '1';
```

```
                wait for 0 ns;
        end loop;
        wait;
end process;


INPUTS: process
begin
        --reset <= '0';
        reset <= '1';
        wait until ko'event and ko = '1';
        reset <= '0';

   x(0).rail0 <= '1';
   x(0).rail1 <= '0';
   x(1).rail0 <= '1';
   x(1).rail1 <= '0';
   x(2).rail0 <= '1';
   x(2).rail1 <= '0';
   x(3).rail0 <= '1';
   x(3).rail1 <= '0';
   y(0).rail0 <= '1';
   y(0).rail1 <= '0';
   y(1).rail0 <= '1';
   y(1).rail1 <= '0';
   y(2).rail0 <= '1';
   y(2).rail1 <= '0';
   y(3).rail0 <= '1';
   y(3).rail1 <= '0';

        wait for 0 ns;

        while (x(3).rail1 = '0' or x(2).rail1 = '0' or x(1).rail1 = '0' or x(0).rail1 = '0')
loop
                wait until ko'event and ko = '0';

                cy(2) <= y(0) and y(1);
                cy(3) <= y(0) and y(1) and y(2);

                cx(2) <= x(0) and x(1);
        cx(3) <= x(0) and x(1) and x(2);
                wait for 0 ns;
                y_temp <= y;
                y_next(0) <= not(y(0));
                y_next(1) <= y(1) xor y(0);
                y_next(2) <= cy(2) xor y(2);
```

```
                y_next(3) <= cy(3) xor y(3);

        x_temp <= x;
        x_next(0) <= not(x(0));
        x_next(1) <= x(1) xor x(0);
        x_next(2) <= cx(2) xor x(2);
        x_next(3) <= cx(3) xor x(3);

        x(0).rail0 <= '0';
        x(0).rail1 <= '0';
        x(1).rail0 <= '0';
        x(1).rail1 <= '0';
        x(2).rail0 <= '0';
        x(2).rail1 <= '0';
        x(3).rail0 <= '0';
        x(3).rail1 <= '0';
        y(0).rail0 <= '0';
        y(0).rail1 <= '0';
        y(1).rail0 <= '0';
        y(1).rail1 <= '0';
        y(2).rail0 <= '0';
        y(2).rail1 <= '0';
        y(3).rail0 <= '0';
        y(3).rail1 <= '0';
                wait until ko'event and ko = '1';
                if (y_temp(3).rail1 = '1' and y_temp(2).rail1 = '1' and
y_temp(1).rail1 = '1' and y_temp(0).rail1 = '1') then
                        x <= x_next;
                else
                        x <= x_temp;
                end if;
                y <= y_next;
                wait for 0 ns;

        end loop;
        while (x(3).rail1 = '1' or x(2).rail1 = '1' or x(1).rail1 = '1' or x(0).rail1 = '1')
loop
        wait until ko'event and ko = '0';

        cy(2) <= y(0) and y(1);
        cy(3) <= y(0) and y(1) and y(2);

        cx(2) <= x(0) and x(1);
        cx(3) <= x(0) and x(1) and x(2);
        wait for 0 ns;
        y_temp <= y;
```

```
                    y_next(0) <= not(y(0));
                    y_next(1) <= y(1) xor y(0);
                    y_next(2) <= cy(2) xor y(2);
                    y_next(3) <= cy(3) xor y(3);

                    x_temp <= x;
                    x_next(0) <= not(x(0));
                    x_next(1) <= x(1) xor x(0);
                    x_next(2) <= cx(2) xor x(2);
                    x_next(3) <= cx(3) xor x(3);

                    x(0).rail0 <= '0';
                    x(0).rail1 <= '0';
                    x(1).rail0 <= '0';
                    x(1).rail1 <= '0';
                    x(2).rail0 <= '0';
                    x(2).rail1 <= '0';
                    x(3).rail0 <= '0';
                    x(3).rail1 <= '0';
                    y(0).rail0 <= '0';
                    y(0).rail1 <= '0';
                    y(1).rail0 <= '0';
                    y(1).rail1 <= '0';
                    y(2).rail0 <= '0';
                    y(2).rail1 <= '0';
                    y(3).rail0 <= '0';
                    y(3).rail1 <= '0';
                    wait until ko'event and ko = '1';
                    if (y_temp(3).rail1 = '1' and y_temp(2).rail1 = '1' and y_temp(1).rail1 = '1'
and y_temp(0).rail1 = '1') then
                        x <= x_next;
                    else
                        x <= x_temp;
                    end if;
                    y <= y_next;
                        wait for 0 ns;

                end loop;
                wait;
            end process;

            OUTPUTS: process
            variable ss: side;
            variable bw: width := 6;
            variable l: line;
                    variable tm: time;
```

```
                variable once: bit := '0';
--      file t: text is out "/home/bsparkma/Senior_Thesis/ModelSim/t.txt";
                variable j: integer := 0;

    begin
--              ki.rail0 <= '0';
                while (true) loop
                        ki <= '1';

end TESTBENCH;


configuration CFG_TB_MULT4x4 of TB_MULT4x4 is
        for TESTBENCH
                for UUT: MULT4x4 --_1n
                end for;{Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.ncl_signals.all;
use work.dual_rail.all;

use work.functions.all;
use std.textio.all;

entity TB_MULT4x4 is
end;


architecture TESTBENCH of TB_MULT4x4 is
        signal x, y, x_temp, x_next, y_temp, y_next: DUAL_RAIL_LOGIC_VECTOR(3
downto 0);
      signal s: DUAL_RAIL_LOGIC_VECTOR(7 downto 0);
      signal xy_calc: std_logic_vector(7 downto 0) := "00000000";
        signal ki, ko, reset: STD_LOGIC;
        signal cx, cy: DUAL_RAIL_LOGIC_VECTOR(3 downto 2);
        signal incorrect: std_logic := '0';

        type output_array is array(0 to 256) of std_logic_vector(7 downto 0);
      signal s_calc_array: output_array;

        component mult4x4 --_1n
                port(x, y: in DUAL_RAIL_LOGIC_VECTOR(3 downto 0);
                        ki, reset: in STD_LOGIC;
                        s: out DUAL_RAIL_LOGIC_VECTOR(7 downto 0);
                        ko: out STD_LOGIC);
        end component;
```

```
begin
        UUT: MULT4x4 --_1n
                port map(x, y, ki, reset, s, ko);

        CALC_ANSWER: process
        begin
                for i in 0 to 256 loop
                        s_calc_array(i) <= xy_calc(7 downto 4) * xy_calc(3 downto 0);
                        xy_calc <= xy_calc + '1';
                        wait for 0 ns;
                end loop;
                wait;
        end process;


        INPUTS: process
        begin
                --reset <= '0';
                reset <= '1';
                wait until ko'event and ko = '1';
                reset <= '0';

          x(0).rail0 <= '1';
          x(0).rail1 <= '0';
          x(1).rail0 <= '1';
          x(1).rail1 <= '0';
          x(2).rail0 <= '1';
          x(2).rail1 <= '0';
          x(3).rail0 <= '1';
          x(3).rail1 <= '0';
          y(0).rail0 <= '1';
          y(0).rail1 <= '0';
          y(1).rail0 <= '1';
          y(1).rail1 <= '0';
          y(2).rail0 <= '1';
          y(2).rail1 <= '0';
          y(3).rail0 <= '1';
          y(3).rail1 <= '0';

                wait for 0 ns;

                while (x(3).rail1 = '0' or x(2).rail1 = '0' or x(1).rail1 = '0' or x(0).rail1 = '0')
loop
                        wait until ko'event and ko = '0';
```

```
        cy(2) <= y(0) and y(1);
        cy(3) <= y(0) and y(1) and y(2);

        cx(2) <= x(0) and x(1);
    cx(3) <= x(0) and x(1) and x(2);
        wait for 0 ns;
        y_temp <= y;
        y_next(0) <= not(y(0));
        y_next(1) <= y(1) xor y(0);
        y_next(2) <= cy(2) xor y(2);
        y_next(3) <= cy(3) xor y(3);

    x_temp <= x;
    x_next(0) <= not(x(0));
    x_next(1) <= x(1) xor x(0);
    x_next(2) <= cx(2) xor x(2);
    x_next(3) <= cx(3) xor x(3);

    x(0).rail0 <= '0';
    x(0).rail1 <= '0';
    x(1).rail0 <= '0';
    x(1).rail1 <= '0';
    x(2).rail0 <= '0';
    x(2).rail1 <= '0';
    x(3).rail0 <= '0';
    x(3).rail1 <= '0';
    y(0).rail0 <= '0';
    y(0).rail1 <= '0';
    y(1).rail0 <= '0';
    y(1).rail1 <= '0';
    y(2).rail0 <= '0';
    y(2).rail1 <= '0';
    y(3).rail0 <= '0';
    y(3).rail1 <= '0';
        wait until ko'event and ko = '1';
        if (y_temp(3).rail1 = '1' and y_temp(2).rail1 = '1' and
y_temp(1).rail1 = '1' and y_temp(0).rail1 = '1') then
            x <= x_next;
        else
            x <= x_temp;
        end if;
        y <= y_next;
        wait for 0 ns;

    end loop;
```

```
                while (x(3).rail1 = '1' or x(2).rail1 = '1' or x(1).rail1 = '1' or x(0).rail1 = '1')
loop
                wait until ko'event and ko = '0';

                cy(2) <= y(0) and y(1);
                cy(3) <= y(0) and y(1) and y(2);

                cx(2) <= x(0) and x(1);
                cx(3) <= x(0) and x(1) and x(2);
                wait for 0 ns;
                y_temp <= y;
                y_next(0) <= not(y(0));
                y_next(1) <= y(1) xor y(0);
                y_next(2) <= cy(2) xor y(2);
                y_next(3) <= cy(3) xor y(3);

                x_temp <= x;
                x_next(0) <= not(x(0));
                x_next(1) <= x(1) xor x(0);
                x_next(2) <= cx(2) xor x(2);
                x_next(3) <= cx(3) xor x(3);

                x(0).rail0 <= '0';
                x(0).rail1 <= '0';
                x(1).rail0 <= '0';
                x(1).rail1 <= '0';
                x(2).rail0 <= '0';
                x(2).rail1 <= '0';
                x(3).rail0 <= '0';
                x(3).rail1 <= '0';
                y(0).rail0 <= '0';
                y(0).rail1 <= '0';
                y(1).rail0 <= '0';
                y(1).rail1 <= '0';
                y(2).rail0 <= '0';
                y(2).rail1 <= '0';
                y(3).rail0 <= '0';
                y(3).rail1 <= '0';
                wait until ko'event and ko = '1';
                if (y_temp(3).rail1 = '1' and y_temp(2).rail1 = '1' and y_temp(1).rail1 = '1'
and y_temp(0).rail1 = '1') then
                        x <= x_next;
                else
                        x <= x_temp;
                end if;
                y <= y_next;
```

```
                    wait for 0 ns;

            end loop;
            wait;
        end process;

        OUTPUTS: process
        variable ss: side;
        variable bw: width := 6;
        variable l: line;
                variable tm: time;
                variable once: bit := '0';
--      file t: text is out "/home/bsparkma/Senior_Thesis/ModelSim/t.txt";
                variable j: integer := 0;

    begin
--              ki.rail0 <= '0';
                while (true) loop
                        ki <= '1';
                        wait until s'event and is_data(s);

                        for i in 0 to 7 loop
                                if (s(i).rail1 /= s_calc_array(j)(i)) then
                                        incorrect <= '1';
                                end if;
                        end loop;

                        if (once = '0') then
                                once := '1';
                        else
--                              write(l, now - tm, ss, bw);
--                              writeline(t, l);
                        end if;
                        tm := now;
                        ki <= '0';
                        wait until s'event and is_null(s);
                        j := j+1;
                end loop;
        end process;

end TESTBENCH;


configuration CFG_TB_MULT4x4 of TB_MULT4x4 is
        for TESTBENCH
                for UUT: MULT4x4 --_1n
```

```
                end for;}
        end for;
end;
                        wait until s'event and is_data(s);

                        for i in 0 to 7 loop
                                if (s(i).rail1 /= s_calc_array(j)(i)) then
                                        incorrect <= '1';
                                end if;
                        end loop;

                        if (once = '0') then
                                once := '1';
                        else
--                              write(l, now - tm, ss, bw);
--                              writeline(t, l);
                        end if;
                        tm := now;
                        ki <= '0';
                        wait until s'event and is_null(s);
                        j := j+1;
                end loop;
        end process;

end TESTBENCH;


configuration CFG_TB_MULT4x4 of TB_MULT4x4 is
        for TESTBENCH
                for UUT: MULT4x4 --_1n
                end for;{Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.ncl_signals.all;
use work.dual_rail.all;

use work.functions.all;
use std.textio.all;

entity TB_MULT4x4 is
end;


architecture TESTBENCH of TB_MULT4x4 is
        signal x, y, x_temp, x_next, y_temp, y_next: DUAL_RAIL_LOGIC_VECTOR(3
downto 0);
```

```
    signal s: DUAL_RAIL_LOGIC_VECTOR(7 downto 0);
    signal xy_calc: std_logic_vector(7 downto 0) := "00000000";
      signal ki, ko, reset: STD_LOGIC;
      signal cx, cy: DUAL_RAIL_LOGIC_VECTOR(3 downto 2);
      signal incorrect: std_logic := '0';

      type output_array is array(0 to 256) of std_logic_vector(7 downto 0);
    signal s_calc_array: output_array;

      component mult4x4 --_1n
            port(x, y: in DUAL_RAIL_LOGIC_VECTOR(3 downto 0);
                  ki, reset: in STD_LOGIC;
                  s: out DUAL_RAIL_LOGIC_VECTOR(7 downto 0);
                  ko: out STD_LOGIC);
      end component;

begin
      UUT: MULT4x4 --_1n
            port map(x, y, ki, reset, s, ko);

      CALC_ANSWER: process
      begin
            for i in 0 to 256 loop
                  s_calc_array(i) <= xy_calc(7 downto 4) * xy_calc(3 downto 0);
                  xy_calc <= xy_calc + '1';
                  wait for 0 ns;
            end loop;
            wait;
      end process;


      INPUTS: process
      begin
            --reset <= '0';
            reset <= '1';
            wait until ko'event and ko = '1';
            reset <= '0';

        x(0).rail0 <= '1';
        x(0).rail1 <= '0';
        x(1).rail0 <= '1';
        x(1).rail1 <= '0';
        x(2).rail0 <= '1';
        x(2).rail1 <= '0';
        x(3).rail0 <= '1';
        x(3).rail1 <= '0';
```

```
y(0).rail0 <= '1';
y(0).rail1 <= '0';
y(1).rail0 <= '1';
y(1).rail1 <= '0';
y(2).rail0 <= '1';
y(2).rail1 <= '0';
y(3).rail0 <= '1';
y(3).rail1 <= '0';

    wait for 0 ns;

    while (x(3).rail1 = '0' or x(2).rail1 = '0' or x(1).rail1 = '0' or x(0).rail1 = '0')
loop
        wait until ko'event and ko = '0';

        cy(2) <= y(0) and y(1);
        cy(3) <= y(0) and y(1) and y(2);

        cx(2) <= x(0) and x(1);
    cx(3) <= x(0) and x(1) and x(2);
        wait for 0 ns;
        y_temp <= y;
        y_next(0) <= not(y(0));
        y_next(1) <= y(1) xor y(0);
        y_next(2) <= cy(2) xor y(2);
        y_next(3) <= cy(3) xor y(3);

    x_temp <= x;
    x_next(0) <= not(x(0));
    x_next(1) <= x(1) xor x(0);
    x_next(2) <= cx(2) xor x(2);
    x_next(3) <= cx(3) xor x(3);

    x(0).rail0 <= '0';
    x(0).rail1 <= '0';
    x(1).rail0 <= '0';
    x(1).rail1 <= '0';
    x(2).rail0 <= '0';
    x(2).rail1 <= '0';
    x(3).rail0 <= '0';
    x(3).rail1 <= '0';
    y(0).rail0 <= '0';
    y(0).rail1 <= '0';
    y(1).rail0 <= '0';
    y(1).rail1 <= '0';
    y(2).rail0 <= '0';
```

```
y(2).rail1 <= '0';
y(3).rail0 <= '0';
y(3).rail1 <= '0';
        wait until ko'event and ko = '1';
        if (y_temp(3).rail1 = '1' and y_temp(2).rail1 = '1' and
y_temp(1).rail1 = '1' and y_temp(0).rail1 = '1') then
                x <= x_next;
        else
                x <= x_temp;
        end if;
        y <= y_next;
        wait for 0 ns;

end loop;
while (x(3).rail1 = '1' or x(2).rail1 = '1' or x(1).rail1 = '1' or x(0).rail1 = '1')
loop
wait until ko'event and ko = '0';

cy(2) <= y(0) and y(1);
cy(3) <= y(0) and y(1) and y(2);

cx(2) <= x(0) and x(1);
cx(3) <= x(0) and x(1) and x(2);
wait for 0 ns;
y_temp <= y;
y_next(0) <= not(y(0));
y_next(1) <= y(1) xor y(0);
y_next(2) <= cy(2) xor y(2);
y_next(3) <= cy(3) xor y(3);

x_temp <= x;
x_next(0) <= not(x(0));
x_next(1) <= x(1) xor x(0);
x_next(2) <= cx(2) xor x(2);
x_next(3) <= cx(3) xor x(3);

x(0).rail0 <= '0';
x(0).rail1 <= '0';
x(1).rail0 <= '0';
x(1).rail1 <= '0';
x(2).rail0 <= '0';
x(2).rail1 <= '0';
x(3).rail0 <= '0';
x(3).rail1 <= '0';
y(0).rail0 <= '0';
y(0).rail1 <= '0';
```

```
                    y(1).rail0 <= '0';
                    y(1).rail1 <= '0';
                    y(2).rail0 <= '0';
                    y(2).rail1 <= '0';
                    y(3).rail0 <= '0';
                    y(3).rail1 <= '0';
                    wait until ko'event and ko = '1';
                    if (y_temp(3).rail1 = '1' and y_temp(2).rail1 = '1' and y_temp(1).rail1 = '1'
and y_temp(0).rail1 = '1') then
                        x <= x_next;
                    else
                        x <= x_temp;
                    end if;
                    y <= y_next;
                    wait for 0 ns;

                end loop;
                wait;
            end process;

            OUTPUTS: process
            variable ss: side;
            variable bw: width := 6;
            variable l: line;
                    variable tm: time;
                    variable once: bit := '0';
--          file t: text is out "/home/bsparkma/Senior_Thesis/ModelSim/t.txt";
                    variable j: integer := 0;
        begin
--                  ki.rail0 <= '0';
                    while (true) loop
                        ki <= '1';
                        wait until s'event and is_data(s);

                        for i in 0 to 7 loop
                            if (s(i).rail1 /= s_calc_array(j)(i)) then
                                incorrect <= '1';
                            end if;
                        end loop;

                        if (once = '0') then
                            once := '1';
                        else
--                          write(l, now - tm, ss, bw);
--                          writeline(t, l);
                        end if;
```

```
                    tm := now;
                    ki <= '0';
                    wait until s'event and is_null(s);
                    j := j+1;
                end loop;
        end process;
end TESTBENCH;

configuration CFG_TB_MULT4x4 of TB_MULT4x4 is
        for TESTBENCH
                for UUT: MULT4x4 --_1n
                end for;}
        end for;
end;
```

B. VERILOG FILE

```
module mult4x4 ( vdd, gnd, x_3__RAIL1, x_3__RAIL0, x_2__RAIL1, x_2__RAIL0, x_1__RAIL1,
x_1__RAIL0, x_0__RAIL1, x_0__RAIL0, y_3__RAIL1, y_3__RAIL0, y_2__RAIL1, y_2__RAIL0,
y_1__RAIL1, y_1__RAIL0, y_0__RAIL1, y_0__RAIL0, ki, reset, s_7__RAIL1, s_7__RAIL0,
s_6__RAIL1, s_6__RAIL0, s_5__RAIL1, s_5__RAIL0, s_4__RAIL1, s_4__RAIL0,
s_3__RAIL1, s_3__RAIL0, s_2__RAIL1, s_2__RAIL0, s_1__RAIL1, s_1__RAIL0, s_0__RAIL1,
s_0__RAIL0, ko ) ;
    inout vdd, gnd;  input x_3__RAIL1 ;
     input x_3__RAIL0 ;
     input x_2__RAIL1 ;
     input x_2__RAIL0 ;
     input x_1__RAIL1 ;
     input x_1__RAIL0 ;
     input x_0__RAIL1 ;
     input x_0__RAIL0 ;
     input y_3__RAIL1 ;
     input y_3__RAIL0 ;
     input y_2__RAIL1 ;
     input y_2__RAIL0 ;
     input y_1__RAIL1 ;
     input y_1__RAIL0 ;
     input y_0__RAIL1 ;
     input y_0__RAIL0 ;
     input ki ;
     input reset ;
     output s_7__RAIL1 ;
     output s_7__RAIL0 ;
     output s_6__RAIL1 ;
     output s_6__RAIL0 ;
     output s_5__RAIL1 ;
     output s_5__RAIL0 ;
     output s_4__RAIL1 ;
     output s_4__RAIL0 ;
     output s_3__RAIL1 ;
     output s_3__RAIL0 ;
     output s_2__RAIL1 ;
     output s_2__RAIL0 ;
     output s_1__RAIL1 ;
     output s_1__RAIL0 ;
     output s_0__RAIL1 ;
     output s_0__RAIL0 ;
     output ko ;
    wire  di1_7__RAIL1,  di1_7__RAIL0,  di1_6__RAIL1,  di1_6__RAIL0,  di1_5__RAIL1,
di1_5__RAIL0, di1_4__RAIL1, di1_4__RAIL0, di1_3__RAIL1, di1_3__RAIL0, di1_2__RAIL1,
di1_2__RAIL0, di1_1__RAIL1, di1_1__RAIL0, di1_0__RAIL1, di1_0__RAIL0, di2_7__RAIL1,
di2_7__RAIL0, di2_6__RAIL1, di2_6__RAIL0, di2_5__RAIL1, di2_5__RAIL0, di2_4__RAIL1,
di2_4__RAIL0, di2_3__RAIL1, di2_3__RAIL0, di2_2__RAIL1, di2_2__RAIL0, di2_1__RAIL1,
di2_1__RAIL0, di2_0__RAIL1, di2_0__RAIL0, do1_7__RAIL1, do1_7__RAIL0, do1_6__RAIL1,
do1_6__RAIL0, do1_5__RAIL1, do1_5__RAIL0, do1_4__RAIL1, do1_4__RAIL0, do1_3__RAIL1,
do1_3__RAIL0, do1_2__RAIL1, do1_2__RAIL0, do1_1__RAIL1, do1_1__RAIL0, do1_0__RAIL1,
do1_0__RAIL0, do2_7__RAIL1, do2_7__RAIL0;
```

```
        wire            do2_6__RAIL1, do2_6__RAIL0, do2_5__RAIL1, do2_5__RAIL0, do2_4__RAIL1,
do2_4__RAIL0, do2_3__RAIL1, do2_3__RAIL0, do2_2__RAIL1, do2_2__RAIL0,          do2_1__RAIL1,
do2_1__RAIL0, do2_0__RAIL1, do2_0__RAIL0, kod_7, kod_6,       kod_5, kod_4, kod_3, kod_2, kod_1,
kod_0, ko1, ko2, ki1, ki2, s1, s2,                COMP_l1_1, COMP_l1_2, SELECT_INPUT_d1,
SELECT_INPUT_d3,    SELECT_INPUT_r0,                  SELECT_INPUT_r1,    SELECT_INPUT_r2,
SELECT_INPUT_r3,  SELECT_OUTPUT_d1,               SELECT_OUTPUT_d3,   SELECT_OUTPUT_r0,
SELECT_OUTPUT_r1, SELECT_OUTPUT_r2,      SELECT_OUTPUT_r3;

        //endofwire
        wire buffwire0_0ko, buffwire1_1kos00seq;
        wire buffwire0_0reset, buffwire1_0reset, buffwire2_1reset, buffwire3_2reset, buffwire2_1resets00seq;
        wire buffwire0_0s2, buffwire1_1s2s00seq;
        wire buffwire0_0s1, buffwire1_1s1s00seq;
        wire buffwire0_0ki, buffwire1_1kis00seq;
        wire buffwire0_0ko1, buffwire1_1ko1s00seq;
        wire buffwire0_0ko2, buffwire1_1ko2s00seq;

          mult4x4_1n COMB1 ( .vdd(vdd), .gnd(gnd), .x_3__RAIL1 (di1_7__RAIL1), .x_3__RAIL0
(di1_7__RAIL0), .x_2__RAIL1 (          di1_6__RAIL1), .x_2__RAIL0 (di1_6__RAIL0), .x_1__RAIL1 (
di1_5__RAIL1), .x_1__RAIL0 (di1_5__RAIL0), .x_0__RAIL1 (          di1_4__RAIL1), .x_0__RAIL0
(di1_4__RAIL0), .y_3__RAIL1 (          di1_3__RAIL1), .y_3__RAIL0 (di1_3__RAIL0), .y_2__RAIL1 (
di1_2__RAIL1), .y_2__RAIL0 (di1_2__RAIL0), .y_1__RAIL1 (          di1_1__RAIL1), .y_1__RAIL0
(di1_1__RAIL0), .y_0__RAIL1 (          di1_0__RAIL1), .y_0__RAIL0 (di1_0__RAIL0), .ki (ki1), .reset
(buffwire0_0reset),  .s_7__RAIL1  (do1_7__RAIL1),  .s_7__RAIL0  (do1_7__RAIL0),  .s_6__RAIL1  (
do1_6__RAIL1), .s_6__RAIL0 (do1_6__RAIL0), .s_5__RAIL1 (          do1_5__RAIL1), .s_5__RAIL0
(do1_5__RAIL0), .s_4__RAIL1 (          do1_4__RAIL1), .s_4__RAIL0 (do1_4__RAIL0), .s_3__RAIL1 (
do1_3__RAIL1), .s_3__RAIL0 (do1_3__RAIL0), .s_2__RAIL1 (          do1_2__RAIL1), .s_2__RAIL0
(do1_2__RAIL0), .s_1__RAIL1 (          do1_1__RAIL1), .s_1__RAIL0 (do1_1__RAIL0), .s_0__RAIL1 (
do1_0__RAIL1), .s_0__RAIL0 (do1_0__RAIL0), .ko (ko1)) ;
          mult4x4_1n COMB2 ( .vdd(vdd), .gnd(gnd), .x_3__RAIL1 (di2_7__RAIL1), .x_3__RAIL0
(di2_7__RAIL0), .x_2__RAIL1 (          di2_6__RAIL1), .x_2__RAIL0 (di2_6__RAIL0), .x_1__RAIL1 (
di2_5__RAIL1), .x_1__RAIL0 (di2_5__RAIL0), .x_0__RAIL1 (          di2_4__RAIL1), .x_0__RAIL0
(di2_4__RAIL0), .y_3__RAIL1 (          di2_3__RAIL1), .y_3__RAIL0 (di2_3__RAIL0), .y_2__RAIL1 (
di2_2__RAIL1), .y_2__RAIL0 (di2_2__RAIL0), .y_1__RAIL1 (          di2_1__RAIL1), .y_1__RAIL0
(di2_1__RAIL0), .y_0__RAIL1 (          di2_0__RAIL1), .y_0__RAIL0 (di2_0__RAIL0), .ki (ki2), .reset
(buffwire0_0reset),  .s_7__RAIL1  (do2_7__RAIL1),  .s_7__RAIL0  (do2_7__RAIL0),  .s_6__RAIL1  (
do2_6__RAIL1), .s_6__RAIL0 (do2_6__RAIL0), .s_5__RAIL1 (          do2_5__RAIL1), .s_5__RAIL0
(do2_5__RAIL0), .s_4__RAIL1 (          do2_4__RAIL1), .s_4__RAIL0 (do2_4__RAIL0), .s_3__RAIL1 (
do2_3__RAIL1), .s_3__RAIL0 (do2_3__RAIL0), .s_2__RAIL1 (          do2_2__RAIL1), .s_2__RAIL0
(do2_2__RAIL0), .s_1__RAIL1 (          do2_1__RAIL1), .s_1__RAIL0 (do2_1__RAIL0), .s_0__RAIL1 (
do2_0__RAIL1), .s_0__RAIL0 (do2_0__RAIL0), .ko (ko2)) ;
          th44x0 COMP_g0 ( .vdd(vdd), .gnd(gnd), .a (kod_0), .b (kod_1), .c (kod_2), .d (kod_3), .z (
COMP_l1_1)) ;
          th44x0 COMP_g1 ( .vdd(vdd), .gnd(gnd), .a (kod_4), .b (kod_5), .c (kod_6), .d (kod_7), .z (
COMP_l1_2)) ;
          th22x0 COMP_g2 ( .vdd(vdd), .gnd(gnd), .a (COMP_l1_1), .b (COMP_l1_2), .z (ko)) ;
          th33nx0 SELECT_INPUT_g0 ( .vdd(vdd), .gnd(gnd), .a (buffwire0_0ko), .b (SELECT_INPUT_d3), .c
(SELECT_INPUT_r1)        , .rst (buffwire0_0reset), .z (s2)) ;
          th33dx0 SELECT_INPUT_g1 ( .vdd(vdd), .gnd(gnd), .a (buffwire0_0ko), .b (buffwire0_0s2), .c
(SELECT_INPUT_r2), .rst (buffwire0_0reset), .z (SELECT_INPUT_d1)) ;
          th33nx0 SELECT_INPUT_g2 ( .vdd(vdd), .gnd(gnd), .a (buffwire0_0ko), .b (SELECT_INPUT_d1), .c
(SELECT_INPUT_r3)        , .rst (buffwire0_0reset), .z (s1)) ;
          th33nx0 SELECT_INPUT_g3 ( .vdd(vdd), .gnd(gnd), .a (buffwire0_0ko), .b (buffwire0_0s1), .c
(SELECT_INPUT_r0), .rst (buffwire0_0reset), .z (SELECT_INPUT_d3)) ;
          invx0 SELECT_INPUT_i0 ( .vdd(vdd), .gnd(gnd), .i (buffwire0_0s2), .zb (SELECT_INPUT_r0)) ;
```

```
        invx0 SELECT_INPUT_i1 ( .vdd(vdd), .gnd(gnd), .i (SELECT_INPUT_d1), .zb (SELECT_INPUT_r1))
;
        invx0 SELECT_INPUT_i2 ( .vdd(vdd), .gnd(gnd), .i (buffwire0_0s1), .zb (SELECT_INPUT_r2)) ;
        invx0 SELECT_INPUT_i3 ( .vdd(vdd), .gnd(gnd), .i (SELECT_INPUT_d3), .zb (SELECT_INPUT_r3))
;
        th12x0  MUX_OUTPUT_struct_7_comp0  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_7__RAIL0),  .b
(do2_7__RAIL0), .z (      s_7__RAIL0)) ;
        th12x0  MUX_OUTPUT_struct_7_comp1  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_7__RAIL1),  .b
(do2_7__RAIL1), .z (      s_7__RAIL1)) ;
        th12x0  MUX_OUTPUT_struct_6_comp0  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_6__RAIL0),  .b
(do2_6__RAIL0), .z (      s_6__RAIL0)) ;
        th12x0  MUX_OUTPUT_struct_6_comp1  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_6__RAIL1),  .b
(do2_6__RAIL1), .z (      s_6__RAIL1)) ;
        th12x0  MUX_OUTPUT_struct_5_comp0  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_5__RAIL0),  .b
(do2_5__RAIL0), .z (      s_5__RAIL0)) ;
        th12x0  MUX_OUTPUT_struct_5_comp1  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_5__RAIL1),  .b
(do2_5__RAIL1), .z (      s_5__RAIL1)) ;
        th12x0  MUX_OUTPUT_struct_4_comp0  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_4__RAIL0),  .b
(do2_4__RAIL0), .z (      s_4__RAIL0)) ;
        th12x0  MUX_OUTPUT_struct_4_comp1  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_4__RAIL1),  .b
(do2_4__RAIL1), .z (      s_4__RAIL1)) ;
        th12x0  MUX_OUTPUT_struct_3_comp0  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_3__RAIL0),  .b
(do2_3__RAIL0), .z (      s_3__RAIL0)) ;
        th12x0  MUX_OUTPUT_struct_3_comp1  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_3__RAIL1),  .b
(do2_3__RAIL1), .z (      s_3__RAIL1)) ;
        th12x0  MUX_OUTPUT_struct_2_comp0  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_2__RAIL0),  .b
(do2_2__RAIL0), .z (      s_2__RAIL0)) ;
        th12x0  MUX_OUTPUT_struct_2_comp1  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_2__RAIL1),  .b
(do2_2__RAIL1), .z (      s_2__RAIL1)) ;
        th12x0  MUX_OUTPUT_struct_1_comp0  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_1__RAIL0),  .b
(do2_1__RAIL0), .z (      s_1__RAIL0)) ;
        th12x0  MUX_OUTPUT_struct_1_comp1  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_1__RAIL1),  .b
(do2_1__RAIL1), .z (      s_1__RAIL1)) ;
        th12x0  MUX_OUTPUT_struct_0_comp0  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_0__RAIL0),  .b
(do2_0__RAIL0), .z (      s_0__RAIL0)) ;
        th12x0  MUX_OUTPUT_struct_0_comp1  (  .vdd(vdd),  .gnd(gnd),  .a  (do1_0__RAIL1),  .b
(do2_0__RAIL1), .z (      s_0__RAIL1)) ;
        th33nx0 SELECT_OUTPUT_g0 ( .vdd(vdd), .gnd(gnd), .a (buffwire0_0ki), .b (SELECT_OUTPUT_d3),
.c (      SELECT_OUTPUT_r1), .rst (buffwire0_0reset), .z (ki2)) ;
        th33dx0  SELECT_OUTPUT_g1  (  .vdd(vdd),  .gnd(gnd),  .a  (buffwire0_0ki),  .b  (ki2),  .c
(SELECT_OUTPUT_r2), .rst (buffwire0_0reset), .z (SELECT_OUTPUT_d1)) ;
        th33nx0 SELECT_OUTPUT_g2 ( .vdd(vdd), .gnd(gnd), .a (buffwire0_0ki), .b (SELECT_OUTPUT_d1),
.c (      SELECT_OUTPUT_r3), .rst (buffwire0_0reset), .z (ki1)) ;
        th33nx0  SELECT_OUTPUT_g3  (  .vdd(vdd),  .gnd(gnd),  .a  (buffwire0_0ki),  .b  (ki1),  .c
(SELECT_OUTPUT_r0), .rst (buffwire0_0reset), .z (SELECT_OUTPUT_d3)) ;
        invx0 SELECT_OUTPUT_i0 ( .vdd(vdd), .gnd(gnd), .i (ki2), .zb (SELECT_OUTPUT_r0)) ;
        invx0  SELECT_OUTPUT_i1  (  .vdd(vdd),  .gnd(gnd),  .i  (SELECT_OUTPUT_d1),  .zb
(SELECT_OUTPUT_r1)) ;
        invx0 SELECT_OUTPUT_i2 ( .vdd(vdd), .gnd(gnd), .i (ki1), .zb (SELECT_OUTPUT_r2)) ;
        invx0  SELECT_OUTPUT_i3  (  .vdd(vdd),  .gnd(gnd),  .i  (SELECT_OUTPUT_d3),  .zb
(SELECT_OUTPUT_r3)) ;
        th33nx0  DEMUX_INPUT_struct_7_comp_i11  (  .vdd(vdd),  .gnd(gnd),  .a  (x_3__RAIL1),  .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire0_0reset), .z (di1_7__RAIL1)) ;
        th33nx0  DEMUX_INPUT_struct_7_comp_i10  (  .vdd(vdd),  .gnd(gnd),  .a  (x_3__RAIL0),  .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire0_0reset), .z (di1_7__RAIL0)) ;
```

```
th33nx0 DEMUX_INPUT_struct_7_comp_i21 ( .vdd(vdd), .gnd(gnd), .a (x_3__RAIL1), .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire0_0reset), .z (di2_7__RAIL1)) ;
th33nx0 DEMUX_INPUT_struct_7_comp_i20 ( .vdd(vdd), .gnd(gnd), .a (x_3__RAIL0), .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire0_0reset), .z (di2_7__RAIL0)) ;
th14bx0 DEMUX_INPUT_struct_7_comp_k0 ( .vdd(vdd), .gnd(gnd), .a (di1_7__RAIL1), .b
(di1_7__RAIL0), .c (di2_7__RAIL1), .d (di2_7__RAIL0), .zb (kod_7)) ;
th33nx0 DEMUX_INPUT_struct_6_comp_i11 ( .vdd(vdd), .gnd(gnd), .a (x_2__RAIL1), .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire0_0reset), .z (di1_6__RAIL1)) ;
th33nx0 DEMUX_INPUT_struct_6_comp_i10 ( .vdd(vdd), .gnd(gnd), .a (x_2__RAIL0), .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire0_0reset), .z (di1_6__RAIL0)) ;
th33nx0 DEMUX_INPUT_struct_6_comp_i21 ( .vdd(vdd), .gnd(gnd), .a (x_2__RAIL1), .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire0_0reset), .z (di2_6__RAIL1)) ;
th33nx0 DEMUX_INPUT_struct_6_comp_i20 ( .vdd(vdd), .gnd(gnd), .a (x_2__RAIL0), .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire0_0reset), .z (di2_6__RAIL0)) ;
th14bx0 DEMUX_INPUT_struct_6_comp_k0 ( .vdd(vdd), .gnd(gnd), .a (di1_6__RAIL1), .b
(di1_6__RAIL0), .c (di2_6__RAIL1), .d (di2_6__RAIL0), .zb (kod_6)) ;
th33nx0 DEMUX_INPUT_struct_5_comp_i11 ( .vdd(vdd), .gnd(gnd), .a (x_1__RAIL1), .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire0_0reset), .z (di1_5__RAIL1)) ;
th33nx0 DEMUX_INPUT_struct_5_comp_i10 ( .vdd(vdd), .gnd(gnd), .a (x_1__RAIL0), .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire0_0reset), .z (di1_5__RAIL0)) ;
th33nx0 DEMUX_INPUT_struct_5_comp_i21 ( .vdd(vdd), .gnd(gnd), .a (x_1__RAIL1), .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire0_0reset), .z (di2_5__RAIL1)) ;
th33nx0 DEMUX_INPUT_struct_5_comp_i20 ( .vdd(vdd), .gnd(gnd), .a (x_1__RAIL0), .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire0_0reset), .z (di2_5__RAIL0)) ;
th14bx0 DEMUX_INPUT_struct_5_comp_k0 ( .vdd(vdd), .gnd(gnd), .a (di1_5__RAIL1), .b
(di1_5__RAIL0), .c (di2_5__RAIL1), .d (di2_5__RAIL0), .zb (kod_5)) ;
th33nx0 DEMUX_INPUT_struct_4_comp_i11 ( .vdd(vdd), .gnd(gnd), .a (x_0__RAIL1), .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire0_0reset), .z (di1_4__RAIL1)) ;
th33nx0 DEMUX_INPUT_struct_4_comp_i10 ( .vdd(vdd), .gnd(gnd), .a (x_0__RAIL0), .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire0_0reset), .z (di1_4__RAIL0)) ;
th33nx0 DEMUX_INPUT_struct_4_comp_i21 ( .vdd(vdd), .gnd(gnd), .a (x_0__RAIL1), .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire0_0reset), .z (di2_4__RAIL1)) ;
th33nx0 DEMUX_INPUT_struct_4_comp_i20 ( .vdd(vdd), .gnd(gnd), .a (x_0__RAIL0), .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire0_0reset), .z (di2_4__RAIL0)) ;
th14bx0 DEMUX_INPUT_struct_4_comp_k0 ( .vdd(vdd), .gnd(gnd), .a (di1_4__RAIL1), .b
(di1_4__RAIL0), .c (di2_4__RAIL1), .d (di2_4__RAIL0), .zb (kod_4)) ;
th33nx0 DEMUX_INPUT_struct_3_comp_i11 ( .vdd(vdd), .gnd(gnd), .a (y_3__RAIL1), .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire0_0reset), .z (di1_3__RAIL1)) ;
th33nx0 DEMUX_INPUT_struct_3_comp_i10 ( .vdd(vdd), .gnd(gnd), .a (y_3__RAIL0), .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire0_0reset), .z (di1_3__RAIL0)) ;
th33nx0 DEMUX_INPUT_struct_3_comp_i21 ( .vdd(vdd), .gnd(gnd), .a (y_3__RAIL1), .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire0_0reset), .z (di2_3__RAIL1)) ;
th33nx0 DEMUX_INPUT_struct_3_comp_i20 ( .vdd(vdd), .gnd(gnd), .a (y_3__RAIL0), .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire0_0reset), .z (di2_3__RAIL0)) ;
th14bx0 DEMUX_INPUT_struct_3_comp_k0 ( .vdd(vdd), .gnd(gnd), .a (di1_3__RAIL1), .b
(di1_3__RAIL0), .c (di2_3__RAIL1), .d (di2_3__RAIL0), .zb (kod_3)) ;
th33nx0 DEMUX_INPUT_struct_2_comp_i11 ( .vdd(vdd), .gnd(gnd), .a (y_2__RAIL1), .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire1_0reset), .z (di1_2__RAIL1)) ;
th33nx0 DEMUX_INPUT_struct_2_comp_i10 ( .vdd(vdd), .gnd(gnd), .a (y_2__RAIL0), .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire1_0reset), .z (di1_2__RAIL0)) ;
th33nx0 DEMUX_INPUT_struct_2_comp_i21 ( .vdd(vdd), .gnd(gnd), .a (y_2__RAIL1), .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire1_0reset), .z (di2_2__RAIL1)) ;
th33nx0 DEMUX_INPUT_struct_2_comp_i20 ( .vdd(vdd), .gnd(gnd), .a (y_2__RAIL0), .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire1_0reset), .z (di2_2__RAIL0)) ;
th14bx0 DEMUX_INPUT_struct_2_comp_k0 ( .vdd(vdd), .gnd(gnd), .a (di1_2__RAIL1), .b
(di1_2__RAIL0), .c (di2_2__RAIL1), .d (di2_2__RAIL0), .zb (kod_2)) ;
```

```
        th33nx0  DEMUX_INPUT_struct_1_comp_i11  (  .vdd(vdd),  .gnd(gnd),  .a  (y_1__RAIL1),  .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire1_0reset), .z (di1_1__RAIL1)) ;
        th33nx0  DEMUX_INPUT_struct_1_comp_i10  (  .vdd(vdd),  .gnd(gnd),  .a  (y_1__RAIL0),  .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire1_0reset), .z (di1_1__RAIL0)) ;
        th33nx0  DEMUX_INPUT_struct_1_comp_i21  (  .vdd(vdd),  .gnd(gnd),  .a  (y_1__RAIL1),  .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire1_0reset), .z (di2_1__RAIL1)) ;
        th33nx0  DEMUX_INPUT_struct_1_comp_i20  (  .vdd(vdd),  .gnd(gnd),  .a  (y_1__RAIL0),  .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire1_0reset), .z (di2_1__RAIL0)) ;
        th14bx0  DEMUX_INPUT_struct_1_comp_k0  (  .vdd(vdd),  .gnd(gnd),  .a  (di1_1__RAIL1),  .b
(di1_1__RAIL0),        .c (di2_1__RAIL1), .d (di2_1__RAIL0), .zb (kod_1)) ;
        th33nx0  DEMUX_INPUT_struct_0_comp_i11  (  .vdd(vdd),  .gnd(gnd),  .a  (y_0__RAIL1),  .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire1_0reset), .z (di1_0__RAIL1)) ;
        th33nx0  DEMUX_INPUT_struct_0_comp_i10  (  .vdd(vdd),  .gnd(gnd),  .a  (y_0__RAIL0),  .b
(buffwire0_0s1), .c (buffwire0_0ko1), .rst (buffwire1_0reset), .z (di1_0__RAIL0)) ;
        th33nx0  DEMUX_INPUT_struct_0_comp_i21  (  .vdd(vdd),  .gnd(gnd),  .a  (y_0__RAIL1),  .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire1_0reset), .z (di2_0__RAIL1)) ;
        th33nx0  DEMUX_INPUT_struct_0_comp_i20  (  .vdd(vdd),  .gnd(gnd),  .a  (y_0__RAIL0),  .b
(buffwire0_0s2), .c (buffwire0_0ko2), .rst (buffwire1_0reset), .z (di2_0__RAIL0)) ;
        th14bx0  DEMUX_INPUT_struct_0_comp_k0  (  .vdd(vdd),  .gnd(gnd),  .a  (di1_0__RAIL1),  .b
(di1_0__RAIL0),        .c (di2_0__RAIL1), .d (di2_0__RAIL0), .zb (kod_0)) ;
    invert_a Gbuff_ko_0 ( .vdd(vdd), .gnd(gnd), .a(buffwire1_1kos00seq), .z(buffwire0_0ko));
    invert_a Gbuff_ko_1 ( .vdd(vdd), .gnd(gnd), .a(ko), .z(buffwire1_1kos00seq));
    buffer_c Gbuff_reset_0 ( .vdd(vdd), .gnd(gnd), .a(buffwire2_1reset), .z(buffwire0_0reset));
    invert_a Gbuff_reset_1 ( .vdd(vdd), .gnd(gnd), .a(buffwire2_1resets00seq), .z(buffwire1_0reset));
    invert_a Gbuff_reset_2 ( .vdd(vdd), .gnd(gnd), .a(buffwire2_1reset), .z(buffwire2_1resets00seq));
    invert_a Gbuff_reset_4 ( .vdd(vdd), .gnd(gnd), .a(buffwire3_2reset), .z(buffwire2_1reset));
    invert_a Gbuff_reset_5 ( .vdd(vdd), .gnd(gnd), .a(reset), .z(buffwire3_2reset));
    invert_c Gbuff_s2_0 ( .vdd(vdd), .gnd(gnd), .a(buffwire1_1s2s00seq), .z(buffwire0_0s2));
    invert_a Gbuff_s2_1 ( .vdd(vdd), .gnd(gnd), .a(s2), .z(buffwire1_1s2s00seq));
    invert_c Gbuff_s1_0 ( .vdd(vdd), .gnd(gnd), .a(buffwire1_1s1s00seq), .z(buffwire0_0s1));
    invert_a Gbuff_s1_1 ( .vdd(vdd), .gnd(gnd), .a(s1), .z(buffwire1_1s1s00seq));
    invert_a Gbuff_ki_0 ( .vdd(vdd), .gnd(gnd), .a(buffwire1_1kis00seq), .z(buffwire0_0ki));
    invert_a Gbuff_ki_1 ( .vdd(vdd), .gnd(gnd), .a(ki), .z(buffwire1_1kis00seq));
    invert_c Gbuff_ko1_0 ( .vdd(vdd), .gnd(gnd), .a(buffwire1_1ko1s00seq), .z(buffwire0_0ko1));
    invert_a Gbuff_ko1_1 ( .vdd(vdd), .gnd(gnd), .a(ko1), .z(buffwire1_1ko1s00seq));
    invert_c Gbuff_ko2_0 ( .vdd(vdd), .gnd(gnd), .a(buffwire1_1ko2s00seq), .z(buffwire0_0ko2));
    invert_a Gbuff_ko2_1 ( .vdd(vdd), .gnd(gnd), .a(ko2), .z(buffwire1_1ko2s00seq));
    endmodule
```

C. ADDITIONAL-MULTIPLIER VHDL FILES

C.1 Two-Multiplier Design: mult4x4_1stage2.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use work.ncl_signals.all;

entity mult4x4_1n2 is
  port(x, y: in dual_rail_logic_VECTOR(3 downto 0);
       ki, reset: in std_logic;
       s: out dual_rail_logic_VECTOR(7 downto 0);
       ko: out std_logic);
end;

architecture BEHAVIOR of mult4x4_1n2 is

       signal di2: dual_rail_logic_vector(7 downto 0);
       signal ki2: std_logic;

  component mult4x4_1n
    port(x, y: IN dual_rail_logic_vector (3 downto 0);
              ki, reset: IN std_logic;
              s: OUT dual_rail_logic_vector (7 downto 0);
              ko: OUT std_logic);
  end component;

begin

       U0: mult4x4_1n
       port map(x, y, ki2, reset, di2, ko);

       U1: mult4x4_1n
       port map(di2(7 downto 4), di2(3 downto 0), ki, reset, s, ki2);

end BEHAVIOR;
```

C.2 Four-Multiplier Design: mult4x4_1stage4.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use work.ncl_signals.all;

entity mult4x4_1n4 is
  port(x, y: in dual_rail_logic_VECTOR(3 downto 0);
       ki, reset: in std_logic;
       s: out dual_rail_logic_VECTOR(7 downto 0);
       ko: out std_logic);
end;

architecture BEHAVIOR of mult4x4_1n4 is
```

```
        signal di2: dual_rail_logic_vector(7 downto 0);
        signal ki2: std_logic;

    component mult4x4_1n2
      port(x, y: IN dual_rail_logic_vector (3 downto 0);
                 ki, reset: IN std_logic;
                 s: OUT dual_rail_logic_vector (7 downto 0);
                 ko: OUT std_logic);
    end component;

begin

        U0: mult4x4_1n2
        port map(x, y, ki2, reset, di2, ko);

        U1: mult4x4_1n2
        port map(di2(7 downto 4), di2(3 downto 0), ki, reset, s, ki2);

end BEHAVIOR;
```