California State University, Fresno
Lyles College of Engineering
Electrical and Computer Engineering Department

FINAL PROJECT REPORT

DESIGN, SYNTHESIS & TEST PATTERN GENERATION OF HIGH-SPEED BOOTH MULTIPLIER

ECE-242: DIGITAL SYSTEMS TESTING AND TESTABLE DESIGN

Date Submitted:05/06/2023.

SUBMITTED BY:

DINESH REDDY MUNNANGI

INSTRUCTOR SECTION

Comments: _____

_____

_____

Final Grade: _____

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The Booth multiplier is a hardware implementation of a multiplication algorithm aimed at enhancing speed and efficiency in digital circuit design. This paper presents a Booth multiplier model designed using an 8-bit adder-subtractor module to perform multiplication operations on two 8-bit binary inputs, generating a 16-bit output. The module is generated and verified using Verilog, synthesized using Synopsys Design Compiler. Timing, area, power, and cell reports were generated to assess the performance and characteristics of the synthesized design. Test patterns were generated using Synopsys TetraMAX. The generated test patterns can be used after manufacturing the chip to find faulty chips. The results indicate successful simulation, synthesis, and verification of the Booth multiplier model, demonstrating its potential for efficient multiplication in digital systems. Future directions for optimizing performance, reducing power consumption, and exploring application-specific customization are discussed. Overall, this paper contributes to advancing digital circuit design methodologies and multiplication algorithms for various computing applications.

Keywords: Pattern Generation, Adder-Subtractor, Design Compiler, TetraMAX

I. INTRODUCTION

In the upcoming VLSI technology, multipliers play a crucial role in implementing various computational tasks efficiently. Imagine you're designing a digital signal processing (DSP) system for a real-world scenario, like processing audio signals in a music player [1]. The multiplier circuit serves a main purpose in VLSI systems, enabling the rapid calculation of products essential for wide range of applications [2].

In this scenario, multipliers are essential components for tasks such as filtering, equalization, or performing Fourier transforms for spectral analysis [1]. For instance, when you're applying an equalization filter to adjust the frequency response of an audio signal, you often need to multiply each sample of the signal by a corresponding coefficient. These coefficients represent the filter's characteristics and multiplying them with the input signal alters its frequency components accordingly [2].

Moreover, in applications like wireless communication systems, multipliers are fundamental for tasks like modulation and demodulation, where complex mathematical operations are performed on signals in real-time. Multipliers enable these systems to process data efficiently, helping achieve high throughput and low latency. So, in the real world, the efficient implementation of multipliers in Verilog directly impacts the performance and functionality of various digital systems, ranging from consumer electronics to telecommunications infrastructure.

This technical report aims to provide a simplified overview of multiplier circuits in the context of VLSI testing. We will explore the basic principles behind multiplier circuit design, the challenges associated with testing these circuits, and strategies employed to ensure their accuracy and reliability [1].

By delving into the world of multiplier circuits in VLSI testing, this report seeks to offer insights accessible to both novice and seasoned professionals in the field. Through a clearer understanding of multiplier circuits and their testing methodologies, we aim to contribute to the advancement of VLSI design and testing practices [2].

Through this integrated approach, combining Verilog design, simulation with ModelSim, synthesis with Synopsys DC compiler, and test pattern generation with Synopsys TetraMAX, we aim to develop a low-power, high-speed 8-bit Booth multiplier that meets the stringent requirements of modern digital systems. This project showcases the power of Verilog and industry-standard tools in enabling efficient and reliable digital design implementations.

II. LITERATURE SURVEY

In the realm of digital design, various types of multipliers offer distinct advantages and trade-offs, catering to different requirements in terms of speed, area efficiency, and power consumption. Array multiplier, one of the simplest forms, implements multiplication by decomposing the operation into a series of AND and shift operations. While straightforward, it tends to be slower and occupies more area compared to other types due to its fully parallel structure [2].

Vedic multiplier, inspired by ancient Indian mathematics, employs techniques like Urdhva Tiryakbhyam to achieve faster multiplication. It breaks down the multiplication process into smaller, simpler steps, reducing the number of partial products generated and improving speed and efficiency. Booth multiplier, on the other hand, optimizes the multiplication process by encoding consecutive same bits in the multiplier into partial products, thereby reducing the number of operations required and enhancing speed [3].

Wallace tree multiplier and Dadda multiplier represent further refinements in terms of area efficiency and speed. Wallace tree multiplier reduces the number of partial products by grouping them into weighted sets and summing them in a hierarchical tree structure. Similarly, Dadda multiplier employs a similar tree-based approach but further optimizes it by reducing the number of partial products and levels in the tree, thus saving area, and improving speed. Each of these multiplier architectures presents digital designers with a range of options to choose from, depending on the specific requirements of their design in terms of speed, area, and power efficiency [2].

The choice of multiplier design technique depends on the specific requirements of the digital system, including area constraints, speed, and power consumption considerations. Each multiplier architecture offers distinct advantages and trade-offs, allowing designers to select the most suitable approach based on the application's needs[4]. Further research and innovation in multiplier design continue to drive advancements in digital arithmetic circuits, enabling the development of more efficient and high-performance digital systems.

In summary, we are developing a Booth multiplier that empowers digital designers to create high-speed, area-efficient, and low-power multiplication solutions tailored to their specific design requirements. By harnessing the benefits of Booth multiplication technique, designers can enhance the performance and energy efficiency of their digital systems, ultimately delivering more robust and cost-effective solutions to end-users.

III. PROPOSED MODEL

A. Booth Multiplier

A Booth multiplier is like a smart calculator that can quickly do multiplication. It's named after Andrew Donald Booth, who came up with the idea in 1950. This type of multiplication is good because it's fast, uses less space, and doesn't use up too much power. Here's how it works: instead of doing every single step of the multiplication, it looks for patterns in the numbers you're multiplying. For example, if you're multiplying by a number with a lot of zeros in a row, the Booth multiplier doesn't have to do as many steps. It kind of skips ahead, which makes things go faster [2].

To figure out these shortcuts, the Booth multiplier looks at pairs of numbers in the one you're multiplying by. It's like a secret code that helps it figure out the answer more efficiently. This makes the Booth multiplier handy for lots of different math problems, whether you're working with positive or negative numbers. It is very efficient too. It works on the string bits 0's in the multiplier that requires no additional bit only shift the right-most string bits and a string of 1's in a multiplier bit weight 2^k to weight 2^m that can be considered as $2^{k+1} - 2^m$ [3].

B. Booth's Algorithm

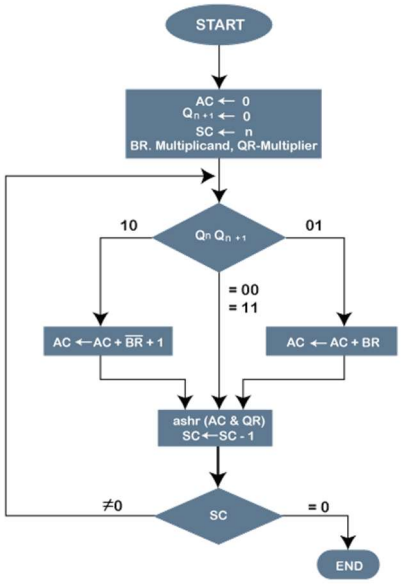

Figure 3.1: Booth's Algorithm [3]

- The process starts with setting the accumulator (AC) to 0.
- Then it subtracts the number we want to multiply by (multiplicand) from the product register (PR). The product register is initialized to 0.
- If the sign bit (SC) of the product register is not 0, it adds 1 to the quotient register (QR) and subtracts the multiplicand from the product register again.
- After that, it shifts the AC and QR together to the right by one bit.
- Then it checks the least significant bit (ashr) of both the AC and QR. If it's not 0, it subtracts the multiplicand from the product register and goes back to shifting.
- If the least significant bit is 0, it adds the multiplicand to the product register and then it shifts again.
- These steps continue until the counter (SC) reaches 0.

The result of the multiplication is then stored in the AC and QR. This flowchart specifically refers to a version of Booth's algorithm known as a signed digit Booth multiplier.

C. Modules and Submodules

The "booth_multiplier()" module takes two signed 8-bit inputs, representing the multiplicand and multiplier, and produces one signed 16-bit output, the product. It incorporates the "booth_substep()" submodule, responsible for executing one iteration of Booth's multiplication algorithm. This submodule takes an 8-bit signed accumulator, multiplier, multiplicand, and a control signal "q0", generating updated values for the accumulator, multiplier, and the next control signal "next_q0". Additionally, it utilizes the "eight_bit_adder_subtractor()" submodule for performing addition or subtraction operations on two 8-bit signed numbers based on the value of the carry-in signal (cin). This submodule employs lower-level components like "xor2" for bitwise XOR operations and fa for full-adder functionality to execute addition or subtraction operations accurately. Overall, the "booth_multiplier()" module orchestrates the Booth multiplication process by coordinating the actions of its submodules to generate the desired product efficiently.

D. Procedure and Implementation

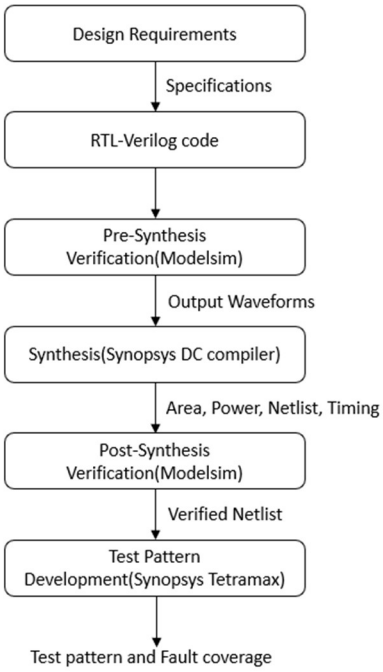

Figure 3.2: Flowchart of the Designing Booth Multiplier

Implementing a low-power, high-speed 8-bit Booth multiplier using an 8-bit adder-subtractor in Verilog presents an exciting challenge in digital design. Leveraging Verilog's capabilities, we can define the functionality of both the multiplier and the adder-subtractor in a concise and efficient manner. By carefully optimizing the design for low power consumption and high-speed operation, we aim to achieve a balance between performance and energy efficiency.

In the simulation phase using ModelSim, we'll thoroughly test the functionality of our design, ensuring that it meets the required specifications and operates correctly under various input conditions [2]. ModelSim provides a powerful environment for simulating digital circuits, allowing us to verify the correctness of our Verilog code and identify any potential issues or bugs before proceeding to synthesis.

Synthesis using Synopsys DC compiler involves translating our Verilog code into a gate-level netlist optimized for the target technology. During this phase, we'll focus on optimizing for both speed and power, utilizing synthesis directives and constraints to guide the tool towards achieving our design goals. By leveraging the capabilities of Synopsys DC compiler, we can explore various optimization techniques and trade-offs to achieve the desired balance between power consumption and performance [3].

Finally, test pattern generation using Synopsys TetraMAX enables us to generate comprehensive test patterns to validate the functionality and reliability of our design post-synthesis. TetraMAX employs advanced algorithms to generate efficient test patterns that achieve high fault coverage, ensuring the robustness of our design in real-world applications [4].

Through this integrated approach, combining Verilog design, simulation with ModelSim, synthesis with Synopsys DC compiler, and test pattern generation with Synopsys TetraMAX, we aim to develop a low-

power, high-speed 8-bit Booth multiplier that meets the stringent requirements of modern digital systems. This project showcases the power of Verilog and industry-standard tools in enabling efficient and reliable digital design implementations [5].

E. Software used

ModelSim-Intel® FPGA Standard Edition, Version 18.1: This tool is great for pre-synthesis verification, allowing you to simulate your Verilog code and ensure that it behaves as expected before moving on to synthesis. You can verify functionality and debug any issues in your design using ModelSim's simulation capabilities.

Synopsys Design Compiler® RTL synthesis: Synopsys DC Compiler is a powerful tool for synthesis, which translates your Verilog code into a gate-level netlist optimized for the target technology. It also generates reports on area, power, timing, and cells, providing valuable insights into the synthesized design's characteristics.

Synopsys TetraMAX®: TetraMAX is used for test pattern generation, ensuring that your synthesized design is thoroughly tested for faults and errors. By generating comprehensive test patterns, TetraMAX helps validate the functionality and reliability of your design post-synthesis.

IV. RESULTS AND DISCUSSION

A. Pre-Synthesis Verification

The Verilog code has been written for multiplier module and compiled in Modelsim. The compiled code has no errors, and a test bench is developed for the multiplier module. This testbench also compiled successfully with no errors. Now we are going to simulate the multiplier using the testbench module and check the output waveforms. This waveform shows that developed RTL is correct or not. If we get proper multiplier outputs from this module, then this module can be processed to next phase. Otherwise, the code needs to be checked once again. The multiplier module has been named as booth.v, and testbench has been named booth.vt.

Below is the output waveform of booth multiplier, the z shows output and a, b are inputs. The outputs shown below are a)5*6=30, b)6*7=42, c)9*7=63, d)15*15=225, e)30*30=900, f)25*25=625. This shows the outputs are proper and the module is reliable.

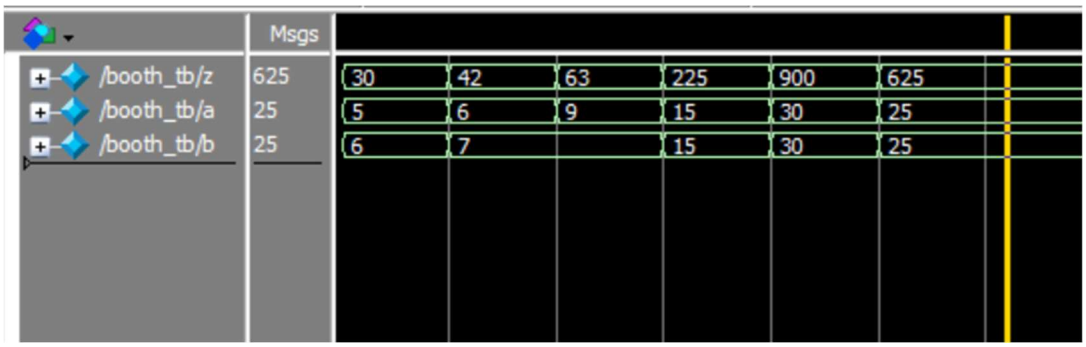

Figure 4.1: Pre-synthesis verification waveforms of multiplier

B. Synthesis

The Verilog code and its testbench are used in this step. 2 new files will be created with the previous files called booth.vfs and booth.vfv. booth.vfs file include only main module and submodules (no testbench). booth.vfv includes all modules, submodules and testbench. Now these will be used to synthesize in Synopsys DC compiler. There are some DC files in the DC files and examples folder, we need to import these files and use “make NAME=booth synth” command to synthesize the module. This generates area, time, power, netlist, cell reports for the multiplier.

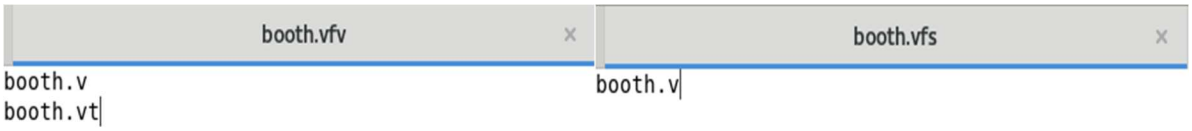

Figure 4.2: vfv and vfs files

The area report for the Booth multiplier design indicates a total of 32 ports, 429 nets, and 413 cells, all of which are combinational. There are no sequential cells, macros, or black boxes present in the design. The combinational area is measured at 490.77, with an additional 35.38 for buffer/inverter gates. The non-combinational and macro/black box areas are both reported as zero. However, the total area is listed as undefined, possibly due to unspecified interconnect area. Below is the area report for multiplier.

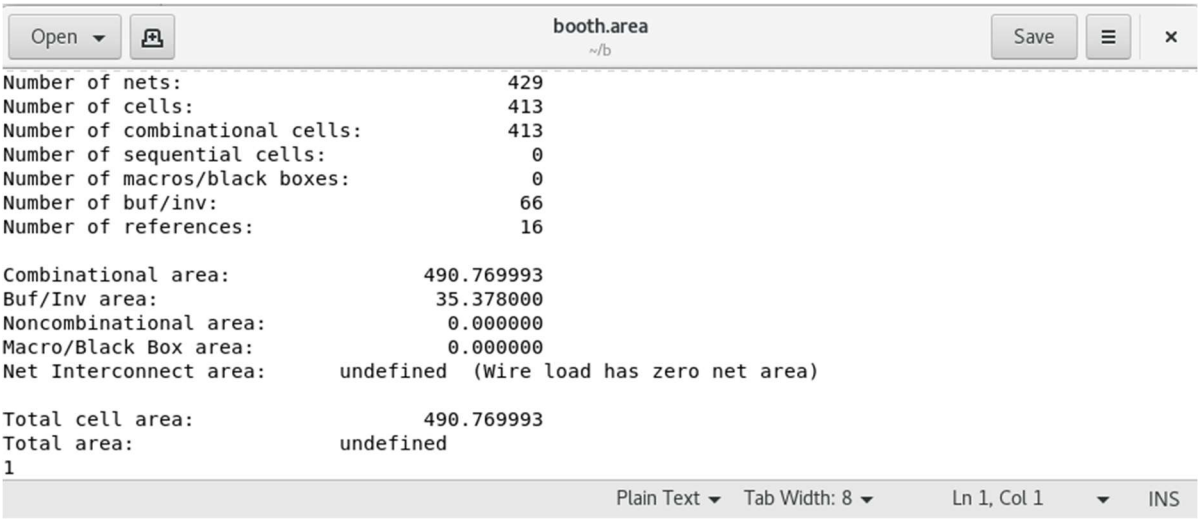

Figure 4.3: Booth Multiplier area report

The power report is also generated, this shows total power is 716.13 µw. Below image shows combinational power including internal, switching, leakage power.

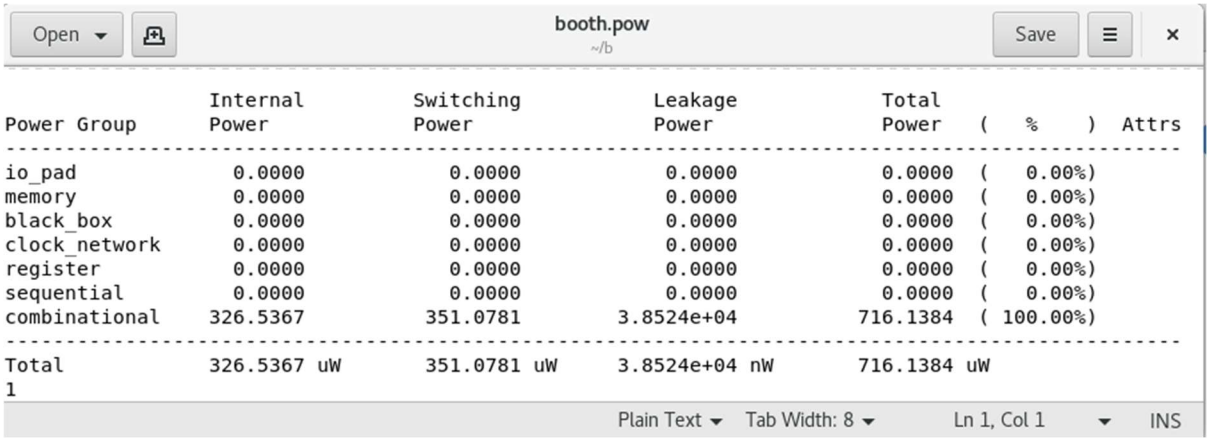

Figure 4.4: Booth Multiplier power report

Below is the timing report for the booth multiplier. It includes details such as propagation delays, setup and hold times, and critical paths, which are essential for ensuring the proper functioning of the design within specified timing constraints.

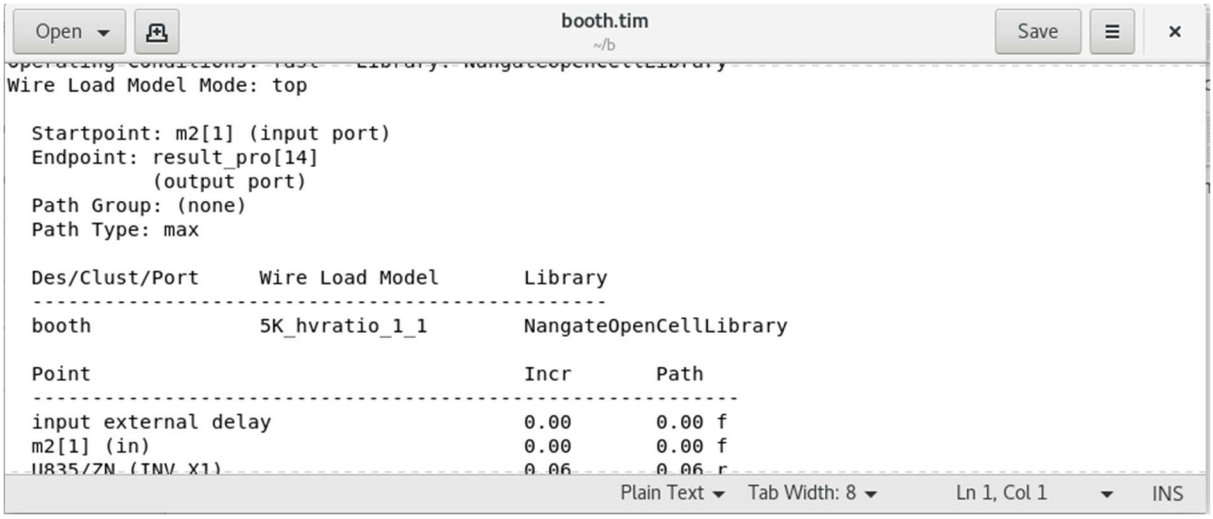

Figure 4.5: Booth Multiplier timing report

This below image shows the netlist for the multiplier. This image shows the code changed to a gate level netlist. The netlist for the multiplier describes how the various components and connections within the circuit are organized and interconnected.

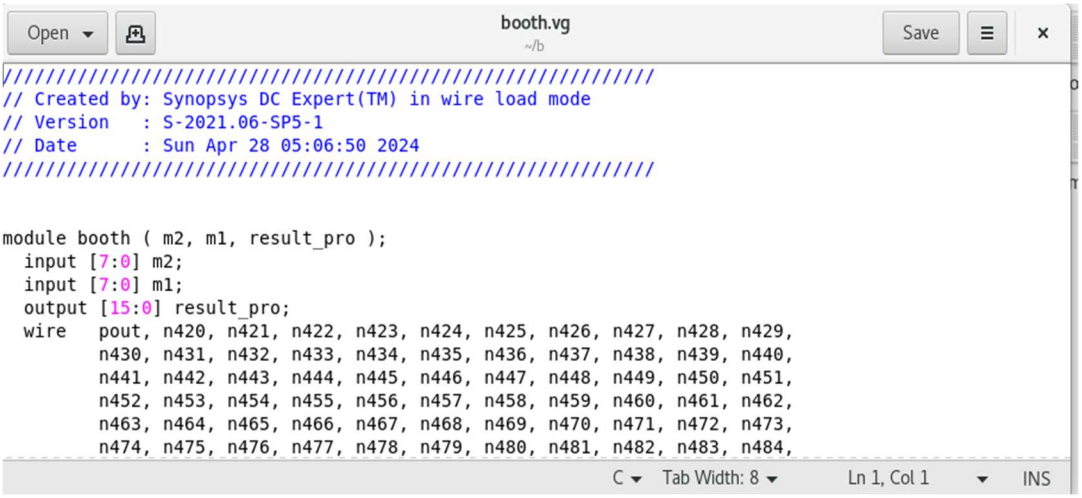

Figure 4.6: Booth Multiplier netlist

C. Post-Synthesis Verification

The netlist is again compiled in modelsim, we also import NangateOpenCellLibrary.v file to modelsim and compiler everything. Now we will modify the testbench with netlist module name and simulate this again. So that we will verify our model again. The figure 4.7 shows the schematic of the Netlist. The outputs shown in the figure 4.8 are same as the pre synthesis outputs. Now we can proceed to next step.

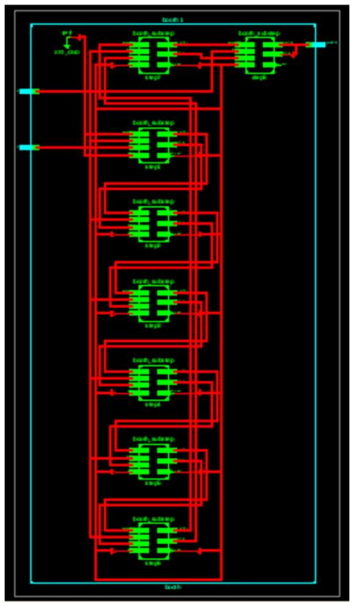

Figure 4.7: Schematic of multiplier

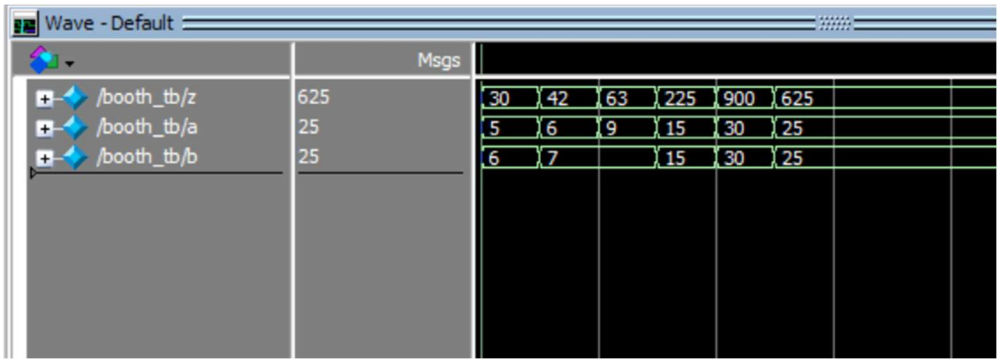

Figure 4.8: Post-synthesis verification waveforms of multiplier

D. Test Pattern Generation

We used the Synopsys Tetramax tool to generate test pattern. First we load the netlist into tetramax, we also load NangateOpenCellLibrary.v. After loading we build the model, then we check it with DRC. Then we will use ATPG to generate test patterns. This shows no of detected faults and undetected faults with fault coverage. Next, we will write the patterns into a STIL file.

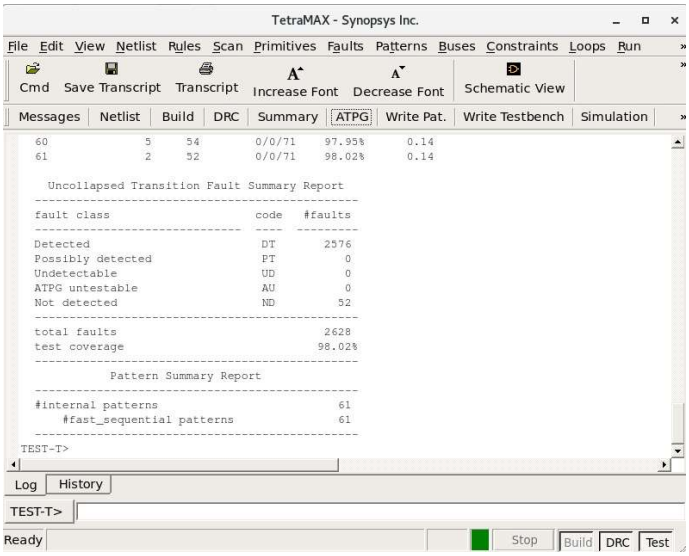

Figure 4.9: Pattern Generation Report

The provided STIL output contains information related to fault analysis and test coverage for a design. It includes a summary of various fault classes, such as Detected (DT), Possibly detected (PT), Undetectable (UD), ATPG untestable (AU), and not detected (ND). The report indicates that 2617 faults were detected, while 11 faults were not detected out of a total of 2628 faults analyzed. The test coverage achieved is reported as 98.02%. The test patterns generated shows terms “pi”- primary input and “po”- primary output. So here these input patterns are having 16 bits which are 2 inputs of 8 bit. Ex: pi-0010101100011000 is primary input and po-LLLLLHLLLLLLHLLL is the corresponding primary output for that combination. We will use these patterns for testing the chip. Total 61 patterns were generated to test all the faults in the multiplier circuit. After implementing a physical chip for this module. We only need these 61 patterns to find whether it is faulty or not.

V. CONCLUSION

In conclusion, the design process for the multiplier using Verilog has been successfully completed. The Verilog code underwent thorough simulation in ModelSim, ensuring its functionality and correctness without any errors. Subsequently, the Verilog code was synthesized using Synopsys Design Compiler, and comprehensive reports on area, cell, power, and timing were analyzed to assess the design's characteristics.

The netlist generated through synthesis was further verified to ensure that the design specifications were accurately translated into hardware implementation. Additionally, test patterns were generated using TetraMAX, providing a set of comprehensive tests to validate the functionality of the multiplier. With these steps completed, the multiplier design is ready for further stages in the manufacturing process. The generated test patterns will be instrumental in identifying any faulty chips post-manufacturing, ensuring the reliability and functionality of the multiplier in real-world applications.

Overall, the successful completion of these design and verification steps demonstrates the robustness and effectiveness of the multiplier design process, showcasing the capabilities of the multiplier use and industry-standard tools in digital design.

VI. FUTURE SCOPE

Exploring integration opportunities with emerging technologies such as neuromorphic computing, quantum computing, or approximate computing can open new avenues for innovation. Investigating how the Booth multiplier can be adapted or enhanced to leverage the unique capabilities of these technologies can lead to breakthroughs in computational efficiency and performance. Further optimization of the Booth multiplier algorithm and its hardware implementation could lead to improvements in speed and efficiency. Techniques such as pipelining, parallel processing, and algorithmic refinements can be explored to enhance performance metrics such as throughput and latency.

The future scope for the Booth multiplier designed above encompasses a wide range of possibilities, including performance optimization, power efficiency, area reduction, error resilience, HLS exploration, application-specific customization [3], and integration with emerging technologies. Continued research and innovation in these areas have the potential to advance the state-of-the-art in digital multiplication algorithms and enable new capabilities in computing systems.

REFERENCES

[1] Design of an Efficient High-Speed Radix-4 Booth Multiplier for both Signed and Unsigned Numbers. D. Kalaiyarasi, M. Saraswathi, 2018 Fourth International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB).

[2] Design and implementation of high speed modified booth multiplier using hybrid adder. Divya Govekar, Ameeta Amonkar, 2017 International Conference on Computing Methodologies and Communication (ICCMC).

[3] https://www.javatpoint.com/booths-multiplication-algorithm-in-coa.

[4] A well-structured modified Booth multiplier design. Li-Rong Wang, Shyh-Jye Jou;Chung-Len Lee, 2008 IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT).

[5] Testing and Verification of 8-Bit Full Adder. Mehmet Cetin, Karthik Velagapudi, Venkateshwar Reddy Enukonda, California State University, Fresno-Lyles College of Engineering, Electrical and Computer Engineering Department.

APPENDIX

A. Multiplier Verilog Code

```
module xorgate2 (input wire input0, input1, output wire o);
  assign o = input0 ^ input1;
endmodule

module fulladder ( input wire input0, input1, carryin, output wire sumout, carryout);
   assign sumout = input0 ^ input1 ^ carryin;
   assign carryout = (input0 & input1) | (input1 & carryin) | (carryin & input0);
endmodule

module eightbit_addsub(
   input wire carryin,
   input wire [7:0] input0,input1,
   output wire [7:0] sumout);
         wire carryout;
         wire [7:0] temporary;
         wire [7:0] i_p_int;
   xorgate2 x0 (input1[0], carryin, i_p_int[0]);
   xorgate2 x1 (input1[1], carryin, i_p_int[1]);
   xorgate2 x2 (input1[2], carryin, i_p_int[2]);
   xorgate2 x3 (input1[3], carryin, i_p_int[3]);
   xorgate2 x4 (input1[4], carryin, i_p_int[4]);
   xorgate2 x5 (input1[5], carryin, i_p_int[5]);
   xorgate2 x6 (input1[6], carryin, i_p_int[6]);
   xorgate2 x7 (input1[7], carryin, i_p_int[7]);

         fulladder fulladder1(input0[0], i_p_int[0], carryin,    sumout[0], temporary[0]);
         fulladder fulladder2(input0[1], i_p_int[1], temporary[0], sumout[1], temporary[1]);
         fulladder fulladder3(input0[2], i_p_int[2], temporary[1], sumout[2], temporary[2]);
         fulladder fulladder4(input0[3], i_p_int[3], temporary[2], sumout[3], temporary[3]);
         fulladder fulladder5(input0[4], i_p_int[4], temporary[3], sumout[4], temporary[4]);
         fulladder fulladder6(input0[5], i_p_int[5], temporary[4], sumout[5], temporary[5]);
         fulladder fulladder7(input0[6], i_p_int[6], temporary[5], sumout[6], temporary[6]);
         fulladder fulladder8(input0[7], i_p_int[7], temporary[6], sumout[7], carryout);

endmodule


module booth_substep(
   input wire signed [7:0] accum,
```

```
   input wire signed [7:0] P,
   input wire signed p0,
   input wire signed [7:0] m1,
   output reg signed [7:0] accum_q,
   output reg signed [7:0] next_P,
   output reg p0_next);

         wire [7:0] AS_temp;

         eightbit_addsub myadd(P[0], accum, m1, AS_temp);

                  always @(*) begin
                  if(P[0] == p0) begin
        p0_next = P[0];
        next_P = P>>1;
        next_P[7] = accum[0];

        accum_q = accum>>1;

                           if (accum[7] == 1)
          accum_q[7] = 1;
                  end

                  else begin
        p0_next = P[0];
        next_P = P>>1;
        next_P[7] = AS_temp[0];

        accum_q = AS_temp>>1;

                           if (AS_temp[7] == 1)
          accum_q[7] = 1;
                  end
end
endmodule
module booth(
   input signed[7:0] m2, m1,
   output signed [15:0] result_pro);
         wire signed [7:0] P[0:6];
         wire signed [7:0] accum[0:7];
         wire signed[7:0] p0;
         wire pout;
```

```
        assign accum[0] = 8'b00000000;
        booth_substep step1(accum[0], m2, 1'b0, m1, accum[1],      P[0],       p0[1]);
        booth_substep step2(accum[1], P[0],     p0[1], m1, accum[2],      P[1],       p0[2]);
        booth_substep step3(accum[2], P[1],     p0[2], m1, accum[3],      P[2],       p0[3]);
        booth_substep step4(accum[3], P[2],     p0[3], m1, accum[4],      P[3],       p0[4]);
        booth_substep step5(accum[4], P[3],     p0[4], m1, accum[5],      P[4],       p0[5]);
        booth_substep step6(accum[5], P[4],     p0[5], m1, accum[6],      P[5],       p0[6]);
        booth_substep step7(accum[6], P[5],     p0[6], m1, accum[7],      P[6],       p0[7]);
        booth_substep step8(accum[7], P[6],     p0[7], m1, result_pro[15:8], result_pro[7:0], pout);
endmodule
```

B. Testbench

```
module booth_tb;
wire signed [15:0] z;
reg signed [7:0] a,b;
booth my_booth(.m2(a),.m1(b),.result_pro(z));
initial begin
a = 5; b = 6;
   #10 a = 6; b = 7;
   #10 a = 9; b = 7;
   #10 a = 15; b = 15;
   #10 a = 30; b = 30;
   #10 a = 25; b = 25;
end
endmodule
```

C. Netlist

```
////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : S-2021.06-SP5-1
// Date      : Sun Apr 28 05:06:50 2024
////////////////////////////////////////////////////////////


module booth ( m2, m1, result_pro );
 input [7:0] m2;
 input [7:0] m1;
 output [15:0] result_pro;
 wire   pout, n420, n421, n422, n423, n424, n425, n426, n427, n428, n429,
       n430, n431, n432, n433, n434, n435, n436, n437, n438, n439, n440,
       n441, n442, n443, n444, n445, n446, n447, n448, n449, n450, n451,
       n452, n453, n454, n455, n456, n457, n458, n459, n460, n461, n462,
       n463, n464, n465, n466, n467, n468, n469, n470, n471, n472, n473,
```

```
    n474, n475, n476, n477, n478, n479, n480, n481, n482, n483, n484,
    n485, n486, n487, n488, n489, n490, n491, n492, n493, n494, n495,
    n496, n497, n498, n499, n500, n501, n502, n503, n504, n505, n506,
    n507, n508, n509, n510, n511, n512, n513, n514, n515, n516, n517,
    n518, n519, n520, n521, n522, n523, n524, n525, n526, n527, n528,
    n529, n530, n531, n532, n533, n534, n535, n536, n537, n538, n539,
    n540, n541, n542, n543, n544, n545, n546, n547, n548, n549, n550,
    n551, n552, n553, n554, n555, n556, n557, n558, n559, n560, n561,
    n562, n563, n564, n565, n566, n567, n568, n569, n570, n571, n572,
    n573, n574, n575, n576, n577, n578, n579, n580, n581, n582, n583,
    n584, n585, n586, n587, n588, n589, n590, n591, n592, n593, n594,
    n595, n596, n597, n598, n599, n600, n601, n602, n603, n604, n605,
    n606, n607, n608, n609, n610, n611, n612, n613, n614, n615, n616,
    n617, n618, n619, n620, n621, n622, n623, n624, n625, n626, n627,
    n628, n629, n630, n631, n632, n633, n634, n635, n636, n637, n638,
    n639, n640, n641, n642, n643, n644, n645, n646, n647, n648, n649,
    n650, n651, n652, n653, n654, n655, n656, n657, n658, n659, n660,
    n661, n662, n663, n664, n665, n666, n667, n668, n669, n670, n671,
    n672, n673, n674, n675, n676, n677, n678, n679, n680, n681, n682,
    n683, n684, n685, n686, n687, n688, n689, n690, n691, n692, n693,
    n694, n695, n696, n697, n698, n699, n700, n701, n702, n703, n704,
    n705, n706, n707, n708, n709, n710, n711, n712, n713, n714, n715,
    n716, n717, n718, n719, n720, n721, n722, n723, n724, n725, n726,
    n727, n728, n729, n730, n731, n732, n733, n734, n735, n736, n737,
    n738, n739, n740, n741, n742, n743, n744, n745, n746, n747, n748,
    n749, n750, n751, n752, n753, n754, n755, n756, n757, n758, n759,
    n760, n761, n762, n763, n764, n765, n766, n767, n768, n769, n770,
    n771, n772, n773, n774, n775, n776, n777, n778, n779, n780, n781,
    n782, n783, n784, n785, n786, n787, n788, n789, n790, n791, n792,
    n793, n794, n795, n796, n797, n798, n799, n800, n801, n802, n803,
    n804, n805, n806, n807, n808, n809, n810, n811, n812, n813, n814,
    n815, n816;
wire   [7:1] p0;
assign p0[7] = m2[6];
assign p0[6] = m2[5];
assign p0[5] = m2[4];
assign p0[4] = m2[3];
assign p0[3] = m2[2];
assign p0[2] = m2[1];
assign p0[1] = m2[0];
assign pout = m2[7];
```

```
BUF_X1 U435 ( .A(result_pro[15]), .Z(result_pro[14]) );
XOR2_X1 U436 ( .A(n420), .B(n421), .Z(result_pro[9]) );
NAND2_X1 U437 ( .A1(n422), .A2(n423), .ZN(n420) );
XOR2_X1 U438 ( .A(n424), .B(n425), .Z(n422) );
XNOR2_X1 U439 ( .A(n426), .B(n427), .ZN(result_pro[8]) );
NAND2_X1 U440 ( .A1(n428), .A2(n423), .ZN(n426) );
XNOR2_X1 U441 ( .A(n429), .B(n430), .ZN(n428) );
XOR2_X1 U442 ( .A(n431), .B(n432), .Z(result_pro[7]) );
NOR2_X1 U443 ( .A1(n433), .A2(n434), .ZN(n431) );
XNOR2_X1 U444 ( .A(pout), .B(n435), .ZN(n434) );
XOR2_X1 U445 ( .A(n436), .B(n437), .Z(result_pro[6]) );
NOR2_X1 U446 ( .A1(n438), .A2(n439), .ZN(n436) );
XNOR2_X1 U447 ( .A(p0[7]), .B(n440), .ZN(n439) );
XOR2_X1 U448 ( .A(n441), .B(n442), .Z(result_pro[5]) );
NOR2_X1 U449 ( .A1(n443), .A2(n444), .ZN(n441) );
XNOR2_X1 U450 ( .A(p0[6]), .B(n445), .ZN(n444) );
INV_X1 U451 ( .A(n446), .ZN(n443) );
XOR2_X1 U452 ( .A(n447), .B(n448), .Z(result_pro[4]) );
NOR2_X1 U453 ( .A1(n449), .A2(n450), .ZN(n447) );
XNOR2_X1 U454 ( .A(p0[5]), .B(n451), .ZN(n450) );
INV_X1 U455 ( .A(n452), .ZN(n449) );
XNOR2_X1 U456 ( .A(n453), .B(n454), .ZN(result_pro[3]) );
NAND2_X1 U457 ( .A1(n455), .A2(n456), .ZN(n453) );
XNOR2_X1 U458 ( .A(n457), .B(n458), .ZN(n455) );
XNOR2_X1 U459 ( .A(n459), .B(n460), .ZN(result_pro[2]) );
NAND2_X1 U460 ( .A1(n461), .A2(n462), .ZN(n459) );
XNOR2_X1 U461 ( .A(n463), .B(n464), .ZN(n461) );
OAI21_X1 U462 ( .B1(n465), .B2(n466), .A(n467), .ZN(result_pro[1]) );
OAI21_X1 U463 ( .B1(n465), .B2(n468), .A(n469), .ZN(n467) );
INV_X1 U464 ( .A(n470), .ZN(n469) );
MUX2_X1 U465 ( .A(n471), .B(n472), .S(n468), .Z(n466) );
NAND2_X1 U466 ( .A1(n470), .A2(n472), .ZN(n471) );
XNOR2_X1 U467 ( .A(n473), .B(n474), .ZN(result_pro[15]) );
NAND2_X1 U468 ( .A1(n475), .A2(n423), .ZN(n473) );
XOR2_X1 U469 ( .A(n476), .B(n477), .Z(n475) );
XNOR2_X1 U470 ( .A(n478), .B(m1[7]), .ZN(n477) );
OAI22_X1 U471 ( .A1(n479), .A2(n480), .B1(n481), .B2(n482), .ZN(n476) );
NOR2_X1 U472 ( .A1(n474), .A2(n483), .ZN(n481) );
XNOR2_X1 U473 ( .A(n484), .B(n480), .ZN(result_pro[13]) );
INV_X1 U474 ( .A(n474), .ZN(n480) );
XOR2_X1 U475 ( .A(n485), .B(n486), .Z(n474) );
NAND2_X1 U476 ( .A1(n487), .A2(n488), .ZN(n485) );
```

```
XOR2_X1 U477 ( .A(n489), .B(n490), .Z(n487) );
XNOR2_X1 U478 ( .A(n491), .B(m1[7]), .ZN(n490) );
OAI22_X1 U479 ( .A1(n492), .A2(n486), .B1(n493), .B2(n494), .ZN(n489) );
NOR2_X1 U480 ( .A1(n495), .A2(n496), .ZN(n493) );
INV_X1 U481 ( .A(n496), .ZN(n486) );
NOR2_X1 U482 ( .A1(n433), .A2(n497), .ZN(n484) );
XNOR2_X1 U483 ( .A(n479), .B(n482), .ZN(n497) );
OAI21_X1 U484 ( .B1(n498), .B2(n499), .A(n500), .ZN(n482) );
INV_X1 U485 ( .A(n501), .ZN(n500) );
AOI21_X1 U486 ( .B1(n499), .B2(n498), .A(n502), .ZN(n501) );
INV_X1 U487 ( .A(n483), .ZN(n479) );
XNOR2_X1 U488 ( .A(m1[6]), .B(n478), .ZN(n483) );
INV_X1 U489 ( .A(n423), .ZN(n433) );
XNOR2_X1 U490 ( .A(n503), .B(n498), .ZN(result_pro[12]) );
XOR2_X1 U491 ( .A(n496), .B(n504), .Z(n498) );
NOR2_X1 U492 ( .A1(n438), .A2(n505), .ZN(n504) );
XNOR2_X1 U493 ( .A(n492), .B(n494), .ZN(n505) );
OAI22_X1 U494 ( .A1(n506), .A2(n507), .B1(n508), .B2(n509), .ZN(n494) );
AND2_X1 U495 ( .A1(n506), .A2(n507), .ZN(n508) );
INV_X1 U496 ( .A(n495), .ZN(n492) );
XNOR2_X1 U497 ( .A(m1[6]), .B(n491), .ZN(n495) );
INV_X1 U498 ( .A(n488), .ZN(n438) );
XOR2_X1 U499 ( .A(n510), .B(n511), .Z(n496) );
NAND2_X1 U500 ( .A1(n512), .A2(n446), .ZN(n510) );
XOR2_X1 U501 ( .A(n513), .B(n514), .Z(n512) );
XNOR2_X1 U502 ( .A(n515), .B(m1[7]), .ZN(n514) );
OAI22_X1 U503 ( .A1(n516), .A2(n511), .B1(n517), .B2(n518), .ZN(n513) );
AND2_X1 U504 ( .A1(n516), .A2(n511), .ZN(n517) );
NAND2_X1 U505 ( .A1(n519), .A2(n423), .ZN(n503) );
XOR2_X1 U506 ( .A(n502), .B(n499), .Z(n519) );
XNOR2_X1 U507 ( .A(m1[5]), .B(n478), .ZN(n499) );
AOI21_X1 U508 ( .B1(n520), .B2(n521), .A(n522), .ZN(n502) );
INV_X1 U509 ( .A(n523), .ZN(n522) );
OAI21_X1 U510 ( .B1(n521), .B2(n520), .A(n524), .ZN(n523) );
XOR2_X1 U511 ( .A(n525), .B(n520), .Z(result_pro[11]) );
XOR2_X1 U512 ( .A(n526), .B(n507), .Z(n520) );
XOR2_X1 U513 ( .A(n527), .B(n511), .Z(n507) );
XNOR2_X1 U514 ( .A(n528), .B(n529), .ZN(n511) );
NAND2_X1 U515 ( .A1(n530), .A2(n452), .ZN(n528) );
XOR2_X1 U516 ( .A(n531), .B(n532), .Z(n530) );
XNOR2_X1 U517 ( .A(n533), .B(m1[7]), .ZN(n532) );
OAI21_X1 U518 ( .B1(n529), .B2(n534), .A(n535), .ZN(n531) );
```

```
INV_X1 U519 ( .A(n536), .ZN(n535) );
AOI21_X1 U520 ( .B1(n534), .B2(n529), .A(n537), .ZN(n536) );
NAND2_X1 U521 ( .A1(n538), .A2(n446), .ZN(n527) );
XOR2_X1 U522 ( .A(n518), .B(n516), .Z(n538) );
XOR2_X1 U523 ( .A(n539), .B(p0[6]), .Z(n516) );
OAI21_X1 U524 ( .B1(n540), .B2(n541), .A(n542), .ZN(n518) );
INV_X1 U525 ( .A(n543), .ZN(n542) );
AOI21_X1 U526 ( .B1(n540), .B2(n541), .A(n544), .ZN(n543) );
NAND2_X1 U527 ( .A1(n545), .A2(n488), .ZN(n526) );
XOR2_X1 U528 ( .A(n509), .B(n506), .Z(n545) );
XNOR2_X1 U529 ( .A(m1[5]), .B(n491), .ZN(n506) );
OAI21_X1 U530 ( .B1(n546), .B2(n547), .A(n548), .ZN(n509) );
INV_X1 U531 ( .A(n549), .ZN(n548) );
AOI21_X1 U532 ( .B1(n547), .B2(n546), .A(n550), .ZN(n549) );
NAND2_X1 U533 ( .A1(n551), .A2(n423), .ZN(n525) );
XOR2_X1 U534 ( .A(n524), .B(n521), .Z(n551) );
XNOR2_X1 U535 ( .A(m1[4]), .B(pout), .ZN(n521) );
AOI22_X1 U536 ( .A1(n552), .A2(n553), .B1(n554), .B2(n555), .ZN(n524) );
OR2_X1 U537 ( .A1(n553), .A2(n552), .ZN(n554) );
XNOR2_X1 U538 ( .A(n556), .B(n553), .ZN(result_pro[10]) );
XOR2_X1 U539 ( .A(n557), .B(n546), .Z(n553) );
XOR2_X1 U540 ( .A(n558), .B(n541), .Z(n546) );
XOR2_X1 U541 ( .A(n559), .B(n529), .Z(n541) );
XNOR2_X1 U542 ( .A(n560), .B(n561), .ZN(n529) );
NAND2_X1 U543 ( .A1(n562), .A2(n456), .ZN(n560) );
XOR2_X1 U544 ( .A(n563), .B(n564), .Z(n562) );
XNOR2_X1 U545 ( .A(n457), .B(m1[7]), .ZN(n564) );
OAI22_X1 U546 ( .A1(n565), .A2(n561), .B1(n566), .B2(n567), .ZN(n563) );
AND2_X1 U547 ( .A1(n565), .A2(n561), .ZN(n566) );
NAND2_X1 U548 ( .A1(n568), .A2(n452), .ZN(n559) );
XOR2_X1 U549 ( .A(n534), .B(n537), .Z(n568) );
XNOR2_X1 U550 ( .A(m1[6]), .B(p0[5]), .ZN(n537) );
OAI22_X1 U551 ( .A1(n569), .A2(n570), .B1(n571), .B2(n572), .ZN(n534) );
AND2_X1 U552 ( .A1(n569), .A2(n570), .ZN(n571) );
NAND2_X1 U553 ( .A1(n573), .A2(n446), .ZN(n558) );
XOR2_X1 U554 ( .A(n540), .B(n544), .Z(n573) );
XNOR2_X1 U555 ( .A(m1[5]), .B(n515), .ZN(n544) );
OAI22_X1 U556 ( .A1(n574), .A2(n575), .B1(n576), .B2(n577), .ZN(n540) );
AND2_X1 U557 ( .A1(n574), .A2(n575), .ZN(n576) );
NAND2_X1 U558 ( .A1(n578), .A2(n488), .ZN(n557) );
XOR2_X1 U559 ( .A(n547), .B(n550), .Z(n578) );
XNOR2_X1 U560 ( .A(n579), .B(n491), .ZN(n550) );
```

```
OAI21_X1 U561 ( .B1(n580), .B2(n581), .A(n582), .ZN(n547) );
INV_X1 U562 ( .A(n583), .ZN(n582) );
AOI21_X1 U563 ( .B1(n580), .B2(n581), .A(n584), .ZN(n583) );
NAND2_X1 U564 ( .A1(n585), .A2(n423), .ZN(n556) );
XNOR2_X1 U565 ( .A(pout), .B(n491), .ZN(n423) );
XOR2_X1 U566 ( .A(n552), .B(n555), .Z(n585) );
XNOR2_X1 U567 ( .A(m1[3]), .B(n478), .ZN(n555) );
OAI21_X1 U568 ( .B1(n425), .B2(n421), .A(n586), .ZN(n552) );
INV_X1 U569 ( .A(n587), .ZN(n586) );
AOI21_X1 U570 ( .B1(n425), .B2(n421), .A(n424), .ZN(n587) );
OAI21_X1 U571 ( .B1(n429), .B2(n427), .A(n588), .ZN(n424) );
OAI21_X1 U572 ( .B1(n589), .B2(n590), .A(n430), .ZN(n588) );
AOI22_X1 U573 ( .A1(n435), .A2(n432), .B1(n591), .B2(pout), .ZN(n430) );
OR2_X1 U574 ( .A1(n432), .A2(n435), .ZN(n591) );
XOR2_X1 U575 ( .A(n592), .B(n593), .Z(n432) );
NAND2_X1 U576 ( .A1(n594), .A2(n488), .ZN(n592) );
XOR2_X1 U577 ( .A(n595), .B(n596), .Z(n594) );
XNOR2_X1 U578 ( .A(m1[0]), .B(n478), .ZN(n435) );
INV_X1 U579 ( .A(n429), .ZN(n590) );
INV_X1 U580 ( .A(n589), .ZN(n427) );
XOR2_X1 U581 ( .A(n597), .B(n598), .Z(n589) );
NAND2_X1 U582 ( .A1(n599), .A2(n488), .ZN(n597) );
XOR2_X1 U583 ( .A(n600), .B(n601), .Z(n599) );
XNOR2_X1 U584 ( .A(m1[1]), .B(n478), .ZN(n429) );
XOR2_X1 U585 ( .A(n602), .B(n581), .Z(n421) );
XOR2_X1 U586 ( .A(n603), .B(n575), .Z(n581) );
XOR2_X1 U587 ( .A(n604), .B(n570), .Z(n575) );
XOR2_X1 U588 ( .A(n605), .B(n561), .Z(n570) );
XNOR2_X1 U589 ( .A(n606), .B(n607), .ZN(n561) );
NAND2_X1 U590 ( .A1(n608), .A2(n462), .ZN(n606) );
XOR2_X1 U591 ( .A(n609), .B(n610), .Z(n608) );
AOI22_X1 U592 ( .A1(n611), .A2(n612), .B1(n613), .B2(n614), .ZN(n610) );
OR2_X1 U593 ( .A1(n613), .A2(n614), .ZN(n612) );
INV_X1 U594 ( .A(n607), .ZN(n613) );
INV_X1 U595 ( .A(n615), .ZN(n611) );
XNOR2_X1 U596 ( .A(m1[7]), .B(p0[3]), .ZN(n609) );
NAND2_X1 U597 ( .A1(n616), .A2(n456), .ZN(n605) );
XOR2_X1 U598 ( .A(n567), .B(n565), .Z(n616) );
XOR2_X1 U599 ( .A(n539), .B(p0[4]), .Z(n565) );
OAI22_X1 U600 ( .A1(n617), .A2(n618), .B1(n619), .B2(n620), .ZN(n567) );
AND2_X1 U601 ( .A1(n617), .A2(n618), .ZN(n619) );
NAND2_X1 U602 ( .A1(n621), .A2(n452), .ZN(n604) );
```

```
XOR2_X1 U603 ( .A(n572), .B(n569), .Z(n621) );
XOR2_X1 U604 ( .A(m1[5]), .B(p0[5]), .Z(n569) );
OAI22_X1 U605 ( .A1(n622), .A2(n623), .B1(n624), .B2(n625), .ZN(n572) );
NOR2_X1 U606 ( .A1(n626), .A2(n627), .ZN(n624) );
INV_X1 U607 ( .A(n626), .ZN(n622) );
NAND2_X1 U608 ( .A1(n628), .A2(n446), .ZN(n603) );
XOR2_X1 U609 ( .A(n577), .B(n574), .Z(n628) );
XOR2_X1 U610 ( .A(n579), .B(p0[6]), .Z(n574) );
OAI21_X1 U611 ( .B1(n629), .B2(n630), .A(n631), .ZN(n577) );
OAI21_X1 U612 ( .B1(n632), .B2(n633), .A(n634), .ZN(n631) );
INV_X1 U613 ( .A(n630), .ZN(n633) );
NAND2_X1 U614 ( .A1(n635), .A2(n488), .ZN(n602) );
XNOR2_X1 U615 ( .A(p0[6]), .B(n491), .ZN(n488) );
XOR2_X1 U616 ( .A(n584), .B(n580), .Z(n635) );
XNOR2_X1 U617 ( .A(m1[3]), .B(n491), .ZN(n580) );
OAI21_X1 U618 ( .B1(n636), .B2(n637), .A(n638), .ZN(n584) );
OAI21_X1 U619 ( .B1(n601), .B2(n598), .A(n600), .ZN(n638) );
AOI22_X1 U620 ( .A1(n595), .A2(n593), .B1(n639), .B2(n596), .ZN(n600) );
AOI22_X1 U621 ( .A1(n440), .A2(n437), .B1(n640), .B2(p0[7]), .ZN(n596) );
OR2_X1 U622 ( .A1(n437), .A2(n440), .ZN(n640) );
XOR2_X1 U623 ( .A(n641), .B(n642), .Z(n437) );
NAND2_X1 U624 ( .A1(n643), .A2(n446), .ZN(n641) );
XOR2_X1 U625 ( .A(n644), .B(n645), .Z(n643) );
XNOR2_X1 U626 ( .A(m1[0]), .B(n491), .ZN(n440) );
INV_X1 U627 ( .A(p0[7]), .ZN(n491) );
OR2_X1 U628 ( .A1(n595), .A2(n593), .ZN(n639) );
XOR2_X1 U629 ( .A(n646), .B(n647), .Z(n593) );
NAND2_X1 U630 ( .A1(n648), .A2(n446), .ZN(n646) );
XOR2_X1 U631 ( .A(n649), .B(n650), .Z(n648) );
XOR2_X1 U632 ( .A(n651), .B(p0[7]), .Z(n595) );
INV_X1 U633 ( .A(n637), .ZN(n598) );
INV_X1 U634 ( .A(n636), .ZN(n601) );
XOR2_X1 U635 ( .A(n652), .B(n630), .Z(n637) );
XOR2_X1 U636 ( .A(n653), .B(n623), .Z(n630) );
INV_X1 U637 ( .A(n627), .ZN(n623) );
XOR2_X1 U638 ( .A(n618), .B(n654), .Z(n627) );
NOR2_X1 U639 ( .A1(n655), .A2(n656), .ZN(n654) );
XNOR2_X1 U640 ( .A(n620), .B(n617), .ZN(n656) );
XNOR2_X1 U641 ( .A(m1[5]), .B(n457), .ZN(n617) );
OAI21_X1 U642 ( .B1(n657), .B2(n658), .A(n659), .ZN(n620) );
OAI21_X1 U643 ( .B1(n660), .B2(n661), .A(n662), .ZN(n659) );
INV_X1 U644 ( .A(n456), .ZN(n655) );
```

```
XOR2_X1 U645 ( .A(n663), .B(n607), .Z(n618) );
MUX2_X1 U646 ( .A(n664), .B(n665), .S(n465), .Z(n607) );
XOR2_X1 U647 ( .A(n666), .B(n667), .Z(n664) );
XNOR2_X1 U648 ( .A(n668), .B(n665), .ZN(n667) );
OAI21_X1 U649 ( .B1(n669), .B2(n665), .A(n670), .ZN(n668) );
INV_X1 U650 ( .A(n671), .ZN(n670) );
AOI21_X1 U651 ( .B1(n665), .B2(n669), .A(n672), .ZN(n671) );
XNOR2_X1 U652 ( .A(m1[7]), .B(p0[2]), .ZN(n666) );
NAND2_X1 U653 ( .A1(n673), .A2(n462), .ZN(n663) );
XNOR2_X1 U654 ( .A(n615), .B(n614), .ZN(n673) );
XNOR2_X1 U655 ( .A(m1[6]), .B(n463), .ZN(n614) );
OAI22_X1 U656 ( .A1(n674), .A2(n675), .B1(n676), .B2(n677), .ZN(n615) );
AND2_X1 U657 ( .A1(n674), .A2(n675), .ZN(n676) );
NAND2_X1 U658 ( .A1(n678), .A2(n452), .ZN(n653) );
XNOR2_X1 U659 ( .A(n625), .B(n626), .ZN(n678) );
XNOR2_X1 U660 ( .A(m1[4]), .B(n533), .ZN(n626) );
OAI21_X1 U661 ( .B1(n679), .B2(n680), .A(n681), .ZN(n625) );
OAI21_X1 U662 ( .B1(n682), .B2(n683), .A(n684), .ZN(n681) );
INV_X1 U663 ( .A(n680), .ZN(n683) );
NAND2_X1 U664 ( .A1(n685), .A2(n446), .ZN(n652) );
XNOR2_X1 U665 ( .A(p0[5]), .B(n515), .ZN(n446) );
XNOR2_X1 U666 ( .A(n634), .B(n629), .ZN(n685) );
INV_X1 U667 ( .A(n632), .ZN(n629) );
XOR2_X1 U668 ( .A(m1[3]), .B(n515), .Z(n632) );
AOI21_X1 U669 ( .B1(n650), .B2(n647), .A(n686), .ZN(n634) );
INV_X1 U670 ( .A(n687), .ZN(n686) );
OAI21_X1 U671 ( .B1(n650), .B2(n647), .A(n649), .ZN(n687) );
AOI21_X1 U672 ( .B1(n644), .B2(n642), .A(n688), .ZN(n649) );
INV_X1 U673 ( .A(n689), .ZN(n688) );
OAI21_X1 U674 ( .B1(n644), .B2(n642), .A(n645), .ZN(n689) );
AOI22_X1 U675 ( .A1(n445), .A2(n442), .B1(n690), .B2(p0[6]), .ZN(n645) );
OR2_X1 U676 ( .A1(n442), .A2(n445), .ZN(n690) );
XOR2_X1 U677 ( .A(n691), .B(n692), .Z(n442) );
NAND2_X1 U678 ( .A1(n693), .A2(n452), .ZN(n691) );
XOR2_X1 U679 ( .A(n694), .B(n695), .Z(n693) );
XNOR2_X1 U680 ( .A(m1[0]), .B(n515), .ZN(n445) );
INV_X1 U681 ( .A(p0[6]), .ZN(n515) );
XOR2_X1 U682 ( .A(n696), .B(n697), .Z(n642) );
NAND2_X1 U683 ( .A1(n698), .A2(n452), .ZN(n696) );
XOR2_X1 U684 ( .A(n699), .B(n700), .Z(n698) );
XOR2_X1 U685 ( .A(n651), .B(p0[6]), .Z(n644) );
XNOR2_X1 U686 ( .A(n701), .B(n680), .ZN(n647) );
```

```
XOR2_X1 U687 ( .A(n702), .B(n658), .Z(n680) );
INV_X1 U688 ( .A(n661), .ZN(n658) );
XNOR2_X1 U689 ( .A(n703), .B(n675), .ZN(n661) );
XNOR2_X1 U690 ( .A(n665), .B(n704), .ZN(n675) );
NOR2_X1 U691 ( .A1(n465), .A2(n705), .ZN(n704) );
XNOR2_X1 U692 ( .A(n669), .B(n672), .ZN(n705) );
XNOR2_X1 U693 ( .A(n539), .B(n472), .ZN(n672) );
OAI21_X1 U694 ( .B1(n706), .B2(n707), .A(n708), .ZN(n669) );
INV_X1 U695 ( .A(n709), .ZN(n708) );
AOI21_X1 U696 ( .B1(n707), .B2(n706), .A(n710), .ZN(n709) );
INV_X1 U697 ( .A(n711), .ZN(n465) );
NAND2_X1 U698 ( .A1(n712), .A2(p0[1]), .ZN(n665) );
XNOR2_X1 U699 ( .A(m1[7]), .B(n713), .ZN(n712) );
NOR2_X1 U700 ( .A1(m1[6]), .A2(n714), .ZN(n713) );
NAND2_X1 U701 ( .A1(n715), .A2(n462), .ZN(n703) );
XOR2_X1 U702 ( .A(n677), .B(n674), .Z(n715) );
XOR2_X1 U703 ( .A(m1[5]), .B(p0[3]), .Z(n674) );
OAI22_X1 U704 ( .A1(n716), .A2(n717), .B1(n718), .B2(n719), .ZN(n677) );
NOR2_X1 U705 ( .A1(n720), .A2(n721), .ZN(n718) );
INV_X1 U706 ( .A(n720), .ZN(n716) );
NAND2_X1 U707 ( .A1(n722), .A2(n456), .ZN(n702) );
XNOR2_X1 U708 ( .A(n662), .B(n657), .ZN(n722) );
INV_X1 U709 ( .A(n660), .ZN(n657) );
XNOR2_X1 U710 ( .A(m1[4]), .B(n457), .ZN(n660) );
OAI21_X1 U711 ( .B1(n723), .B2(n724), .A(n725), .ZN(n662) );
OAI21_X1 U712 ( .B1(n726), .B2(n727), .A(n728), .ZN(n725) );
INV_X1 U713 ( .A(n723), .ZN(n726) );
NAND2_X1 U714 ( .A1(n729), .A2(n452), .ZN(n701) );
XNOR2_X1 U715 ( .A(p0[4]), .B(n533), .ZN(n452) );
XNOR2_X1 U716 ( .A(n684), .B(n679), .ZN(n729) );
INV_X1 U717 ( .A(n682), .ZN(n679) );
XOR2_X1 U718 ( .A(m1[3]), .B(n533), .Z(n682) );
AOI21_X1 U719 ( .B1(n700), .B2(n697), .A(n730), .ZN(n684) );
INV_X1 U720 ( .A(n731), .ZN(n730) );
OAI21_X1 U721 ( .B1(n700), .B2(n697), .A(n699), .ZN(n731) );
AOI22_X1 U722 ( .A1(n694), .A2(n692), .B1(n732), .B2(n695), .ZN(n699) );
AOI22_X1 U723 ( .A1(n451), .A2(n448), .B1(n733), .B2(p0[5]), .ZN(n695) );
OR2_X1 U724 ( .A1(n448), .A2(n451), .ZN(n733) );
XNOR2_X1 U725 ( .A(n734), .B(n735), .ZN(n448) );
NAND2_X1 U726 ( .A1(n736), .A2(n456), .ZN(n734) );
XNOR2_X1 U727 ( .A(n737), .B(n738), .ZN(n736) );
XNOR2_X1 U728 ( .A(m1[0]), .B(n533), .ZN(n451) );
```

```
INV_X1 U729 ( .A(p0[5]), .ZN(n533) );
OR2_X1 U730 ( .A1(n694), .A2(n692), .ZN(n732) );
XNOR2_X1 U731 ( .A(n739), .B(n740), .ZN(n692) );
NAND2_X1 U732 ( .A1(n741), .A2(n456), .ZN(n739) );
XOR2_X1 U733 ( .A(n742), .B(n743), .Z(n741) );
XOR2_X1 U734 ( .A(n651), .B(p0[5]), .Z(n694) );
XOR2_X1 U735 ( .A(n744), .B(n724), .Z(n697) );
INV_X1 U736 ( .A(n727), .ZN(n724) );
XOR2_X1 U737 ( .A(n745), .B(n717), .Z(n727) );
INV_X1 U738 ( .A(n721), .ZN(n717) );
XNOR2_X1 U739 ( .A(n746), .B(n706), .ZN(n721) );
AND2_X1 U740 ( .A1(n747), .A2(p0[1]), .ZN(n706) );
XNOR2_X1 U741 ( .A(n714), .B(n539), .ZN(n747) );
INV_X1 U742 ( .A(m1[6]), .ZN(n539) );
NAND2_X1 U743 ( .A1(n748), .A2(n711), .ZN(n746) );
XOR2_X1 U744 ( .A(n707), .B(n710), .Z(n748) );
XNOR2_X1 U745 ( .A(m1[5]), .B(n472), .ZN(n710) );
OAI22_X1 U746 ( .A1(n749), .A2(n750), .B1(n751), .B2(n752), .ZN(n707) );
AND2_X1 U747 ( .A1(n750), .A2(n749), .ZN(n751) );
NAND2_X1 U748 ( .A1(n753), .A2(n462), .ZN(n745) );
XNOR2_X1 U749 ( .A(n719), .B(n720), .ZN(n753) );
XNOR2_X1 U750 ( .A(m1[4]), .B(n463), .ZN(n720) );
OAI22_X1 U751 ( .A1(n754), .A2(n755), .B1(n756), .B2(n757), .ZN(n719) );
AND2_X1 U752 ( .A1(n754), .A2(n755), .ZN(n756) );
NAND2_X1 U753 ( .A1(n758), .A2(n456), .ZN(n744) );
XNOR2_X1 U754 ( .A(p0[3]), .B(n457), .ZN(n456) );
XNOR2_X1 U755 ( .A(n728), .B(n723), .ZN(n758) );
XOR2_X1 U756 ( .A(m1[3]), .B(n457), .Z(n723) );
AOI21_X1 U757 ( .B1(n742), .B2(n740), .A(n759), .ZN(n728) );
INV_X1 U758 ( .A(n760), .ZN(n759) );
OAI21_X1 U759 ( .B1(n742), .B2(n740), .A(n743), .ZN(n760) );
AOI22_X1 U760 ( .A1(n737), .A2(n735), .B1(n761), .B2(n762), .ZN(n743) );
OR2_X1 U761 ( .A1(n737), .A2(n735), .ZN(n762) );
INV_X1 U762 ( .A(n738), .ZN(n761) );
AOI22_X1 U763 ( .A1(n458), .A2(n454), .B1(n763), .B2(p0[4]), .ZN(n738) );
OR2_X1 U764 ( .A1(n454), .A2(n458), .ZN(n763) );
XNOR2_X1 U765 ( .A(n764), .B(n765), .ZN(n454) );
NAND2_X1 U766 ( .A1(n766), .A2(n462), .ZN(n764) );
XNOR2_X1 U767 ( .A(n767), .B(n768), .ZN(n766) );
XNOR2_X1 U768 ( .A(m1[0]), .B(n457), .ZN(n458) );
XOR2_X1 U769 ( .A(n769), .B(n770), .Z(n735) );
NAND2_X1 U770 ( .A1(n771), .A2(n462), .ZN(n769) );
```

```
XNOR2_X1 U771 ( .A(n772), .B(n773), .ZN(n771) );
XNOR2_X1 U772 ( .A(m1[1]), .B(n457), .ZN(n737) );
INV_X1 U773 ( .A(p0[4]), .ZN(n457) );
XOR2_X1 U774 ( .A(n774), .B(n755), .Z(n740) );
XOR2_X1 U775 ( .A(n775), .B(n750), .Z(n755) );
OAI211_X1 U776 ( .C1(n776), .C2(n777), .A(n714), .B(p0[1]), .ZN(n750) );
NAND3_X1 U777 ( .A1(n579), .A2(n777), .A3(n778), .ZN(n714) );
INV_X1 U778 ( .A(m1[5]), .ZN(n777) );
AND2_X1 U779 ( .A1(n579), .A2(n778), .ZN(n776) );
NAND2_X1 U780 ( .A1(n779), .A2(n711), .ZN(n775) );
XOR2_X1 U781 ( .A(n749), .B(n752), .Z(n779) );
XOR2_X1 U782 ( .A(n579), .B(p0[2]), .Z(n752) );
INV_X1 U783 ( .A(m1[4]), .ZN(n579) );
OAI21_X1 U784 ( .B1(n780), .B2(n781), .A(n782), .ZN(n749) );
INV_X1 U785 ( .A(n783), .ZN(n782) );
AOI21_X1 U786 ( .B1(n781), .B2(n780), .A(n784), .ZN(n783) );
INV_X1 U787 ( .A(n785), .ZN(n781) );
NAND2_X1 U788 ( .A1(n786), .A2(n462), .ZN(n774) );
XNOR2_X1 U789 ( .A(n472), .B(p0[3]), .ZN(n462) );
XOR2_X1 U790 ( .A(n757), .B(n754), .Z(n786) );
XNOR2_X1 U791 ( .A(m1[3]), .B(n463), .ZN(n754) );
OAI22_X1 U792 ( .A1(n787), .A2(n770), .B1(n788), .B2(n772), .ZN(n757) );
OAI22_X1 U793 ( .A1(n767), .A2(n765), .B1(n789), .B2(n790), .ZN(n772) );
INV_X1 U794 ( .A(n768), .ZN(n790) );
AOI22_X1 U795 ( .A1(n464), .A2(n460), .B1(n791), .B2(p0[3]), .ZN(n768) );
OR2_X1 U796 ( .A1(n460), .A2(n464), .ZN(n791) );
XOR2_X1 U797 ( .A(n792), .B(n793), .Z(n460) );
NAND2_X1 U798 ( .A1(n794), .A2(n711), .ZN(n792) );
XOR2_X1 U799 ( .A(n795), .B(n796), .Z(n794) );
XNOR2_X1 U800 ( .A(m1[0]), .B(n463), .ZN(n464) );
AND2_X1 U801 ( .A1(n767), .A2(n765), .ZN(n789) );
XNOR2_X1 U802 ( .A(n797), .B(n798), .ZN(n765) );
NAND2_X1 U803 ( .A1(n799), .A2(n711), .ZN(n797) );
XOR2_X1 U804 ( .A(n800), .B(n801), .Z(n799) );
XNOR2_X1 U805 ( .A(m1[1]), .B(n463), .ZN(n767) );
NOR2_X1 U806 ( .A1(n773), .A2(n802), .ZN(n788) );
INV_X1 U807 ( .A(n802), .ZN(n770) );
XNOR2_X1 U808 ( .A(n803), .B(n780), .ZN(n802) );
AND2_X1 U809 ( .A1(n804), .A2(p0[1]), .ZN(n780) );
XNOR2_X1 U810 ( .A(m1[4]), .B(n778), .ZN(n804) );
NAND2_X1 U811 ( .A1(n805), .A2(n711), .ZN(n803) );
XNOR2_X1 U812 ( .A(p0[1]), .B(n472), .ZN(n711) );
```

```
 XNOR2_X1 U813 ( .A(n785), .B(n784), .ZN(n805) );
 XNOR2_X1 U814 ( .A(m1[3]), .B(n472), .ZN(n784) );
 AOI21_X1 U815 ( .B1(n800), .B2(n798), .A(n806), .ZN(n785) );
 INV_X1 U816 ( .A(n807), .ZN(n806) );
 OAI21_X1 U817 ( .B1(n798), .B2(n800), .A(n801), .ZN(n807) );
 XNOR2_X1 U818 ( .A(n808), .B(p0[2]), .ZN(n801) );
 AOI211_X1 U819 ( .C1(n809), .C2(m1[3]), .A(n778), .B(n810), .ZN(n798) );
 NOR2_X1 U820 ( .A1(n809), .A2(m1[3]), .ZN(n778) );
 OAI21_X1 U821 ( .B1(n811), .B2(n793), .A(n812), .ZN(n800) );
 OAI21_X1 U822 ( .B1(n795), .B2(n813), .A(n796), .ZN(n812) );
 XNOR2_X1 U823 ( .A(m1[1]), .B(n472), .ZN(n796) );
 INV_X1 U824 ( .A(n793), .ZN(n813) );
 OAI211_X1 U825 ( .C1(n814), .C2(n808), .A(n809), .B(p0[1]), .ZN(n793) );
 NAND2_X1 U826 ( .A1(n814), .A2(n808), .ZN(n809) );
 NOR2_X1 U827 ( .A1(m1[1]), .A2(m1[0]), .ZN(n814) );
 INV_X1 U828 ( .A(n795), .ZN(n811) );
 AOI21_X1 U829 ( .B1(n472), .B2(n470), .A(n468), .ZN(n795) );
 XOR2_X1 U830 ( .A(m1[0]), .B(n472), .Z(n468) );
 MUX2_X1 U831 ( .A(n815), .B(n816), .S(n651), .Z(n470) );
 INV_X1 U832 ( .A(m1[1]), .ZN(n651) );
 OR2_X1 U833 ( .A1(n810), .A2(m1[0]), .ZN(n815) );
 INV_X1 U834 ( .A(p0[1]), .ZN(n810) );
 INV_X1 U835 ( .A(p0[2]), .ZN(n472) );
 INV_X1 U836 ( .A(n773), .ZN(n787) );
 XNOR2_X1 U837 ( .A(m1[2]), .B(n463), .ZN(n773) );
 INV_X1 U838 ( .A(p0[3]), .ZN(n463) );
 XOR2_X1 U839 ( .A(n808), .B(p0[4]), .Z(n742) );
 INV_X1 U840 ( .A(m1[2]), .ZN(n808) );
 XOR2_X1 U841 ( .A(m1[2]), .B(p0[5]), .Z(n700) );
 XOR2_X1 U842 ( .A(m1[2]), .B(p0[6]), .Z(n650) );
 XNOR2_X1 U843 ( .A(m1[2]), .B(p0[7]), .ZN(n636) );
 XOR2_X1 U844 ( .A(m1[2]), .B(n478), .Z(n425) );
 INV_X1 U845 ( .A(pout), .ZN(n478) );
 INV_X1 U846 ( .A(n816), .ZN(result_pro[0]) );
 NAND2_X1 U847 ( .A1(p0[1]), .A2(m1[0]), .ZN(n816) );
endmodule
```

D. Test Pattern

```
STIL 1.0 { Design 2005; }
Header {
  Title "  TetraMAX(R)  L-2016.03-SP5-i161014_180153 STIL output";
  Date "Thu May  2 17:22:26 2024";
```

```
  History {
    Ann {*   Uncollapsed Transition Fault Summary Report *}
    Ann {* ----------------------------------------------- *}
    Ann {* fault class                  code   #faults *}
    Ann {* ------------------------------  ----  --------- *}
    Ann {* Detected                      DT      2576 *}
    Ann {* Possibly detected             PT         0 *}
    Ann {* Undetectable                  UD         0 *}
    Ann {* ATPG untestable               AU         0 *}
    Ann {* Not detected                  ND        52 *}
    Ann {* ----------------------------------------------- *}
    Ann {* total faults                         2628 *}
    Ann {* test coverage                        98.02% *}
    Ann {* ----------------------------------------------- *}
    Ann {*  *}
    Ann {*          Pattern Summary Report *}
    Ann {* ----------------------------------------------- *}
    Ann {* #internal patterns                      61 *}
    Ann {*     #fast_sequential patterns            61 *}
    Ann {* ----------------------------------------------- *}
    Ann {*  *}
    Ann {* rule  severity  #fails  description *}
    Ann {* ----  --------  ------  ---------------------------------- *}
    Ann {* N5    warning     163  redefined module *}
    Ann {*  *}
    Ann {* There are no clocks *}
    Ann {* There are no constraint ports *}
    Ann {* There are no equivalent pins *}
    Ann {* There are no net connections *}
    Ann {* top_module_name = booth *}
    Ann {* Unified STIL Flow *}
    Ann {* min_n_shifts = 1 *}
    Ann {* serial_flag = 1 *}
  }
}
Signals {
  "m1[7]" In; "m1[6]" In; "m1[5]" In; "m1[4]" In;
  "m1[3]" In; "m1[2]" In; "m1[1]" In; "m1[0]" In;
  "m2[7]" In; "m2[6]" In; "m2[5]" In; "m2[4]" In;
  "m2[3]" In; "m2[2]" In; "m2[1]" In; "m2[0]" In;
  "result_pro[15]" Out; "result_pro[14]" Out; "result_pro[13]" Out; "result_pro[12]" Out;
"result_pro[11]" Out;
```

```
  "result_pro[10]" Out; "result_pro[9]" Out; "result_pro[8]" Out; "result_pro[7]" Out; "result_pro[6]"
Out;
  "result_pro[5]" Out; "result_pro[4]" Out; "result_pro[3]" Out; "result_pro[2]" Out; "result_pro[1]" Out;
  "result_pro[0]" Out;
}
SignalGroups {
  "_default_In_Timing_" = '"m1[7]" + "m1[6]" + "m1[5]" +
  "m1[4]" + "m1[3]" + "m1[2]" + "m1[1]" +
  "m1[0]" + "m2[7]" + "m2[6]" + "m2[5]" +
  "m2[4]" + "m2[3]" + "m2[2]" + "m2[1]" +
  "m2[0]"'; // #signals=16
  "_pi" = '"m1[7]" + "m1[6]" + "m1[5]" +
  "m1[4]" + "m1[3]" + "m1[2]" + "m1[1]" +
  "m1[0]" + "m2[7]" + "m2[6]" + "m2[5]" +
  "m2[4]" + "m2[3]" + "m2[2]" + "m2[1]" +
  "m2[0]"'; // #signals=16
  "_in" = '"m1[7]" + "m1[6]" + "m1[5]" +
  "m1[4]" + "m1[3]" + "m1[2]" + "m1[1]" +
  "m1[0]" + "m2[7]" + "m2[6]" + "m2[5]" +
  "m2[4]" + "m2[3]" + "m2[2]" + "m2[1]" +
  "m2[0]"'; // #signals=16
  "_default_Out_Timing_" = '"result_pro[15]" + "result_pro[14]" + "result_pro[13]" +
  "result_pro[12]" + "result_pro[11]" + "result_pro[10]" + "result_pro[9]" + "result_pro[8]" +
  "result_pro[7]" + "result_pro[6]" + "result_pro[5]" + "result_pro[4]" + "result_pro[3]" +
  "result_pro[2]" + "result_pro[1]" + "result_pro[0]"'; // #signals=16
  "_po" = '"result_pro[15]" + "result_pro[14]" + "result_pro[13]" + "result_pro[12]" +
  "result_pro[11]" + "result_pro[10]" + "result_pro[9]" + "result_pro[8]" + "result_pro[7]" +
  "result_pro[6]" + "result_pro[5]" + "result_pro[4]" + "result_pro[3]" + "result_pro[2]" +
  "result_pro[1]" + "result_pro[0]"'; // #signals=16
  "_out" = '"result_pro[15]" + "result_pro[14]" + "result_pro[13]" + "result_pro[12]" +
  "result_pro[11]" + "result_pro[10]" + "result_pro[9]" + "result_pro[8]" + "result_pro[7]" +
  "result_pro[6]" + "result_pro[5]" + "result_pro[4]" + "result_pro[3]" + "result_pro[2]" +
  "result_pro[1]" + "result_pro[0]"'; // #signals=16
}
Timing {
  WaveformTable "_default_WFT_" {
    Period '100ns';
    Waveforms {
      "_default_In_Timing_" { 0 { '0ns' D; } }
      "_default_In_Timing_" { 1 { '0ns' U; } }
      "_default_In_Timing_" { Z { '0ns' Z; } }
      "_default_In_Timing_" { N { '0ns' N; } }
```

```
      "_default_Out_Timing_" { X { '0ns' X; } }
      "_default_Out_Timing_" { H { '0ns' X; '40ns' H; } }
      "_default_Out_Timing_" { T { '0ns' X; '40ns' T; } }
      "_default_Out_Timing_" { L { '0ns' X; '40ns' L; } }
    }
  }
}
ScanStructures {
  // Uncomment and modify the following to suit your design
  // ScanChain "chain_name" { ScanIn "chain_input_name"; ScanOut "chain_output_name"; }
}
PatternBurst "_burst_" {
  PatList { "_pattern_" {
  }
}}
PatternExec {
  PatternBurst "_burst_";
}
Procedures {
  "capture" {
    W "_default_WFT_";
    C { "_po"=\r16 X ; }
    "forcePI": V { "_pi"=\r16 # ; }
    "measurePO": V { "_po"=\r16 # ; }
  }
  // Uncomment and modify the following to suit your design
  // load_unload {
    // V { } // force clocks off and scan enable pins active
    // Shift { V { _si=#; _so=#; }} // pulse shift clocks
  // }
}
MacroDefs {
}
Pattern "_pattern_" {
  W "_default_WFT_";
  "precondition all Signals": C { "_pi"=\r16 0 ; "_po"=\r16 X ; }
  Ann {* fast_sequential *}
  "pattern 0": V { "_pi"=1001010000000011; }
  Call "capture" {
    "_pi"=0011001000101110; "_po"=LLLLHLLLHHHHHHLL; }
  Ann {* fast_sequential *}
  "pattern 1": V { "_pi"=1110100111001001; }
```

```
Call "capture" {
  "_pi"=0100110110101100; "_po"=HHHLLHHLHLHHHHLL; }
Ann {* fast_sequential *}
"pattern 2": V { "_pi"=0010100000111000; }
Call "capture" {
  "_pi"=1011001000001100; "_po"=HHHHHHLLLHLHHLLL; }
Ann {* fast_sequential *}
"pattern 3": V { "_pi"=0011111110100000; }
Call "capture" {
  "_pi"=0110101100000001; "_po"=LLLLLLLLHHLHLHH; }
Ann {* fast_sequential *}
"pattern 4": V { "_pi"=0110101100000001; }
Call "capture" {
  "_pi"=0100110110101100; "_po"=HHHLLHHLHLHHHHLL; }
Ann {* fast_sequential *}
"pattern 5": V { "_pi"=0101010111000001; }
Call "capture" {
  "_pi"=1011010100000001; "_po"=HHHHHHHHHLHHLHLH; }
Ann {* fast_sequential *}
"pattern 6": V { "_pi"=1101101101001110; }
Call "capture" {
  "_pi"=0101010110000001; "_po"=HHLHLHLHHHLHLHLH; }
Ann {* fast_sequential *}
"pattern 7": V { "_pi"=1010100111111001; }
Call "capture" {
  "_pi"=1001010100000001; "_po"=HHHHHHHHHLLHLHLH; }
Ann {* fast_sequential *}
"pattern 8": V { "_pi"=1101001000110101; }
Call "capture" {
  "_pi"=0010010101001001; "_po"=LLLLHLHLHLLLHHLH; }
Ann {* fast_sequential *}
"pattern 9": V { "_pi"=0010110000001011; }
Call "capture" {
  "_pi"=0010010110000010; "_po"=HHHLHHLHHHLLHLHL; }
Ann {* fast_sequential *}
"pattern 10": V { "_pi"=1101110101001110; }
Call "capture" {
  "_pi"=1110101000000001; "_po"=HHHHHHHHHHHLHLHL; }
Ann {* fast_sequential *}
"pattern 11": V { "_pi"=1111011100100011; }
Call "capture" {
  "_pi"=0101000101010111; "_po"=LLLHHLHHHLLLLHHH; }
```

```
Ann {* fast_sequential *}
"pattern 12": V { "_pi"=1100010101000001; }
Call "capture" {
  "_pi"=0101110111101011; "_po"=HHHHHLLLLHLHHHHH; }
Ann {* fast_sequential *}
"pattern 13": V { "_pi"=1100001000011011; }
Call "capture" {
  "_pi"=1111101100010110; "_po"=HHHHHHHHHLLHLLHL; }
Ann {* fast_sequential *}
"pattern 14": V { "_pi"=1110101011001010; }
Call "capture" {
  "_pi"=0110000011110111; "_po"=HHHHHHLLHLHLLLLL; }
Ann {* fast_sequential *}
"pattern 15": V { "_pi"=0110000010110011; }
Call "capture" {
  "_pi"=0010101101000000; "_po"=LLLLHLHLHHLLLLLL; }
Ann {* fast_sequential *}
"pattern 16": V { "_pi"=1001001001000110; }
Call "capture" {
  "_pi"=0100101101011111; "_po"=LLLHHLHHHHLHLHLH; }
Ann {* fast_sequential *}
"pattern 17": V { "_pi"=0001010110110010; }
Call "capture" {
  "_pi"=0010101001100000; "_po"=LLLLHHHHHHLLLLLL; }
Ann {* fast_sequential *}
"pattern 18": V { "_pi"=1000101101101111; }
Call "capture" {
  "_pi"=1101010111011110; "_po"=LLLLLHLHHLHHLHHL; }
Ann {* fast_sequential *}
"pattern 19": V { "_pi"=0000101110011011; }
Call "capture" {
  "_pi"=1101010111011110; "_po"=LLLLLHLHHLHHLHHL; }
Ann {* fast_sequential *}
"pattern 20": V { "_pi"=0110101101100111; }
Call "capture" {
  "_pi"=0101010111010000; "_po"=HHHHLLLLLLLHLLLL; }
Ann {* fast_sequential *}
"pattern 21": V { "_pi"=0101101001100000; }
Call "capture" {
  "_pi"=1010101101110011; "_po"=HHLHHLLHHHLHLLLH; }
Ann {* fast_sequential *}
"pattern 22": V { "_pi"=0100101101001111; }
```

```
Call "capture" {
  "_pi"=1010101101000110; "_po"=HHHLHLLLHHLLLLHL; }
Ann {* fast_sequential *}
"pattern 23": V { "_pi"=1100011101110000; }
Call "capture" {
  "_pi"=0110010101110010; "_po"=LLHLHHLLHHHHHLHL; }
Ann {* fast_sequential *}
"pattern 24": V { "_pi"=0110010001011111; }
Call "capture" {
  "_pi"=0110001110011010; "_po"=HHLHHLLLHLLLHHHL; }
Ann {* fast_sequential *}
"pattern 25": V { "_pi"=1011101100001110; }
Call "capture" {
  "_pi"=0110101101111101; "_po"=LLHHLHLLLLHHHHHH; }
Ann {* fast_sequential *}
"pattern 26": V { "_pi"=1110101100000111; }
Call "capture" {
  "_pi"=1001101100001010; "_po"=HHHHHHLLLLLLHHHL; }
Ann {* fast_sequential *}
"pattern 27": V { "_pi"=1101101100001110; }
Call "capture" {
  "_pi"=0110101100011001; "_po"=LLLLHLHLLHHHLLHH; }
Ann {* fast_sequential *}
"pattern 28": V { "_pi"=1000101100000111; }
Call "capture" {
  "_pi"=1001101100001110; "_po"=HHHHHLHLLHHHHLHL; }
Ann {* fast_sequential *}
"pattern 29": V { "_pi"=0110110101110000; }
Call "capture" {
  "_pi"=1001101100010010; "_po"=HHHHHLLLHHHLLHHL; }
Ann {* fast_sequential *}
"pattern 30": V { "_pi"=0010010111000000; }
Call "capture" {
  "_pi"=1101111110010001; "_po"=LLLLHHHLLHLLHHHH; }
Ann {* fast_sequential *}
"pattern 31": V { "_pi"=1010110110000000; }
Call "capture" {
  "_pi"=1101111111110110; "_po"=LLLLLLLHLHLLHLHL; }
Ann {* fast_sequential *}
"pattern 32": V { "_pi"=1000110100010010; }
Call "capture" {
  "_pi"=1101101000011001; "_po"=HHHHHHLLLHLLHLHL; }
```

```
Ann {* fast_sequential *}
"pattern 33": V { "_pi"=0011001000110000; }
Call "capture" {
  "_pi"=0010100100100010; "_po"=LLLLLHLHLHHHLLHL; }
Ann {* fast_sequential *}
"pattern 34": V { "_pi"=1110101010110011; }
Call "capture" {
  "_pi"=0010010111100110; "_po"=HHHHHHLLLLHHHHHL; }
Ann {* fast_sequential *}
"pattern 35": V { "_pi"=0010100001111101; }
Call "capture" {
  "_pi"=1111001001000100; "_po"=HHHHHHLLLHLLHLLL; }
Ann {* fast_sequential *}
"pattern 36": V { "_pi"=1100111111110000; }
Call "capture" {
  "_pi"=1011001011100100; "_po"=LLLLHLLLHLLLHLLL; }
Ann {* fast_sequential *}
"pattern 37": V { "_pi"=0000001011100010; }
Call "capture" {
  "_pi"=0011010111010001; "_po"=HHHHLHHLLHLLLHLH; }
Ann {* fast_sequential *}
"pattern 38": V { "_pi"=0100110001010100; }
Call "capture" {
  "_pi"=0100101011100111; "_po"=HHHHHLLLHHLLLHHL; }
Ann {* fast_sequential *}
"pattern 39": V { "_pi"=0011100100101001; }
Call "capture" {
  "_pi"=1011110010011000; "_po"=LLLHHLHHHLHLLLLL; }
Ann {* fast_sequential *}
"pattern 40": V { "_pi"=1101100010111100; }
Call "capture" {
  "_pi"=0101111010110000; "_po"=HHHLLLHLHLHLLLLL; }
Ann {* fast_sequential *}
"pattern 41": V { "_pi"=1110111110010000; }
Call "capture" {
  "_pi"=0000000111101000; "_po"=HHHHHHHHHHHLHLLL; }
Ann {* fast_sequential *}
"pattern 42": V { "_pi"=0100111111111011; }
Call "capture" {
  "_pi"=0010101001111111; "_po"=LLLHLHLLHHLHLHHL; }
Ann {* fast_sequential *}
"pattern 43": V { "_pi"=1010101110111001; }
```

```
Call "capture" {
  "_pi"=0000011101110010; "_po"=LLLLLLHHLLLHHHHL; }
Ann {* fast_sequential *}
"pattern 44": V { "_pi"=0101111000000100; }
Call "capture" {
  "_pi"=1000101101000000; "_po"=HHHLLLHLHHLLLLLL; }
Ann {* fast_sequential *}
"pattern 45": V { "_pi"=1010101000000111; }
Call "capture" {
  "_pi"=0101010110100101; "_po"=HHHLLLLHHHLLHLLH; }
Ann {* fast_sequential *}
"pattern 46": V { "_pi"=0110000110011100; }
Call "capture" {
  "_pi"=1010101100110000; "_po"=HHHHLLLLLLLHLLLL; }
Ann {* fast_sequential *}
"pattern 47": V { "_pi"=1001010011101000; }
Call "capture" {
  "_pi"=0101010111100000; "_po"=HHHHLHLHLHHLLLLL; }
Ann {* fast_sequential *}
"pattern 48": V { "_pi"=0001101100110001; }
Call "capture" {
  "_pi"=0010101001110000; "_po"=LLLHLLHLLHHLLLLL; }
Ann {* fast_sequential *}
"pattern 49": V { "_pi"=1010010010010011; }
Call "capture" {
  "_pi"=0110101000110000; "_po"=LLLHLLHHHHHLLLLL; }
Ann {* fast_sequential *}
"pattern 50": V { "_pi"=0010101100000001; }
Call "capture" {
  "_pi"=1010101100000101; "_po"=HHHHHHHLLHLHLHHH; }
Ann {* fast_sequential *}
"pattern 51": V { "_pi"=0000101010011011; }
Call "capture" {
  "_pi"=1101010101111101; "_po"=HHHLHLHHLLLLLLLH; }
Ann {* fast_sequential *}
"pattern 52": V { "_pi"=0110101100000001; }
Call "capture" {
  "_pi"=0110011101101001; "_po"=LLHLHLHLLLHHHHHH; }
Ann {* fast_sequential *}
"pattern 53": V { "_pi"=1110101100010000; }
Call "capture" {
  "_pi"=0101011101011001; "_po"=LLLHHHHLLLHHHHHH; }
```

```
  Ann {* fast_sequential *}
  "pattern 54": V { "_pi"=1001011001011111; }
  Call "capture" {
    "_pi"=1101110100110010; "_po"=HHHHHLLHLLHLHLHL; }
  Ann {* fast_sequential *}
  "pattern 55": V { "_pi"=0011111111000100; }
  Call "capture" {
    "_pi"=0100011011011100; "_po"=HHHHLHHLLLHLHLLL; }
  Ann {* fast_sequential *}
  "pattern 56": V { "_pi"=0100100100000110; }
  Call "capture" {
    "_pi"=0100111110101000; "_po"=HHHLLHLLHHLHHLLL; }
  Ann {* fast_sequential *}
  "pattern 57": V { "_pi"=0111101000011010; }
  Call "capture" {
    "_pi"=1100111000010101; "_po"=HHHHHLHHHHHLLHHL; }
  Ann {* fast_sequential *}
  "pattern 58": V { "_pi"=1100000110110010; }
  Call "capture" {
    "_pi"=0101111001011110; "_po"=LLHLLLHLHLLLLHLL; }
  Ann {* fast_sequential *}
  "pattern 59": V { "_pi"=1101111110011001; }
  Call "capture" {
    "_pi"=1010100000000110; "_po"=HHHHHHLHHHHHLLLL; }
  Ann {* fast_sequential *}
  "pattern 60": V { "_pi"=0101000111110100; }
  Call "capture" {
    "_pi"=1101000000110010; "_po"=HHHHLHHLHLHLLLLL; }
}

// Patterns reference 183 V statements, generating 183 test cycles
```