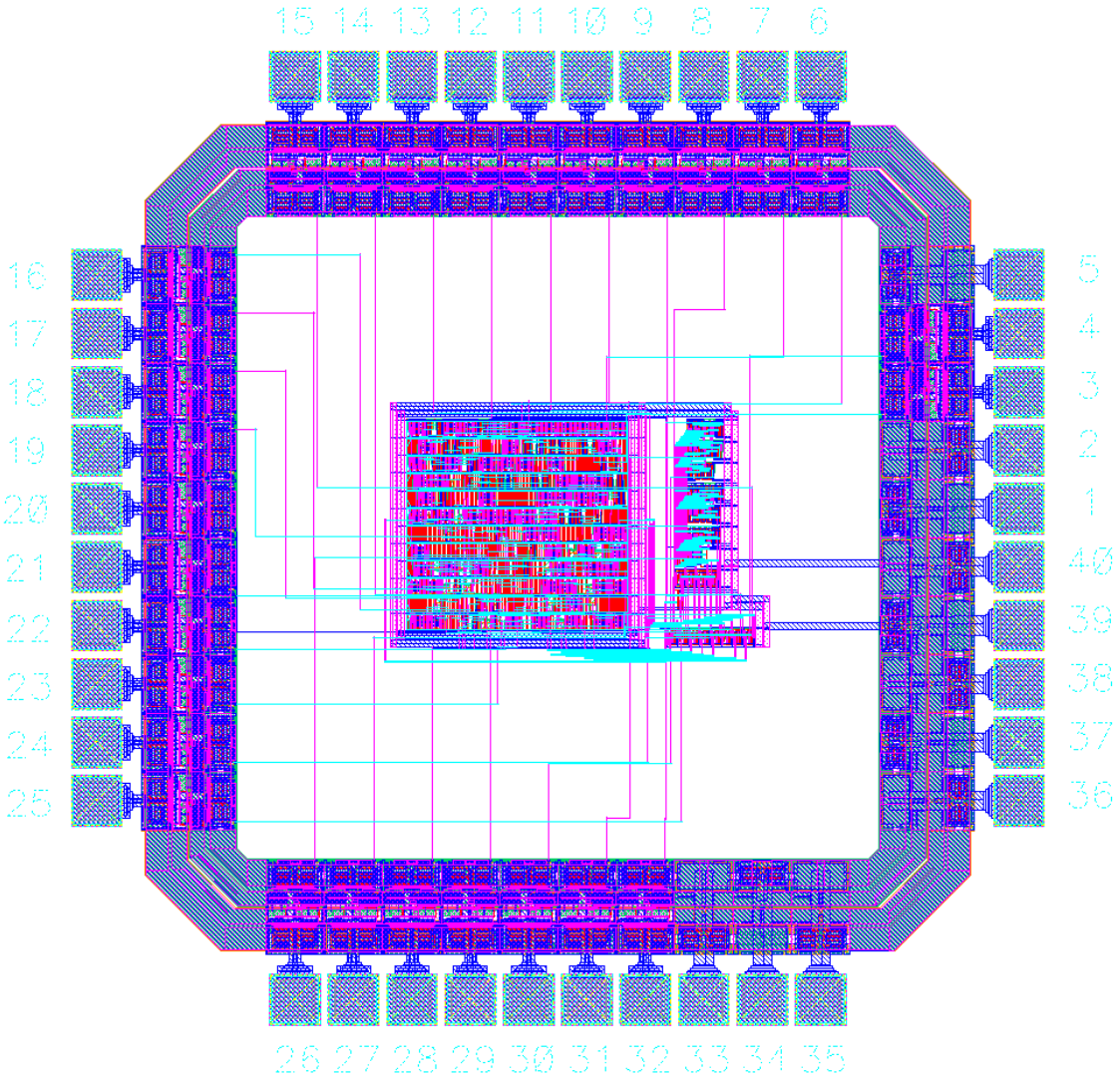
THE E155 ASIC

E158 - Introduction to CMOS VLSI Design

Final Report

Gourav Khadge and Leif Park Jordan

4/21/15



Introduction

The goal of this project was to design an application-specific integrated circuit that would fulfill the requirements of the second and third labs from Harvey Mudd's "Microprocessor-based Systems: Design and Applications" course (E155). These labs require students to design a system to read in numbers from an external keypad and output them to a dual seven-segment display using time-multiplexing. The system must ignore held keys, handle switch bounce, and handle up to two simultaneous key presses.

The ASIC implemented in this project uses a 0.6 μm process and is packaged in a 1.5 mm x 1.5 mm 40-pin MOSIS "Tiny Chip."

Specifications

The inputs and outputs of the ASIC are tabulated in Table 1. Eight pins are used to read from two external switches. Eight more are used to drive the keypad's columns and read from its rows. Nine more are used to drive the seven-segment display and time-multiplexing transistors. Four pins are used as inputs for the two clock phases, synchronous reset, and to select between keypad mode and switch mode. The remaining pins are used for VDD and Ground. The chip's physical pin layout is shown in Figure 1.

Pins	**Pin Type**	**Pin Number**
Reset	Input	3
Mode	Input	4
PH1	Input	6
PH2	Input	7
Switch1[3:0]	Input	[8:11]
Switch2[3:0]	Input	[12:15]
Rows[3:0]	Input	[16:19]
Cols[3:0]	Output	[20:23]
Enable1	Output	24
Enable2	Output	25
Seg[6:0]	Output	[26:32]
VDD	Input/Output	1,5,34,37,39
GND	Input/Output	2, 33,35,36,38,40

Table 1: Pin Specifications

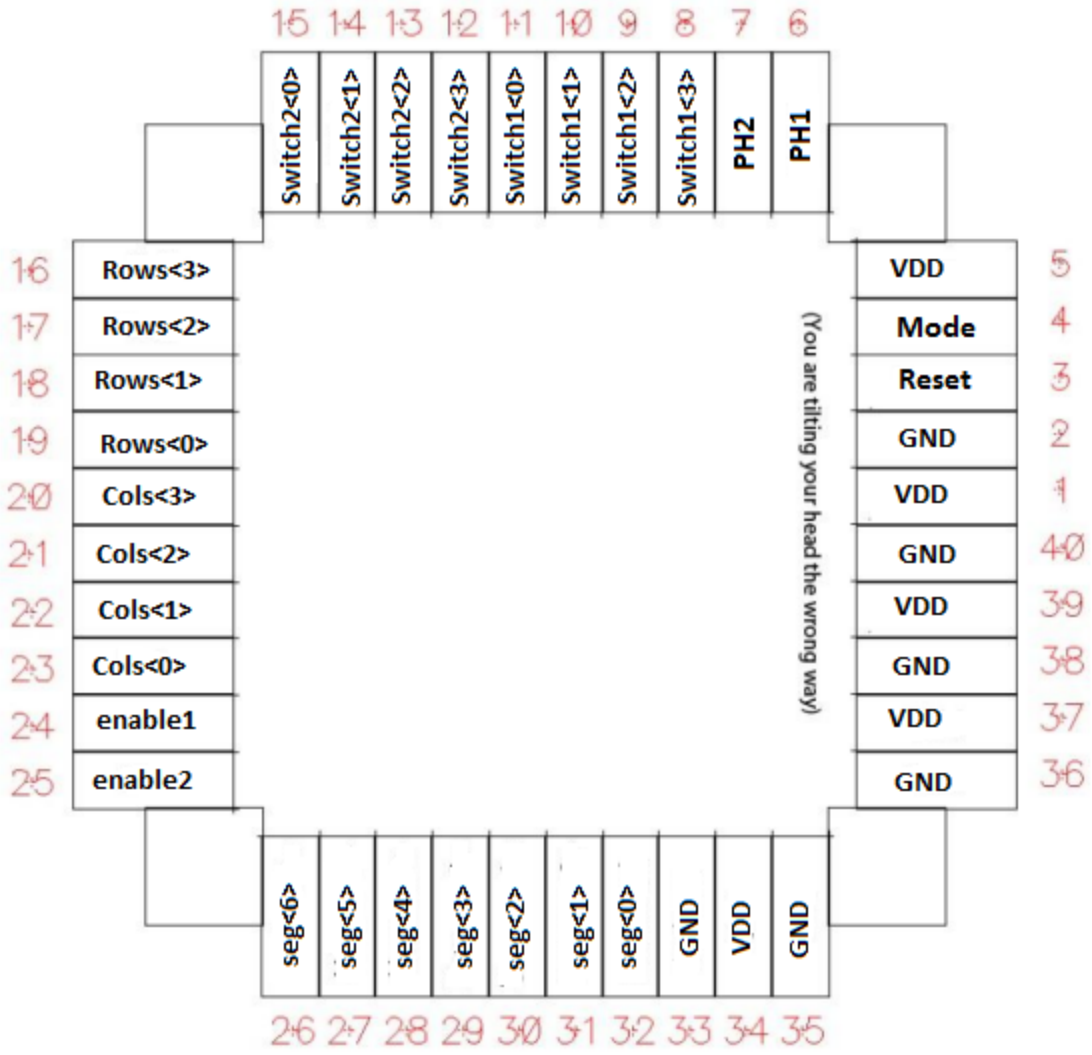


Figure 1: Pinout Diagram

Theory of Operation

The chip is a seven-segment LED controller that can process inputs in two modes. The digits are time multiplexed and output to a two digit seven segment display. The modes are set by the mode input pin (Pin 4) in the following configuration:

1. (Mode pin held low) Take input from two 4 position switches and map input from each switch to a digit on the two digit seven segment display.
2. (Mode pin held high) Take input from a 16 button keypad and output the last two digits entered onto the two digit seven segment display

The seven segments values are output from seg[6:0]. The enable1 and enable2 pins are used to time multiplex the output and are intended to be connected to the base of a transistor between the common anode of a two digit seven segment display and ground. The time multiplexing occurs at roughly 150 Hz with a 40 MHz input clock, avoiding noticeable flicker.

Floorplan

The initial floorplan layout plan is shown in Figure 2. The plan is dominated by two main elements: the keypad controller and the seven segment display.

The estimated sizing for these blocks were calculated by estimating the number of type of each subcircuit element. The keypad controller was synthesized from existing SystemVerilog code in Quartus II 12.0 and the number and types of gates were tabulated. The seven segment display logic block was sketched out. It mainly consists of three 4-bit multiplexers, a seven segment decoder, and a 20 bit counter. The number of logic gates in the seven segment decoder logic was estimated from online datasheets. The 20 bit counter was estimated to be of similar size to a 20 bit ripple carry adder. Basic logic gate sizes were estimated from the provided Cadence libraries.

The final floorplan is shown in Figure 3. It matched our initial plans reasonably well. Our blocks ended up slightly taller than expected. The keypad controller was slightly longer as well, though our seven segment decoder was slightly less wide as expected. All in all, our estimates were roughly within 10% of our initial estimates.

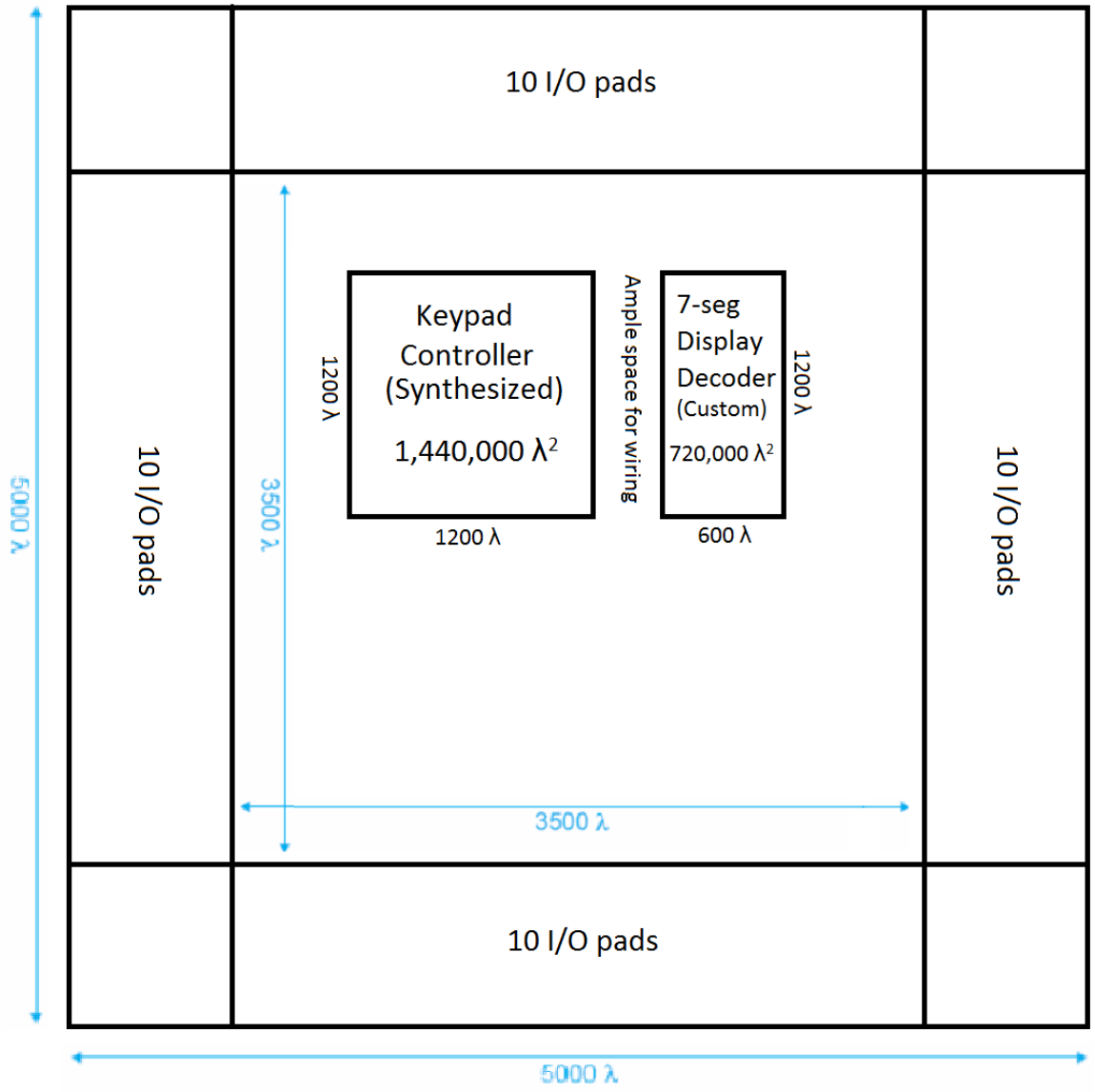


Figure 2: Initial floorplan layout plan

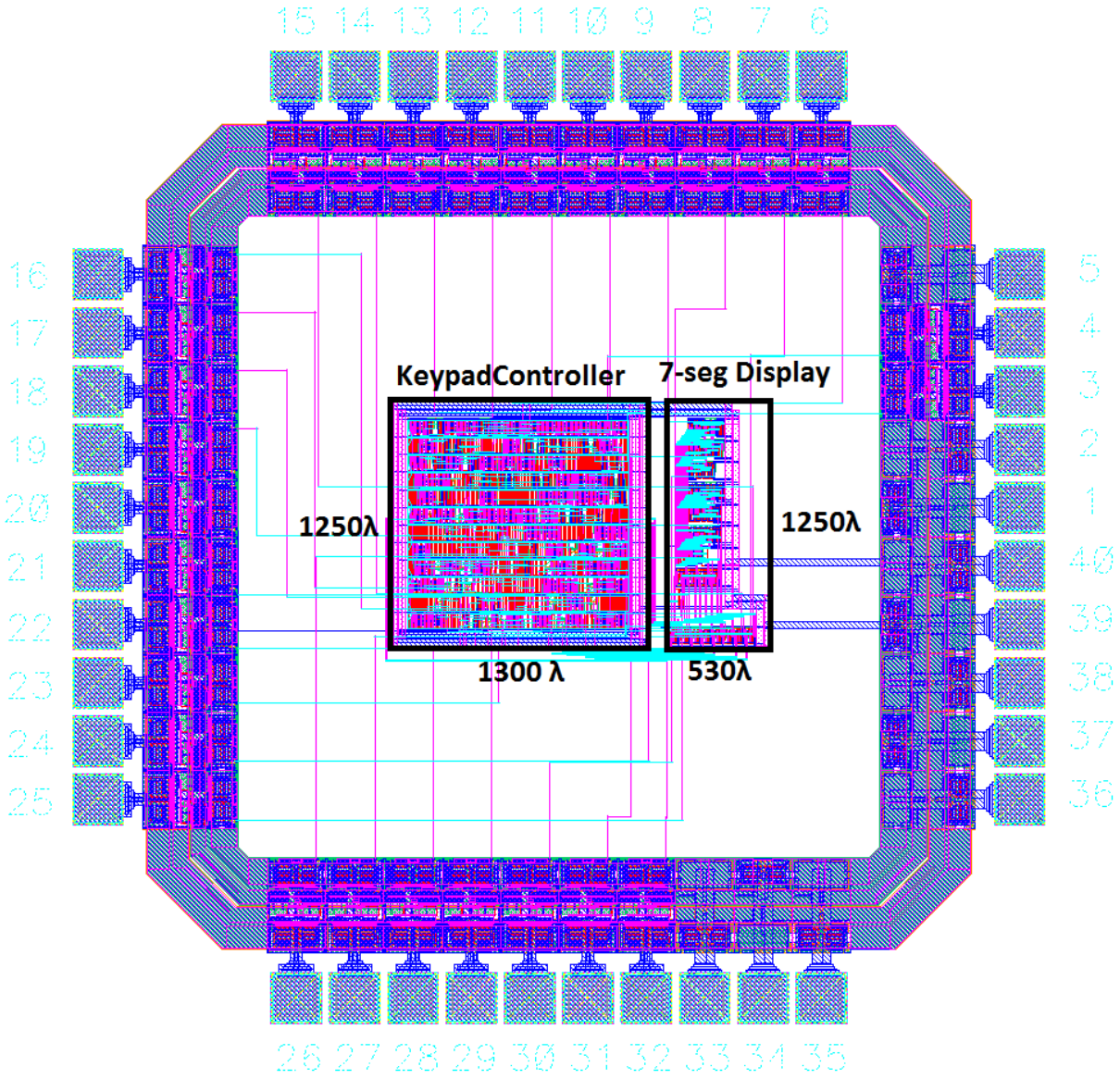


Figure 3: Final Floorplan

Verification

The Verilog code for our chip (Appendix 1) passes our self checking testbench (Appendix 2) using our test vectors (Appendix 3). The chip passes this same testbench on the same set of test vectors. The layout passes DRC, showing that our chip, if manufactured, should not have any physical flaws that would cause issues. The layout also passes LVS showing that our design is consistent with our schematic and should function as the schematic does. There were no analog blocks so no HSPICE simulations were necessary. The project was taped out to a GDS file. The GDS file was

then read back in to Cadence to verify that the tape out process was done correctly. The loaded GDS file passed DRC and LVS.

Postfabrication Test Plan

Were we to fabricate our chip, we would begin testing by checking connectivity between each GND pin and each VDD pin. We would then measure the resistance between VDD and GND to ensure that there were no shorts generated in the manufacturing process. The seven segment display should be wired with seg[0] connecting to A, seg[1] connecting to B, etc. Since the output pins are not wired to source a large amount of current, they need to be connected to transistor circuits to allow seg[6:0] to control transistor current flow to segments A through G.

We would then test functionality, starting with the switch-reading mode. The mode pin would be connected to through a pulldown resistor to ground to enable the switch input mode. We would need to somehow generate a two-phase clock, either using an external clock or using the μMudd32 board's Cyclone III FPGA. We would then connect the switches and keypad to the correct pins, with pulldown resistors to ground on each input pin. The other side of each switch would be connected to VDD. We would then test each possible input from each switch and verify the seven-segment display's output changed as expected. This verifies that both the switch inputs and seven-segment outputs work as anticipated.

Next, the device would be tested in keypad mode. The mode pin should now be connected through a pullup resistor to VDD to enable the keypad input mode. Each key would be pressed in turn and should appear on the seven-segment display. We would watch for switch bounce, which would manifest itself as a single input being recorded twice. With each key verified, we would try holding a key for a long period of time, to ensure the input isn't recorded multiple times. Finally, we would press multiple keys at once, to ensure that the system responds appropriately.

Performing all of these tests by hand would be a quick process once all of the circuitry was wired appropriately.

Future Work

There are a few improvements that could be made to this ASIC in order to improve its user experience. Most notably, there are several analog components that could be added to reduce the number of external components needed. First and foremost, internal pulldown resistors would dramatically reduce the number of external components needed. The pulldown resistor could be made from a long line of

polysilicon attached to an nmos transistor, allowing the user to disable the pulldowns if they don't work as expected.

In addition, the system could be modified to sink more current through the time-multiplexing outputs to the seven-segment display. This would enable the user to connect the time-multiplexing outputs directly to the display, eliminating the need for an external transistor.

Design Time

A summary of the design time for the different aspects of chip development are shown below in Table 2. The plurality of our time was spent in layout, amounting to almost half of our total time spent on the project. Appreciable amounts of time were also spent in Verilog and schematics.

	Design Time (Team Hours)
Project Proposal	**2**
Verilog	**5**
Schematics	**5**
Layout	**12**
Tapeout	**2**
Total	**26**

Table 2: Summary of Design Time

File Locations

Files	**Location on Tera**
Verilog code	/home/lparkjordan/IC_CAD/cadence/VLSIFinalProject/e155ASIC.sv
Test vectors	/home/lparkjordan/IC_CAD/cadence/VLSIFinalProject/e155asic.tv
Synthesis results	/home/lparkjordan/IC_CAD/cadence/VLSIFinalProject/keyscanner_syn.v
All Cadence libraries	/home/lparkjordan/IC_CAD/cadence/VLSIFinalProject/e155ASIC_fixed /home/lparkjordan/IC_CAD/cadence/VLSIFinalProject/muddlib11
GDS	/home/lparkjordan/IC_CAD/cadence/VLSIFinalProject/chip.gds

PDF chip plot	/home/lparkjordan/IC_CAD/cadence/VLSIFinalProject/E155ASIC2.png
PDF of report	/home/lparkjordan/IC_CAD/cadence/VLSIFinalProject/FinalReport.pdf

References

M. Spencer and J. Spjut. “Lab 2: Multiplexed Display.” Laboratory assignment, Harvey Mudd College, Claremont, 2014. Internet: http://pages.hmc.edu/jspjut/class/f2014/e155/lab/lab02.pdf

M. Spencer and J. Spjut. “Lab 3: Keypad Scanner.” Laboratory assignment, Harvey Mudd College, Claremont, 2014. Internet: http://pages.hmc.edu/jspjut/class/f2014/e155/lab/lab03.pdf

Appendices

Appendix 1: Verilog Code

```
module e155ASIC #(parameter TIMEBITS = 3) (input  logic       clk1, clk2, reset,
      // 40 MHz clock
            input logic keypadInputNswitchInput,
                  input logic [3:0] switch1, switch2,        // the four DIP
switches
                  input  logic [3:0]  keypadRows,
                  output logic [3:0]  keypadCols,
                  output logic [6:0]  sevenSeg,               // the segments
of the common-anode 7 segment display
                  output logic      enable1, enable2  // the enables
                  );

logic [3:0]  digit1Keypad, digit2Keypad, digit1, digit2, colEnables;
logic slowtimer;

//choose input of hex digits from either keypad or switches
assign digit1 = (keypadInputNswitchInput)  ? digit1Keypad : switch1;
assign digit2 = (keypadInputNswitchInput)  ? digit2Keypad : switch2;

//use column outputs as enables on output pads
assign keypadCols[0] = colEnables[0] ? colEnables[0] : 1'bZ;
assign keypadCols[1] = colEnables[1] ? colEnables[1] : 1'bZ;
assign keypadCols[2] = colEnables[2] ? colEnables[2] : 1'bZ;
assign keypadCols[3] = colEnables[3] ? colEnables[3] : 1'bZ;
```

```
//get digits from keypad
keyScanner keypadInput(clk1, clk2, reset, keypadRows, colEnables, digit1Keypad,
digit2Keypad, slowtimer);  // output digit1 and digit2 from keypad

//output digits to seven seg
sevenSegOutput out(slowtimer, digit1, digit2, enable1, enable2, sevenSeg);

endmodule

module sevenSegOutput(input logic timer,
                      input logic [3:0] digit1, digit2,
                      output logic  enable1, enable2,
                      output logic [6:0] seg
                      );
 logic [3:0] currentDigit;

assign enable1 = timer;
assign enable2 = ~timer;
assign currentDigit = enable2 ? digit1 : digit2;

always_comb
      case (currentDigit)
      //                gfedcba
      4'b0000: seg = 7'b1000000;
      4'b0001: seg = 7'b1111001;
      4'b0010: seg = 7'b0100100;
      4'b0011: seg = 7'b0110000;
      4'b0100: seg = 7'b0011001;
      4'b0101: seg = 7'b0010010;
      4'b0110: seg = 7'b0000010;
      4'b0111: seg = 7'b1111000;
      4'b1000: seg = 7'b0000000;
      4'b1001: seg = 7'b0011000;
      4'b1010: seg = 7'b0001000;
      4'b1011: seg = 7'b0000011;
      4'b1100: seg = 7'b0100111;
      4'b1101: seg = 7'b0100001;
      4'b1110: seg = 7'b0000110;
      4'b1111: seg = 7'b0001110;
      default: seg = 7'b1111111;
      endcase

endmodule
```

```
module keyScanner #(parameter TIMEBITS = 2) (input  logic     ph1, ph2,
      // timer
            input  logic       reset,
            input  logic [3:0]  rows,
            output logic [3:0]  cols,
            output logic [3:0]  currentDigit, lastDigit,
            output logic slowtimer // 50% duty cycle timer that changes every
2^NBITS-1 cycles
            );                 // the segments


      logic [1:0] state;
      logic [1:0] nextstate;

      logic [3:0] lastrow; // keep track of last button pressed (store row data
corresponding to last key)
      logic [3:0] nextlastrow;
    logic updatedigit; //flag to update digit displays
      logic [3:0] nextdigit;

      logic [3:0] currentSelector; //the digit being used for output

      logic [TIMEBITS-1:0] enabletimer;
      logic slowen; // enable that goes high every 2^NBITS-1 cycles

      flopenr #(TIMEBITS)
enablereg(ph1,ph2,reset,1'b1,enabletimer+1'b1,enabletimer);
      assign slowen = &enabletimer;
      flopenr #(1) timerreg(ph1,ph2,reset,slowen,~slowtimer, slowtimer);

      flopenr #(2) statereg(ph1, ph2, reset, slowen, nextstate, state);
      flopenr #(4) lastrowreg(ph1, ph2, reset, slowen, nextlastrow, lastrow);
      flopenr #(4) lastdigitreg(ph1, ph2, reset, slowen, (updatedigit) ?
currentDigit : lastDigit, lastDigit);
      flopenr #(4) currentdigitreg(ph1, ph2, reset, slowen, (updatedigit) ?
nextdigit : currentDigit, currentDigit);

      // column scan logic
      // Column outputs will be used both as data and output enables to prevent
shorts
      always_comb begin
      case (state)
            2'd0: cols = 4'b1000;
            2'd1: cols = 4'b0100;
            2'd2: cols = 4'b0010;
            2'd3: cols = 4'b0001;
            default: cols = 4'b0000;
      endcase
```

```
end

// next state logic
always_comb begin
// If same button is still down, freeze the system
if (|(lastrow & rows)) begin
      nextdigit = currentDigit; // Can I commment this out?
      updatedigit = 0;
      nextstate = state; // don't change column under consideration
      nextlastrow = lastrow;
end
else begin
      //If something new is pressed!
      if (|rows) begin
      nextstate = state; // don't change column under consideration
      casez (rows)
                  4'b1???: nextlastrow = 4'b1000;
                  4'b01??: nextlastrow = 4'b0100;
                  4'b001?: nextlastrow = 4'b0010;
                  4'b0001: nextlastrow = 4'b0001;
                  default: nextlastrow = 4'b0000; //shouldn't happen
      endcase
      // We have a new digit!
      updatedigit = 1;
      case (state)
            2'd0: casez (rows)
                  4'b1???: nextdigit = 4'h1;//d1;
                  4'b01??: nextdigit = 4'h4;//d4;
                  4'b001?: nextdigit = 4'h7;//d7;
                  4'b0001: nextdigit = 4'he;//de;
                  default: nextdigit = currentDigit; //shouldn't happen
                  endcase
            2'd1: casez (rows)
                  4'b1???: nextdigit = 4'h2;//d2;
                  4'b01??: nextdigit = 4'h5;//d5;
                  4'b001?: nextdigit = 4'h8;//d8;
                  4'b0001: nextdigit = 4'h0;//d0;
                  default: nextdigit = currentDigit; //shouldn't happen
                  endcase
            2'd2: casez (rows)
                  4'b1???: nextdigit = 4'h3;//d3;
                  4'b01??: nextdigit = 4'h6;//d6;
                  4'b001?: nextdigit = 4'h9;//d9;
                  4'b0001: nextdigit = 4'hf;//df;
                  default: nextdigit = currentDigit; //shouldn't happen
                    endcase
            2'd3: casez (rows)
                  4'b1???: nextdigit = 4'ha;//da;
```

```
                        4'b01??: nextdigit = 4'hb;//db;
                        4'b001?: nextdigit = 4'hc;//dc;
                        4'b0001: nextdigit = 4'hd;//dd;
                        default: nextdigit = currentDigit; //shouldn't happen
                        endcase
            endcase
            end

            // If nothing has been pressed here, check next column
            else begin
            updatedigit = 0;
            nextdigit = currentDigit;
            nextlastrow = 4'b0000; // reset knowledge of last button pushed

            case (state)
                  2'd0: nextstate = 2'd1;
                  2'd1: nextstate = 2'd2;
                  2'd2: nextstate = 2'd3;
                  2'd3: nextstate = 2'd0;
                  default: nextstate = 2'd0;
            endcase
            end
      end
      end

endmodule

module flopenr #(parameter WIDTH = 8)
            (input  logic           ph1, ph2, reset, en,
                  input  logic [WIDTH-1:0] d,
                  output logic [WIDTH-1:0] q);

  logic [WIDTH-1:0] d2, resetval;

  assign resetval = 0;

  mux3 #(WIDTH) enrmux(q, d, resetval, {reset, en}, d2);
  flop #(WIDTH) f(ph1, ph2, d2, q);
endmodule

module flop #(parameter WIDTH = 8)
            (input  logic           ph1, ph2,
            input  logic [WIDTH-1:0] d,
            output logic [WIDTH-1:0] q);

  logic [WIDTH-1:0] mid;

  latch1 #(WIDTH) master(ph2, d, mid);
```

```
  latch1 #(WIDTH) slave(ph1, mid, q);
endmodule

module latch1 #(parameter WIDTH = 8)
             (input  logic             ph,
             input  logic [WIDTH-1:0] d,
             output logic [WIDTH-1:0] q);

  always_latch
      if (ph) q <= d;
endmodule

module mux3 #(parameter WIDTH = 8)
             (input  logic [WIDTH-1:0] d0, d1, d2,
             input  logic [1:0]       s,
             output logic [WIDTH-1:0] y);

  always_comb
      casez (s)
      2'b00: y = d0;
      2'b01: y = d1;
      2'b1?: y = d2;
      endcase
endmodule
```

Appendix 2: Self-Checking Verilog Testbench for Chip

```
module testbench();
            logic ph1, ph2, reset;
            logic mode; //inputs
            logic [3:0] switch1, switch2, rows;
            logic [3:0] expectedColumns; // expected outputs
            logic [6:0] expectedSeg;
            logic [1:0] expectedMulti;
            logic [3:0] columns; // actual outputs
            logic [6:0] seg;
            logic multi1, multi2;
            logic [26:0] testvectors[10000:0];
            logic [31:0] vectornum, errors;

            //instantiate device under test
            e155ASIC dut(ph1, ph2, reset, mode, switch1, switch2, rows,
columns, seg, multi1, multi2);

            //generate clock(s)
            always
      begin
      ph1 = 0; ph2 = 0; #2;
      ph1 = 1; # 8;
      ph1 = 0; #2;
      ph2 = 1; # 8;
      end

            //at start of test, load vectors and pulse reset
            initial
                        begin
                                    $readmemb ("e155asic.tv", testvectors);
                                    vectornum = 0; errors = 0;
                                    reset <= 1; #41; reset <= 0;
                        end

            // apply test vectors on rising edge of clock phase 1
            always @ (posedge ph1)
                        begin
                                    #1;
                                    {mode, switch1, switch2, rows,
expectedColumns, expectedSeg, expectedMulti} = testvectors[vectornum];
                        end

            // check results on falling edge of clock phase 2
            always @ (negedge ph2)
            begin
```

```
                        if(~reset) begin //skip during reset
                                    if(columns !== expectedColumns || seg !==
expectedSeg || {multi1, multi2} !== expectedMulti) begin
                                                $display("Error on line %d:
inputs = %h", vectornum+1, {mode, switch1, switch2, rows});
                                                $display(" outputs = %h (%h
expected)", {columns, seg, multi1, multi2}, {expectedColumns, expectedSeg,
expectedMulti});
                                                errors = errors + 1;
                                    end
                                    vectornum = vectornum + 1;
                        end
                        if(testvectors[vectornum] === 27'bx) begin
                                    $display("%d tests completed with %d
errors", vectornum, errors);
                                    $stop;
                        end
            end
endmodule
```

Appendix 3: Test Vectors

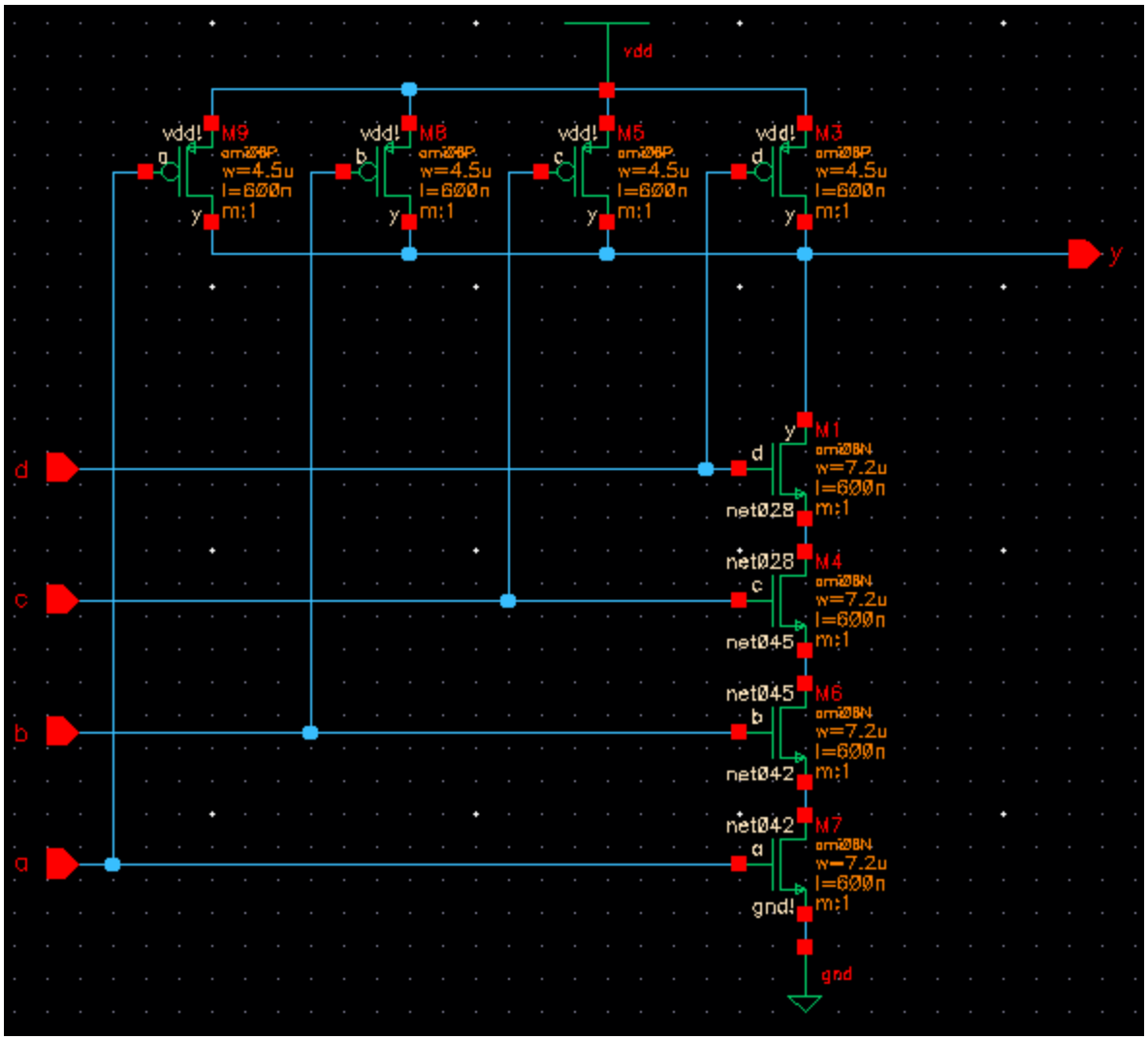
```
00000000000001ZZZ100000001
00000000000001ZZZ100000001
00000000000001ZZZ100000001
00000000000001ZZZ100000001

00001000000000Z1ZZ100000010
00001000000000Z1ZZ100000010
00001000000000Z1ZZ100000010
00001000000000Z1ZZ100000010

00001000000000ZZ1Z111100101
00001000000000ZZ1Z111100101
00001000000000ZZ1Z111100101
00001000000000ZZ1Z111100101

00001000000000ZZZ1100000010
00001000000000ZZZ1100000010
00001000000000ZZZ1100000010
00001000000000ZZZ1100000010

00001000000001ZZZ111100101
00010000000001ZZZ010010001
00011000000001ZZZ011000001
00100000000001ZZZ001100101

00101000100000Z1ZZ111100110
00101001000000Z1ZZ010010010
00101001100000Z1ZZ011000010
00101010000000Z1ZZ001100110

00101010100000ZZ1Z001001001
00110010100000ZZ1Z000001001
00111010100000ZZ1Z111100001
01000010100000ZZ1Z000000001

01000010100000ZZZ1001001010
01000011000000ZZZ1000001010
01000011100000ZZZ1111100010
01000100000000ZZZ1000000010

01001100000001ZZZ001100001
01010100000001ZZZ000100001
01011100000001ZZZ000001101
01100100000001ZZZ010011101

01100100100000Z1ZZ001100010
```

```
01100101000000Z1ZZ000100010
01100101100000Z1ZZ000001110
01100110000000Z1ZZ010011110

01101110000000ZZ1Z010000101
01110110000000ZZ1Z000011001
01111110000000ZZ1Z000111001
01111110000000ZZ1Z000111001

01111110100000ZZZ1010000110
01111111000000ZZZ1000011010
01111111100000ZZZ1000111010
01111111100000ZZZ1000111010

11111110100001ZZZ100000001
11111111000001ZZZ100000001
11111111100001ZZZ100000001
11111111100001ZZZ100000001

11111110100000Z1ZZ100000010
11111111000000Z1ZZ100000010
11111111100000Z1ZZ100000010
11111111100000Z1ZZ100000010

11111110100000ZZ1Z100000001
11111111000000ZZ1Z100000001
11111111100000ZZ1Z100000001
11111111100000ZZ1Z100000001

11111110100000ZZZ1100000010
11111111000001ZZZ1100000010
11111111100001ZZZ1100000010
11111111100001ZZZ1100000010

11111110100001ZZZ1010000101
11111111000001ZZZ1010000101
11111111100001ZZZ1010000101
11111111100001ZZZ1010000101

11111110100001ZZZ1100000010
11111111000001ZZZ1100000010
11111111100001ZZZ1100000010
11111111100001ZZZ1100000010

11111110100001ZZZ1010000101
11111111000001ZZZ1010000101
11111111100001ZZZ1010000101
11111111100001ZZZ1010000101
```

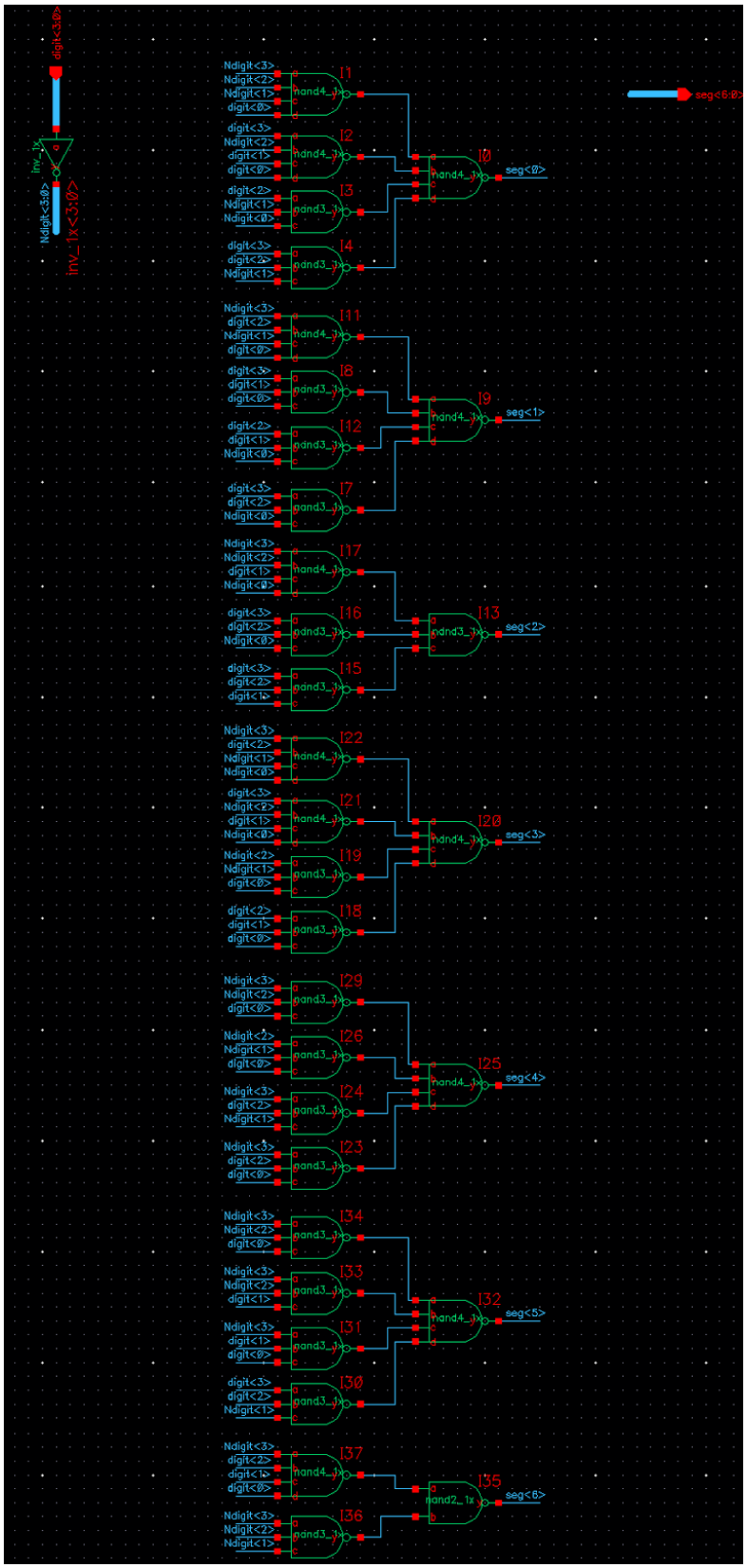
```
11111110100001ZZZ1100000010
11111111000001ZZZ1100000010
11111111100000ZZZ1100000010
11111111100000ZZZ1100000010

11111110100001ZZZ010000101
11111111010001ZZZ010000101
11111111110001ZZZ010000101
11111111110001ZZZ010000101

11111110110001ZZZ010000110
11111111010001ZZZ010000110
11111111110001ZZZ010000110
11111111110001ZZZ010000110

11111110110001ZZZ111100101
11111111010001ZZZ111100101
11111111110001ZZZ111100101
11111111110001ZZZ111100101

11111110110001ZZZ010000110
11111111010001ZZZ010000110
11111111110001ZZZ010000110
11111111110001ZZZ010000110
```

Appendix 4: Schematics

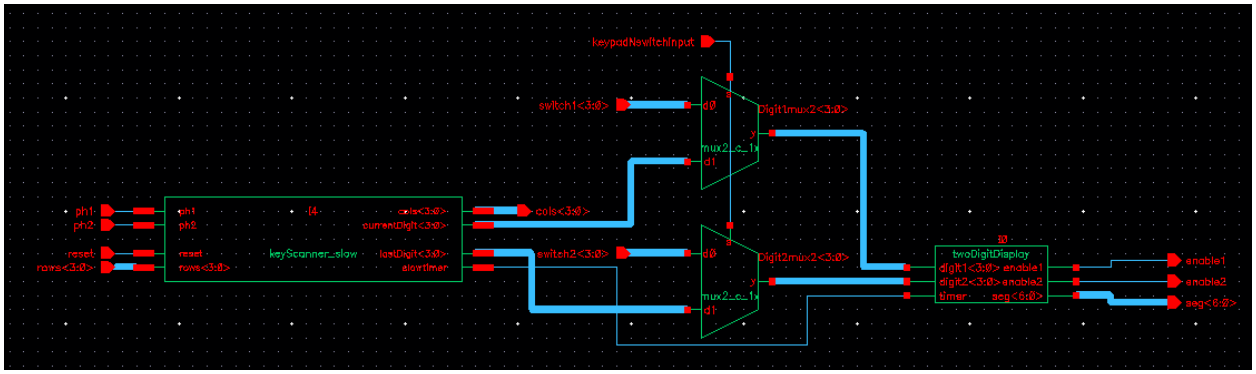
Schematic 1: Nand4



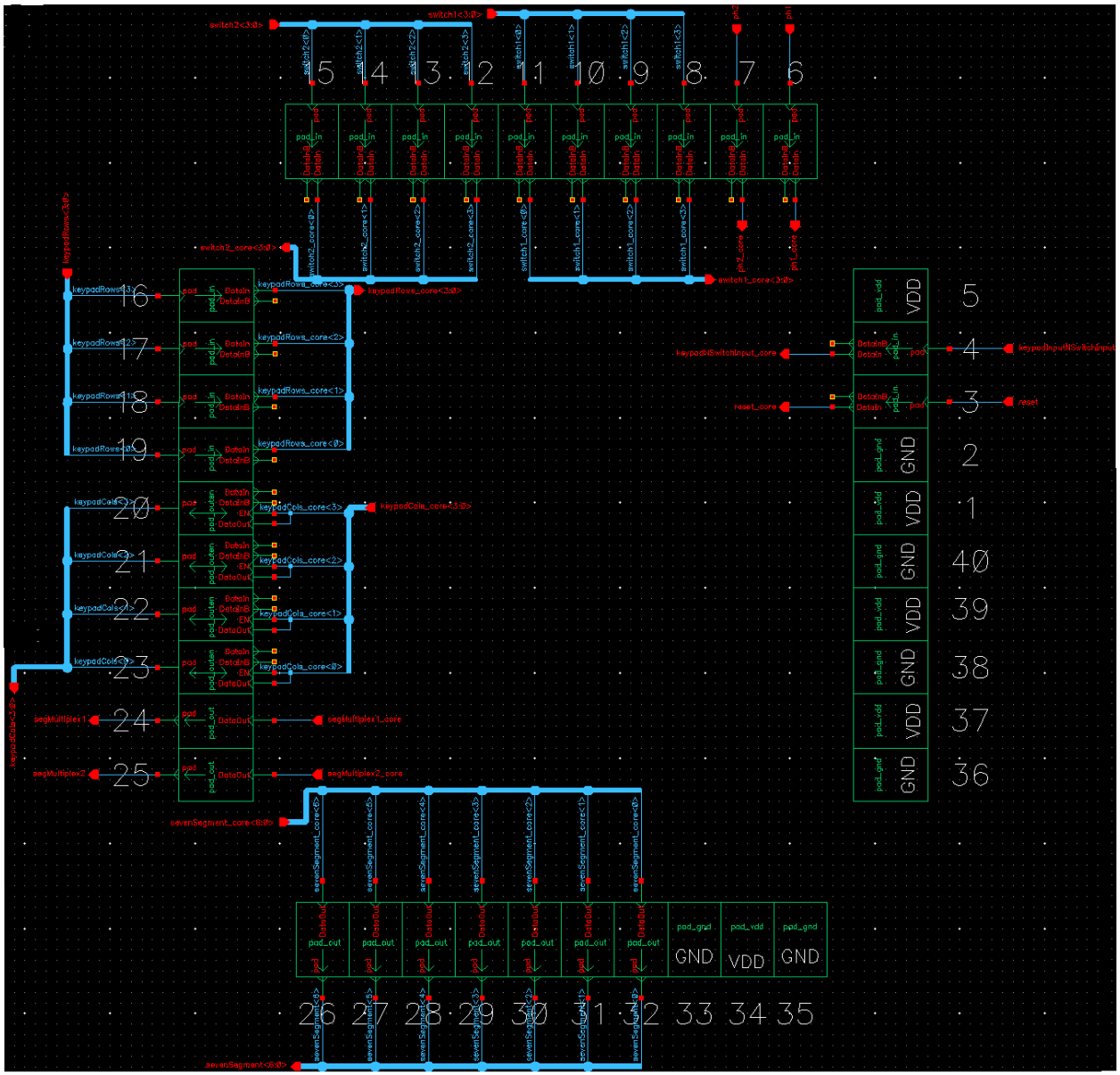
Schematic 2: Seven Segment Decoder



Schematic 3: Core Schematic



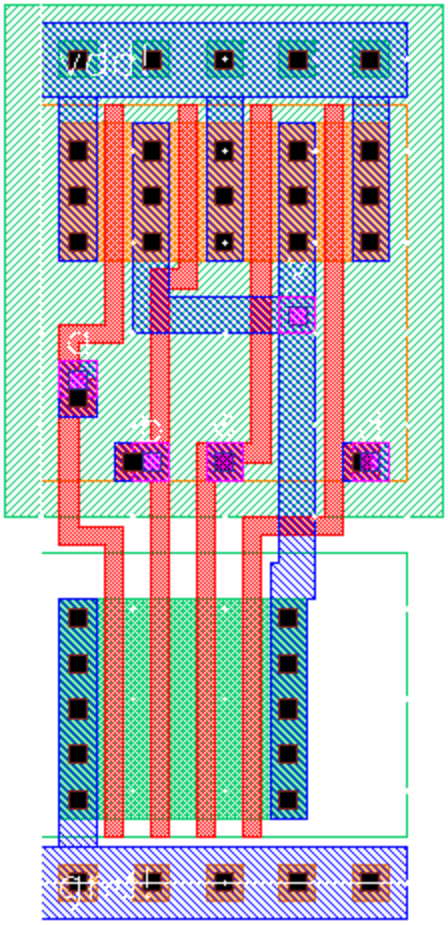
Schematic 4: Padframe



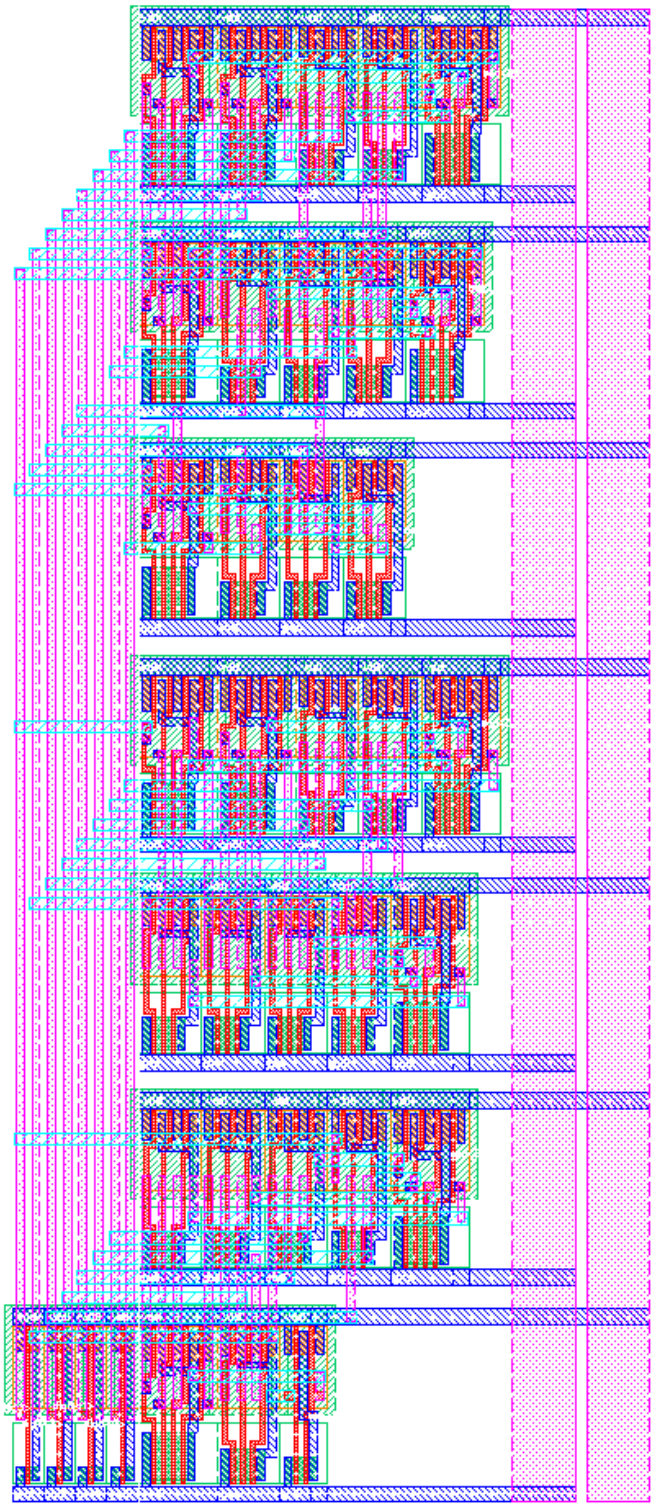
Schematic 5: Chip

Appendix 5: Layouts

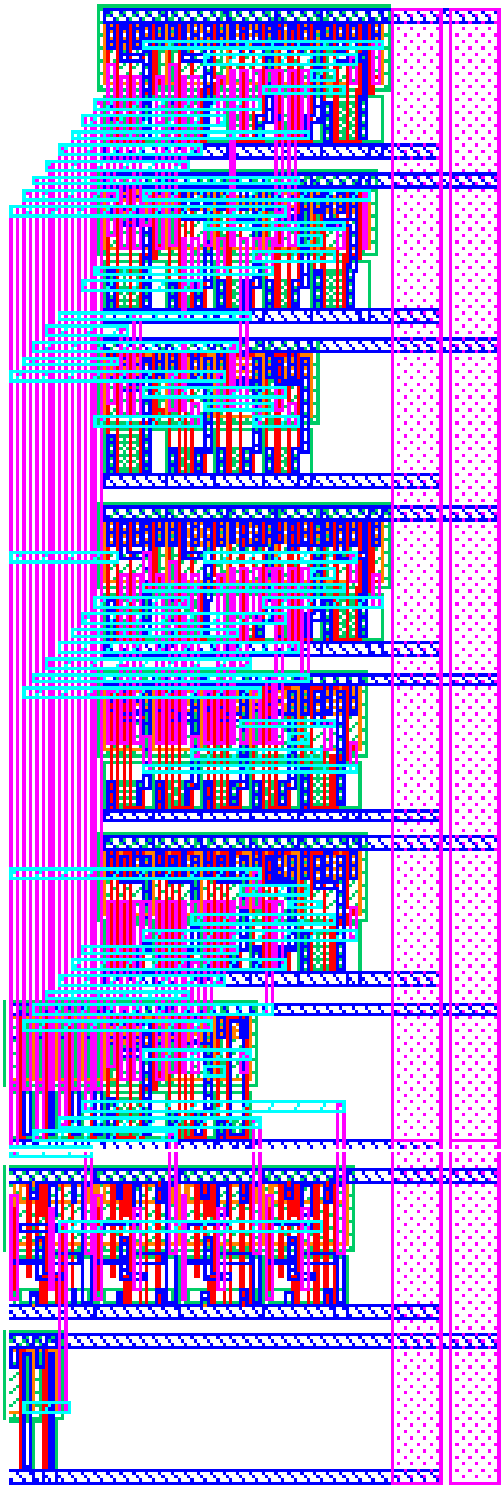
Layout 1: Nand4



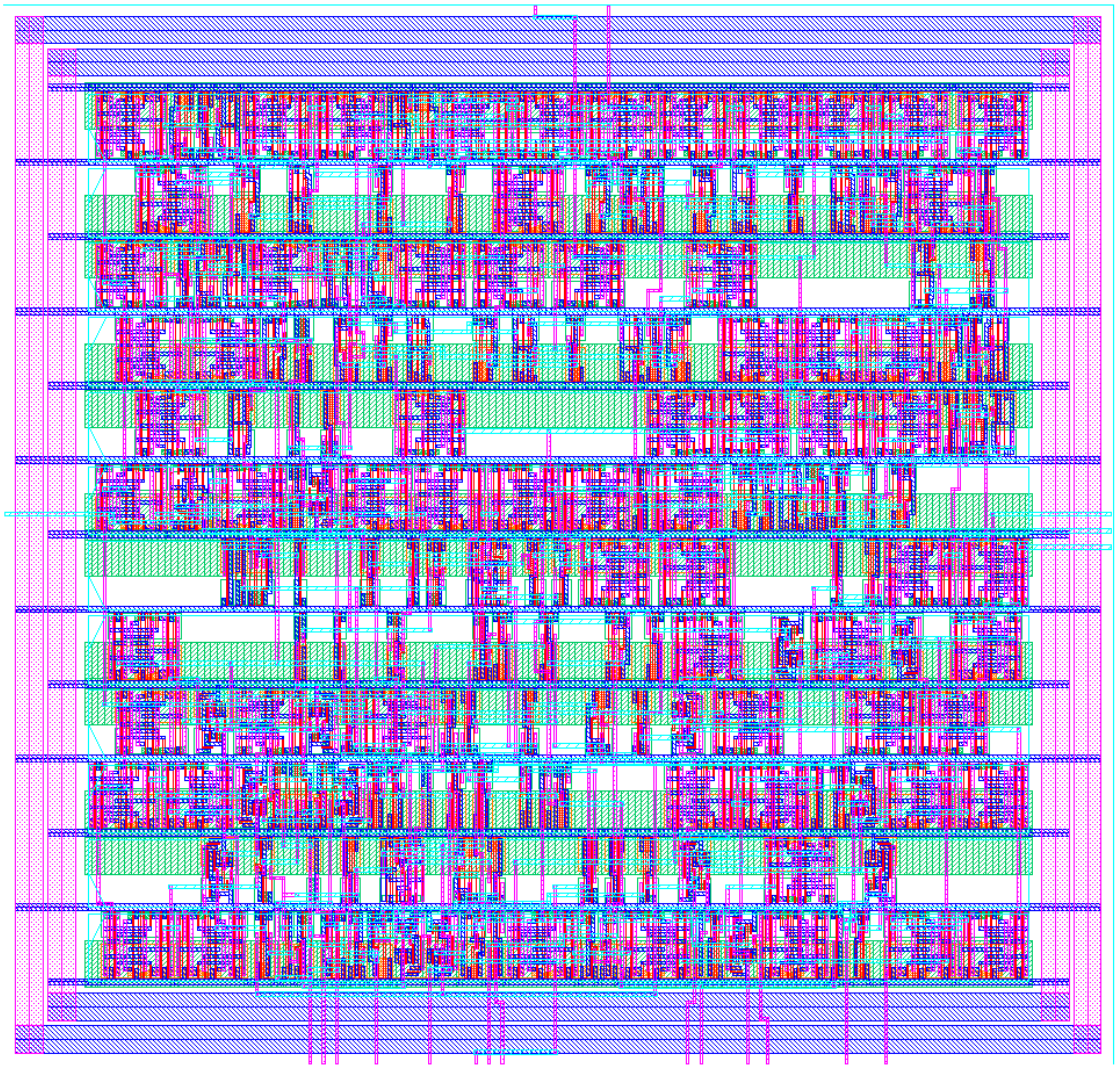
Layout 2: Seven Segment Decoder



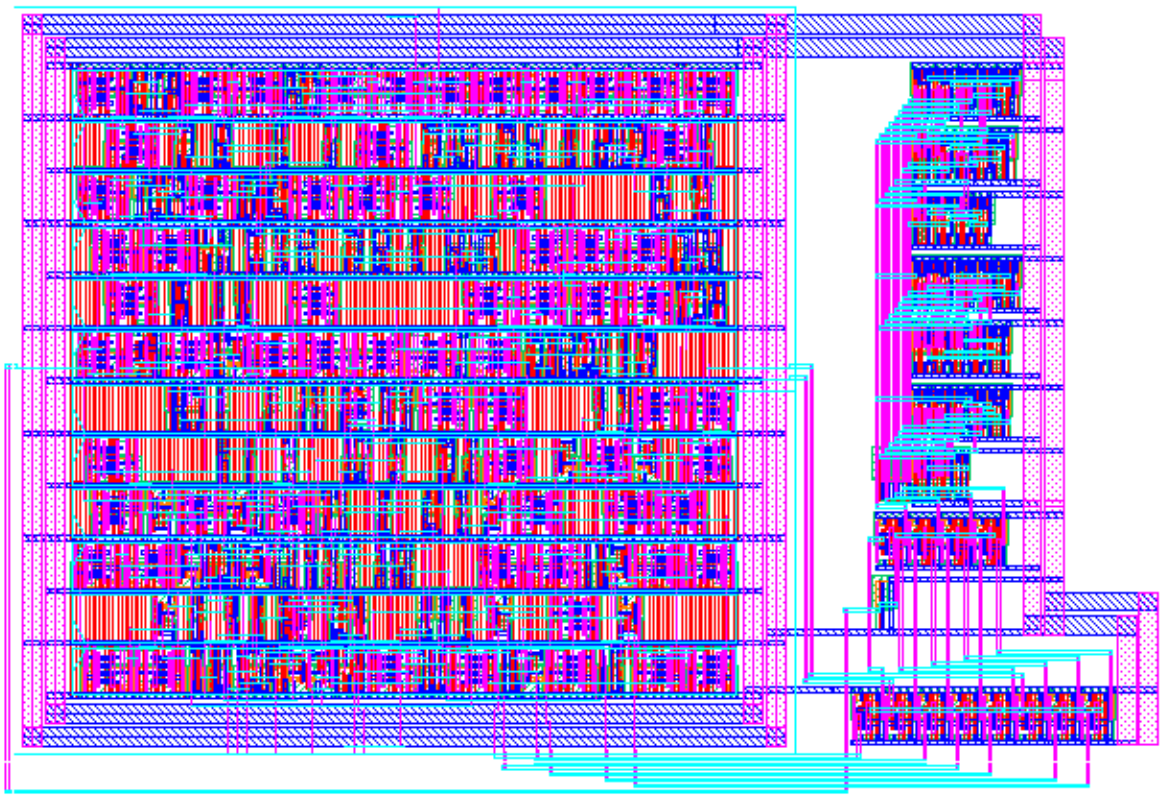
Layout 3: Two Digit Display



Layout 4: Keypad Scanner



Layout 5: Core



Layout 5: Chip