Jack Black!!

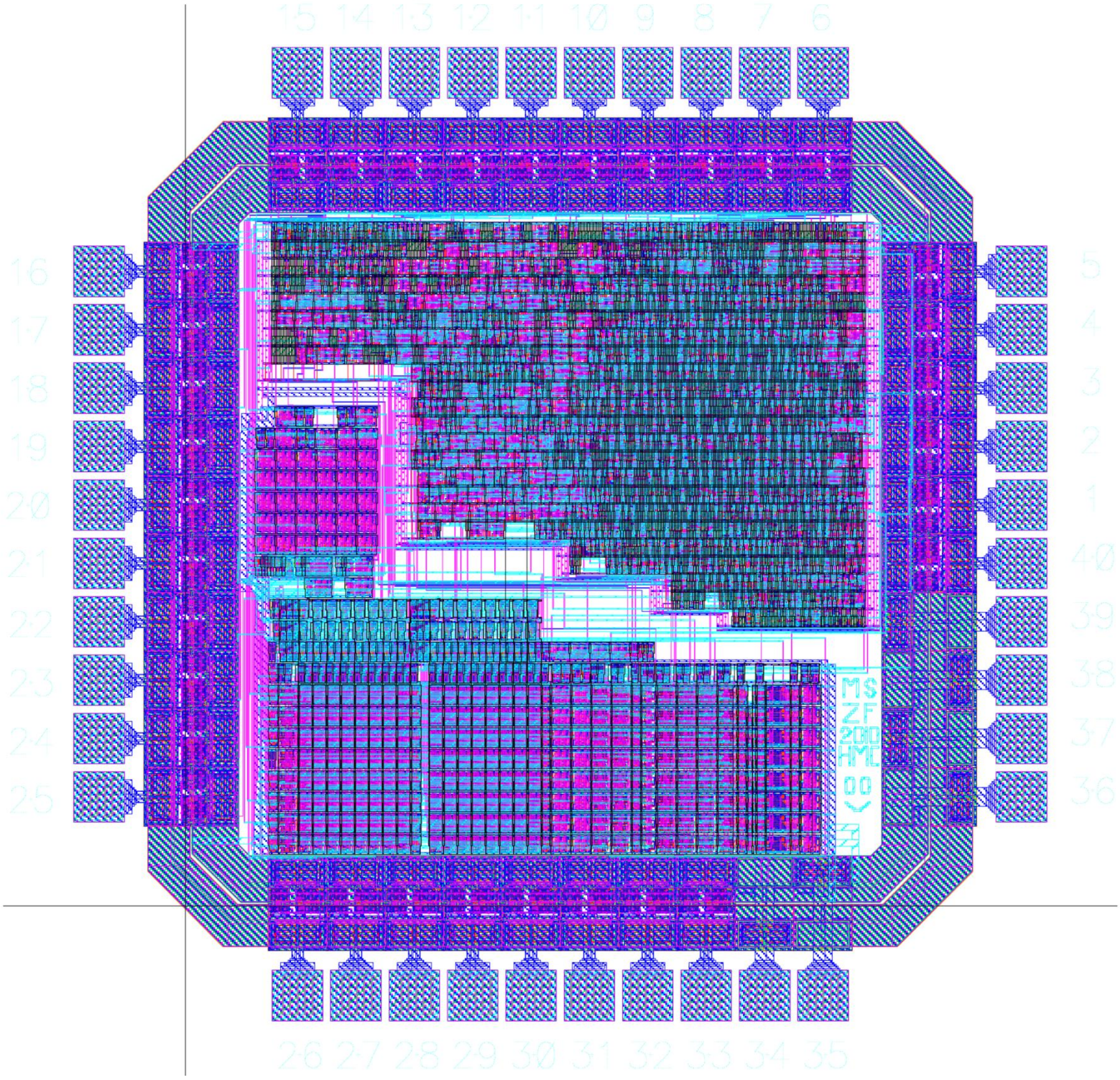
Zeke Flom

Matt Streshinsky

E158

April 19, 2010

Black Jack



Matt Streshinsky
Zeke Flom
E158
April 19, 2010

Introduction

We have designed a chip which will host a game of blackjack between a human player and computer dealer. Once implemented, the player will be able to hit, stand, and reset the game by depressing corresponding buttons. Additionally, a toggle switch will be used to choose between betting 5 credits and 10 credits. During games, the chip will display the user's balance and six cards, reserving five of the card displays to show the cards for whichever player (dealer or user) is currently making hit/stand decisions. The remaining card display shows the most recent card received by the opponent. Aside from the discretized bet amounts and absence of the option for the player to "split" or "double down," the rules are the same as in traditional blackjack. The player will win the game by increasing his/her balance up to the maximum amount of 97 credits. At this point, the balance will rollover and the player will need to hit reset to start a new game. The chip is designed for fabrication on a 40 pin 1.5 x 1.5 mm MOSIS "TinyChip" fabricated in a 0.6µm process.

Specifications

Below is a table of the inputs and outputs to the blackjack chip.

Pin Name <pad#s>	Width	Orientation	Purpose
vdd! <1,5,39>	x	x	Power Supply
gnd! <2,4,40>	x	x	Drain
reset <3>	1	Input	Starts a new game
ph1 <6>	1	Input	Clock 1
ph2 <7>	1	Input	Clock 2
hit/continue <38>	1	input	User raises this bit to hit or continue play
stand/continue <37>	1	input	User raises this bit to stand or continue play
bet amount <36>	1	input	User can toggle this bit to choose amount to bet
$ balance <8:20>	13	output	The player's balance, between 2 and 97 is sent for display
card display <35:24>	12	output	Card (Ace through King) formatted for 14 segment display
display select <23:21>	3	output	Assigns the destination display for the card

Table 1: Chip inputs and outputs

The chip consists of a synthesized controller block that governs the black jack state machine, and a custom datapath which stores game data, carries out computation, and generates random cards. The non-overlapping two phase clock inputs, ph1 and ph2, prevent hold time violations. The user inputs hit, stand, and reset will be 1-bit push switches, which signal the controller to advance to the next state by going high (for at least a clock cycle), and then returning to zero. Hit and stand will also serve as generic 'continue' buttons, allowing the user to signal for the chip to move on to the next hit/stand decision or deal a new game. Bet amount will be controlled by a 1-bit toggle switch. The display outputs 'balance' and 'card display' are decoded on chip to run two 7 segment displays and one 14 segment display respectively. These outputs are 13 and 12 bits because display segments which always remained off were omitted to free up i/o ports. Card display will run six 14 segment displays at

once by cycling between the six cards. An off-chip demux controlled by display select will send the different cards to their appropriate displays.

Controller Module

The Blackjack game was represented by the state diagram shown in Figure 1. The controller governs the transitions between these states and sends instructions to the datapath for comparison and card generation as needed. There are 11 control signals sent from the controller to the datapath and one signal (zero) runs from the datapath back to the controller. The controller decides state transitions solely based on user inputs (hit, stand, reset, and betamount) and the datapath zero logic. The state diagram in Figure 2, however, does not represent the true complexity. Within the state PBUST, for example, the controller must retrieve the player's point sum, check if it is less than 22, check for and collapse the value of an ace from 11 to 1 if necessary, and decrement the number of available aces in memory by one. Additionally, many states include substates which oblige the controller to wait for a 'continue' input from the player to go high and then low before transitioning. This allows the player to advance the game at their own pace and actually have time to read the displays. Thus, each state in the figure is actually an entire state machine consisting of several substates. In total, there are 15 states and 58 substates.

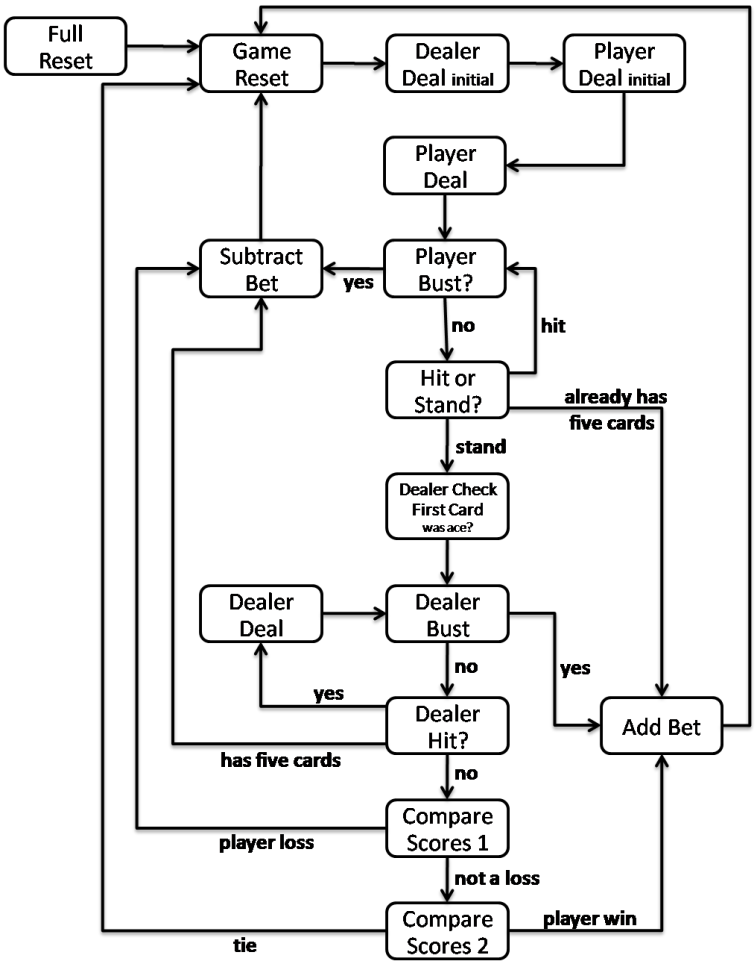


Figure 1 Master state diagram for the controller module

Display Module

The display block in the controller taps into the write data bus of the register array in the datapath in order to maintain its own bank of dealer cards, player cards, and player balance values. It decodes this data for 7 or 14 segment displays and sends it to the display outputs as appropriate. In order to properly identify the write data, the display module takes the write address of the data and the write enable control signal. In order to choose which cards to send to the six 14 segment card displays, it receives an additional signal from the controller denoting whether the player or dealer is currently taking action in the game. Whenever the write enable signal is raised, the controller stores the written data into its own registers based on the write address. Anytime a new card or dollar balance is written in the datapath the display module also receives that new value.

Since the design is limited to 40 pins, there are only enough outputs for one 14 segment and two 7 segment displays. Thus, the balance is displayed on two seven segments and the cards via six 14 segment displays which share the chip output pins by using a card select signal. Every clock cycle, the display module cycles to the next of six card values to be displayed, and updates the card select signal accordingly. Additionally, if it is currently the player's turn, five player cards and one dealer card will be sent for display whereas if the dealer is currently acting, five dealer cards and one player card will be in the six display cycle. With six 14-segment displays, the card-display LED's will have a duty cycle of 1/6. Most of the time, the player will not have five cards to display, but the nonexistent cards will remain in the duty cycle as an output of all lows to their corresponding LED's. As mentioned above, the card select information will need to be decoded off chip by a demux to send the card display bits to the appropriate 14-segment display for each clock cycle.

Since 40 pins is not enough to accommodate the inputs along with the full two seven-segment and one 14-segment outputs, the balance values and card displays were optimized to reduce the number of pins necessary. Bets can be values of either 5 or 10 credits with the balance initialized to 47. In this way, the top left segment of the ones-digit seven-segment LED is always off. Similarly, two segments of the 14-segment card displays which remain low for every possible card were omitted.

Custom Datapath

The datapath consists of a 16x8 register array, random card generator, stripped down ALU, ace detection block, flop banks, and various muxes. For this design, two mips8 8x8 register arrays were assembled side by side to store important game data (see Table 2).

Data	Address	Data	Address
Player Balance	0000	Active Player's Ace Count	1000
Player Card 1	0001	Dealer Card 1	1001
Player Card 2	0010	Dealer Card 2	1010
Player Card 3	0011	Dealer Card 3	1011
Player Card 4	0100	Dealer Card 4	1100
Player Card 5	0101	Dealer Card 5	1101
Player Card Value Sum	0110	Dealer Card Value Sum	1110
Player Number of Cards	0111	Dealer Number of Cards	1111

Table 2 Register Assignments

A mux13 takes a card as its select signal and delivers the point value of that card so that it can be added to the player or dealer's total. A mux10 allows the controller to select additional options for source b of the ALU, such as a constant value, read data from the register array, or a logic value denoting whether a card is an ace. The datapath also contains a loop which allows read data to be fed back as the read address for the next operation. Since the nth card is stored in the nth address, this allows the controller to easily access the location of the most recently dealt player or dealer card by accessing the number of cards that have been dealt to the dealer or player. This substantially reduces the number of unique states required.

Card Generator

The card generator consists of 5, 7, 11, and 13 flip-flop rings, each with a single inverter. Each ring is resettable and outputs one bit value of the random card into an enabled flip flop. A separate cell detects if the number produced by the four oscillating bits is less than 13, indicating a valid card (for cards 0-12 corresponding to Ace through King). When the card is valid, the enable signal for the random card flip flop goes high and the new card is sent to Q. While, technically, the cards are entirely deterministic (with a repeat cycle of 5x7x11x13 = 5005), the player's timing of button presses is a sufficient source of entropy when the clock rate is in the kilohertz range. A uniform distribution of the cards ace through king is difficult to achieve with such a small amount of sequential hardware, but this system comes relatively close as shown in Figure 2.

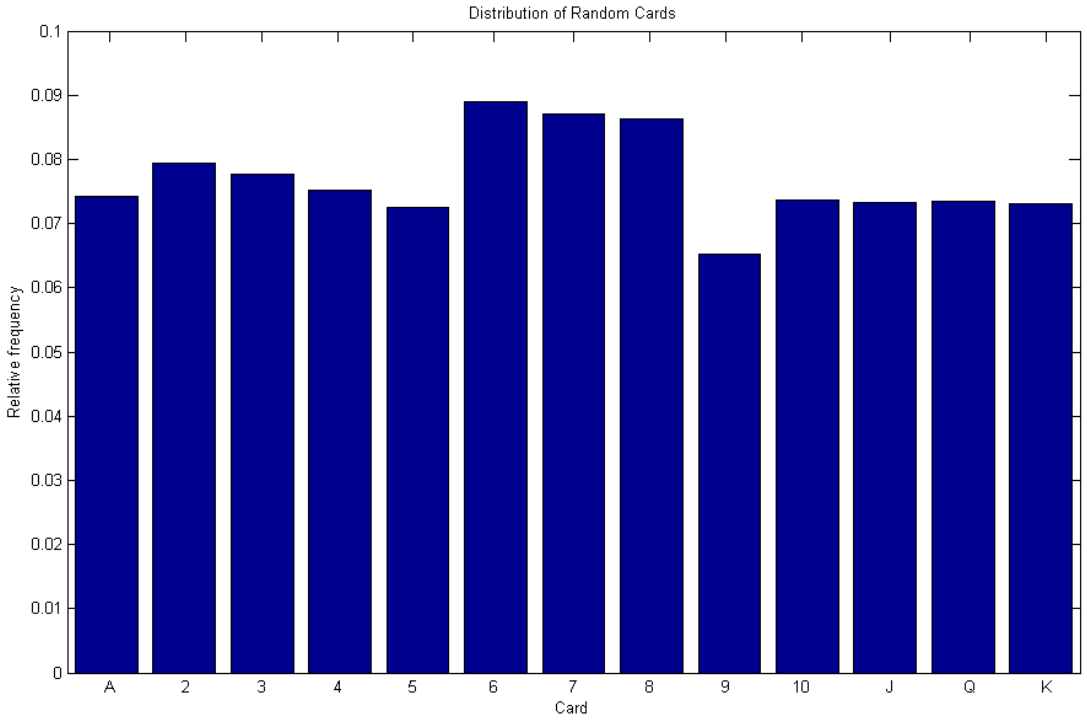


Figure 2 Card generation probability simulation

Floorplan Area

In our design proposal, we estimated the following floorplan for the blackjack chip (Figure 3).

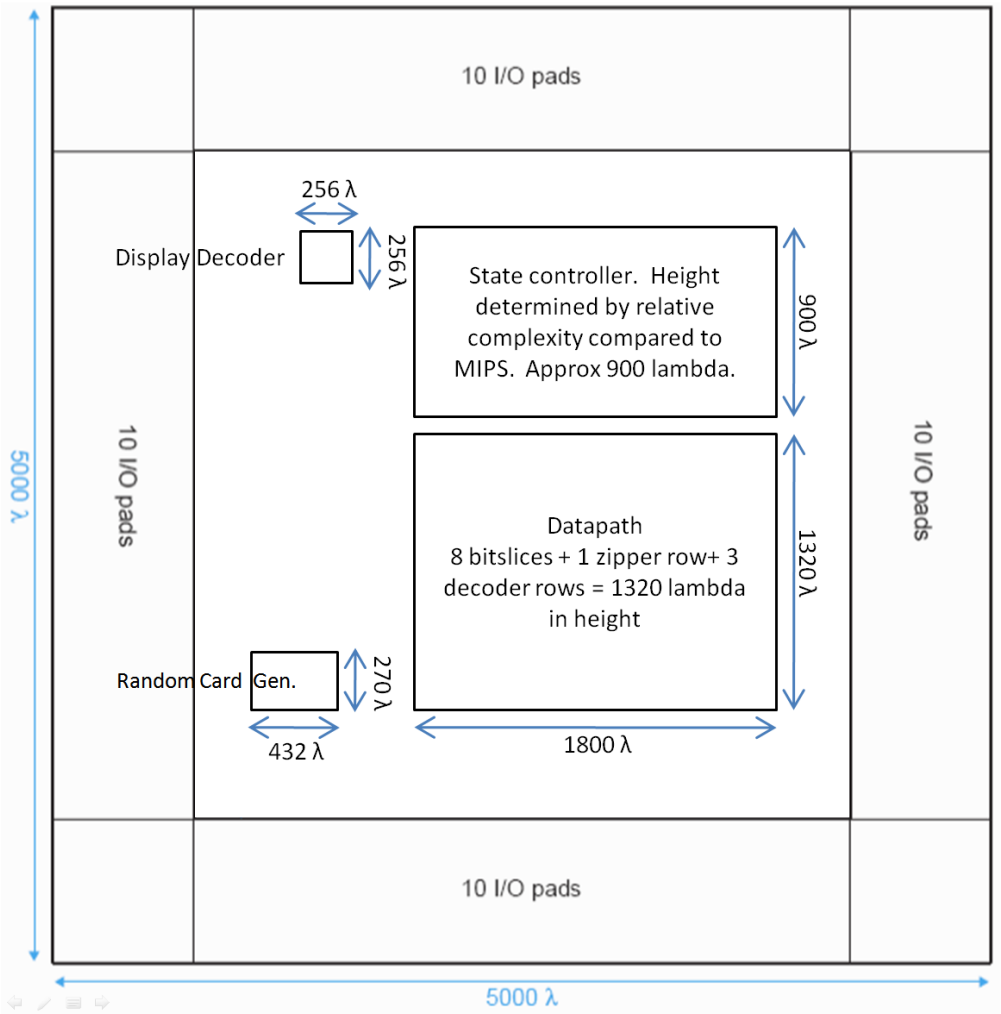


Figure 3 Preliminary floorplan estimate

However, continuing forward with the design, it become apparent the actual floorplan was much larger than we had anticipated for several key reasons.

We had originally modeled the blackjack controller as a much simpler series of state transitions. In developing a fully implemented transition diagram, we noticed many subtle points in proper execution of blackjack. Because of the multicycle datapath implementation, seemingly simple steps such as dealing a card still require several read, write and computation steps. Additionally, the possibility of an ace being valued at either 1 or 11 adds many complexities to our state machine. As a result, the actual area ended up being 5,351,000 λ^2, whereas the original estimate predicted an area of just 1,620,000 λ^2.

Our preliminary design for random card dealing relied on a bank of ring oscillators to generate random bits. Using inverters, as planned, would have required much less hardware, but we decided that the difficulties of properly integrating this design with the sequential logic

of the datapath outweighed the gains afforded in floorplan space. Originally we estimated an area of 116,640 λ^2; the final card generator layout area was 550,000 λ^2.

We also found that to we could not easily fit all of the required game data (player and dealer cards, totals, balance, etc.) into 8 registers. To make room, we added an additional 8x8 register array. Furthermore, the number of constant values we needed was larger than we had at first anticipated, requiring the design of a large mux10 and mux13. Our original datapath area estimate was 2,376,000 λ^2. The actual datapath is 4,500,000 λ^2 in area, where 850,000 λ^2 is from the additional register array.

All in all, the floorplan ended up being much tighter than expected. If the components hadn't fit on the chip we would have had to switch back to a ring oscillator based random card generator, squeeze the game data into a single 8x8 register, and/or replace the mux10 and mux13 with PLA's.

Sliceplan

Shown below is the sliceplan for the 8 bit wordslice of the datapath.

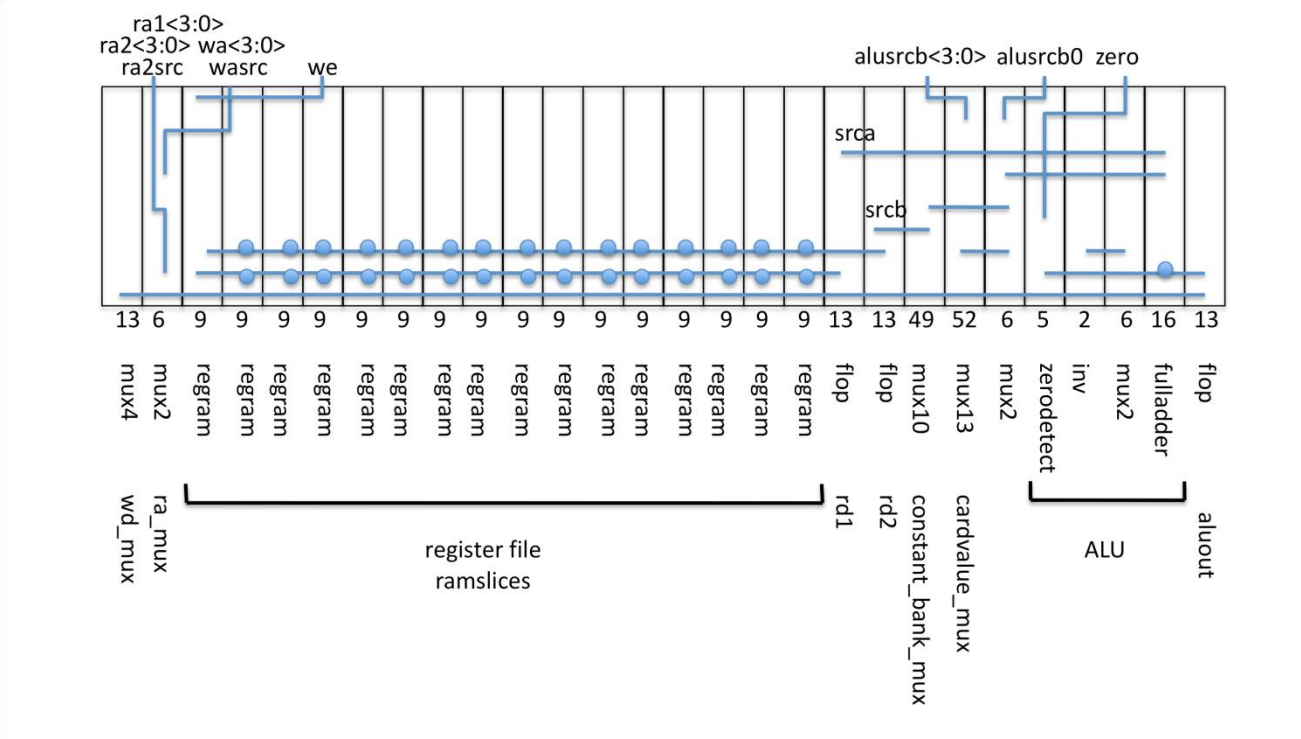


Figure 4 Datapath sliceplan

Pinout

Below is the padframe and pinout diagram for the final blackjack chip.

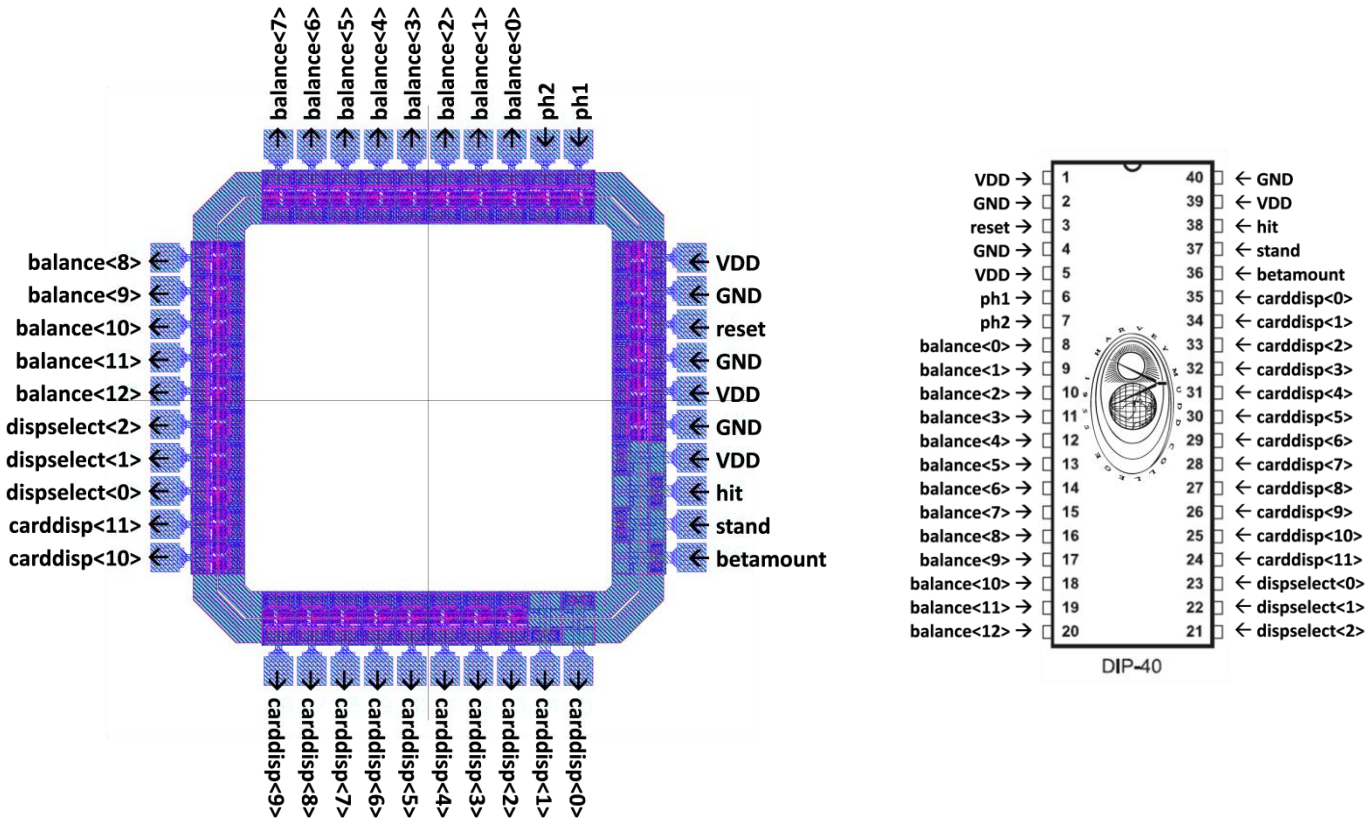


Figure 5 Padframe and pinout diagram for Blackjack chip in DIP-40

Verification

To test the verilog model and schematic netlists, we developed a series of self-checking testbenches that test different in-game logic. The principal testbench tests the entire chip by running several games of blackjack, alternating the player's choice of hit and stand, and checking that the balance display eventually reaches a correct value. The tests shown below check special cases by removing the card generator module from the netlist and replacing it with one that feeds specific cards into the game. Table 3 lists the 7 corner cases checked by each of these testbenches.

FILE	PLAYER BEHAVIOR	PLAYER CARDS	DEALER CARDS	P RESULT	D RESULT	WINNER
BJ1	hit	9 + 9 + 9	9 + 9	bust	18	D
BJ2	stand	10 + 10	10 + 10	20	20	draw
BJ3	hit hit hit	A + A + A + A + A	no turn	15	none	P
BJ4	stand	7 + 7	7 + 7 + 7	14	21	D
BJ5	stand	8 + 8	8 + 8 + 8	16	bust	P
BJ6	stand	A + 10	10 + 10	21	20	P
BJ7	stand	2 + 2	2 + 2 + 2 + 2 + 2	4	10	D

Table 3 Testbench Corner Cases

The verilog and schematic netlists pass our testbench. The CIF layouts pass DRC and LVS checks.

Post-fabrication Test Plan

The best way to verify the chip's operation after fabrication is to connect the outputs to displays on a protoboard and attempt to play a game. Table 4 lists the hardware necessary for full prototyping of the blackjack chip.

Description	Part Number	Manufacturer	Supplier	Supplier Part Number	Quantity	Unit Price	Total
14-segment LED	LDS-E3902RI	Lumex	Mouser	696-LDS-E3902RI	6	$4.26	$25.56
2 digit 7-segment	HDSP-523G	Avago Technologies	Mouser	630-HDSP-523G	1	$1.21	$1.21
1:8 demux	74HCT4051N,112	NXP Semiconductors	Mouser	771-74HCT4051N	1	$0.31	$0.31
Push-button switch	FSM4JSMA	Tyco Electronics	Mouser	506-FSM4JSMA	3	$0.35	$1.05
Tristate Bus Transceiver	SN74LVC245AN	Texas Instruments	Mouser	595-SN74LVC245AN	6	$0.40	$2.40
Toggle switch	7105SYCQE	C&K Electronics	Mouser	611-7105-002	1	$4.19	$4.19
Protoboard	923253-I	3M	Digikey	923253-ND	1	$15.95	$15.95
Op-amp	LM2902N	Fairchild Semi	Digikey	LM2902NFS-ND	5	$0.46	$2.30
							$52.97

Table 4 Bill of Materials, Prices correct as of April 17, 2010

Ideally, an enclosure will be constructed to house these components. All components will be assembled on a breadboard and rearranged into a reasonably aesthetic and compact design. Testing will begin ambitiously with the construction and play of the full prototype as outlined in the schematic shown in Figure 6.

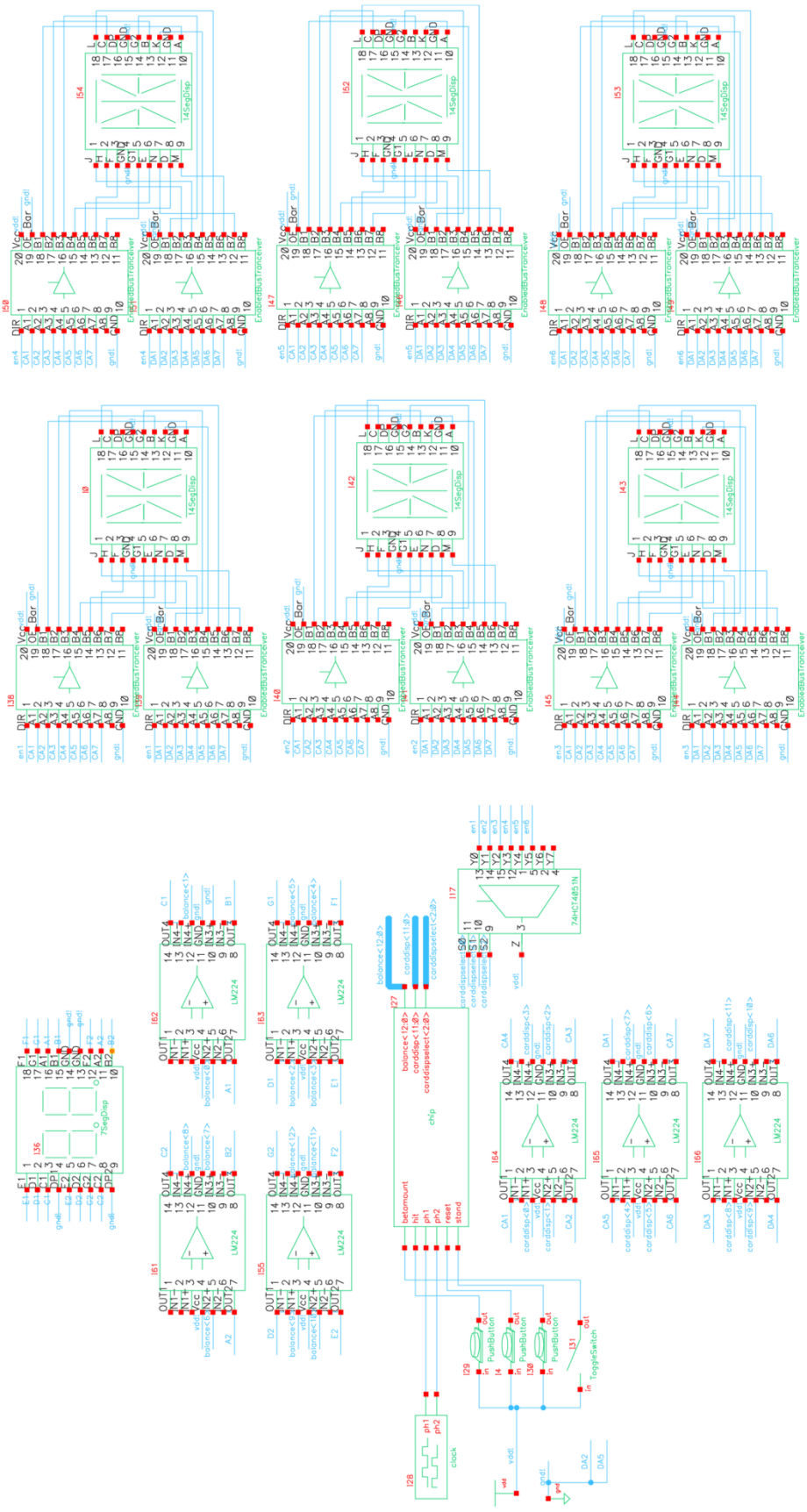


Figure 6 Final Prototype Schematic

In the event that the chip fails to operate as planned, the chip may be placed in an FPGA-based testbench in which we have the ability to control and monitor more precisely the timing of signal inputs and outputs. The full-chip verilog testbench developed for the verilog model and schematics could be adapted for such a setup. If even this effort proves unsuccessful, we can further slow down the process by stepping through clock cycles and observing how the output data changes, comparing this to expected waveforms in netlist simulations.

Shown in Table 5 is a breakdown of where design time was spent in man-hours (includes debugging except when listed separately).

	Design Time (hours)	
Component	schematic	layout
Chip	.5	4
Datapath	3	20
Controller	0.1	9
Padframe	0.75	1.5
Rings 5,7,11,13	1	3.5
Cardgen	1	2
Carden	0.5	2
Muxes	1.5	7
ALU simplified	0.2	1
Flop_1x_4	0.2	0.5
----------	write	debug
Controller	20	6
Other Verilog	40	8
Testbenches	2	1
----------	----------	--------
total		≈140

Table 5 Design time breakdown

File Locations (zflom@chips.eng.hmc.edu)

- Verilog code
  ~/FINALPROJECT/bj.sv
- Test vectors
  ~/IC_CAD/cadence/chip_run1/testall.sh
- Synthesis results
  ~/IC_CAD/soc/controller_disp_syn.v
- All Cadence libraries
  ~/IC_CAD/cadence/
- CIF
  ~/IC_CAD/cadence/bj_cifin
- PDF chip plot
  ~/bjchip.pdf
- PDF of your report
  ~/bj_finalreport.pdf

Appendix A

Verilog Code

```
//-----------------------------------------------------------
// BJ2.sv
// ZFLOM@hmc.edu 23 Jan 2008
//----------------------

// Set delay unit to 1 ns and simulation precision to 0.1 ns (100 ps)

`timescale 1ns / 100ps

// states and instructions
// testbench for testing
module testbench ();   //runs game a few times then checks that balance is correct (player has been winning or losing at expected for given input)


  logic             ph1, ph2;
  logic             reset;
  logic   hit, stand, betamount;
  logic [11:0] counter;
  logic [12:0] balancedisp;
  logic [13:0] carddisp;
  logic [2:0] carddispselect;
  // instantiate devices to be tested
  // .* notation instantiates all ports in the mips module
  // with the correspondingly named signals in this module
  bj dut(.*);

  // external memory for code and data
 // exmemory #(WIDTH) exmem(ph1, ph2, memwrite, adr, writedata, memdata);
       //`include "testfixture.verilog"
  // initialize test

 initial
    begin
                counter <= 12'b000000000000;
                hit <= 1'b0;
      reset <= 1; # 193; reset <=  0;

end

assign betamount = 0;
//assign stand = 0;

always
begin
     ph1 <= 0; ph2 <= 0; #1; // generate clock to sequence tests
     ph1 <= 1; # 4;
     ph1 <= 0; #1;
     ph2 <= 1; # 4;
     //$display("ALWAYS IS WORKING");
end

always
begin
```

```
                if (hit) begin
                        hit = 1'b0; #50;
                        stand = 1'b1;
                end
                else begin
                        stand = 1'b0; #50;
                        hit = 1'b1;

                        end
   counter = counter+1; #20;

end


always @(posedge ph2) begin
if (balancedisp == 13'b1011011111011)
begin
$display("SIM SUCEEDED");
$finish;
end
end


always @(posedge counter[11]) begin
$display("sim FAILED, YOU SUCK");
$finish;
end

endmodule


// simplified BJ processor

typedef enum logic [3:0] {
FULLSRESET,
SRESET, //game reset
PBUST,
DBUST,
DHIT,
PDEALI,
DDEALI,
PDEAL,
DDEAL,
HITSTAND,
ADDBET,
SUBBET,
COMPARESCORES1,
COMPARESCORES2,
DCHECKFIRSTACE
} statelist;

typedef enum logic [2:0] {
CARD0,
CARD1,
CARD2,
CARD3,
CARD4,
CARD5
} dispstatelist;
```

```
typedef enum logic [5:0] {
ADD0,
ADD1,
ADDACE,
ADDCARD,
ADDTOBALANCE,
BALANCECLEAR,
ACECOUNTCLEAR,
DCARD1CLEAR,
DCARD2CLEAR,
DCARD3CLEAR,
DCARD4CLEAR,
DCARD5CLEAR,
DCOLLAPSEACE,
DGETCARDDEST,
DGETCARDDEST2,
DGETCARDDEST3,
ISACE,
NUMACESCLEAR,
DNUMCARDSCLEAR,
DSUMCLEAR,
FINISH,
GETBALANCE,
GETDNUMCARDS,
GETDSUM,
GETPNUMCARDS,
GETPSUM,
GETRANDOM,
GETTOTALS,
PCARD1CLEAR,
PCARD2CLEAR,
PCARD3CLEAR,
PCARD4CLEAR,
PCARD5CLEAR,
PCOLLAPSEACE,
PGETCARDDEST,
PGETCARDDEST2,
PGETCARDDEST3,
GETACECOUNT,
GETACECOUNT2,
PNUMCARDSCLEAR,
PSUMCLEAR,
SLT,
SLT17,
SLT22,
START,
STOREBALANCE,
STOREACECOUNT,
STOREDNUMCARDS,
STOREDSUM,
STOREPNUMCARDS,
STOREPSUM,
SUB,
SUB1,
SUB5,
SUB10,
SUBTOBALANCE,
WAIT1,
```

```
WAIT2
} substatelist;


module bj (    input  logic ph1, ph2, reset, hit, stand, betamount,
        output logic [12:0] balancedisp,
        output logic [11:0] carddisp,
        output logic [2:0] carddispselect);

   logic        zero, alusrcb0, pord, ra2src, wasrc, we;
   logic [3:0]  alusrcb;
   logic [1:0]  wdsrc;
   logic [1:0]  alucontrol;
   logic [3:0] ra1, ra2, wa;
   logic [7:0]        wd;
   logic [3:0]        wdadr;

   // instantaiate controller and datapath

   // . notation connects signals by name, even if the port order
   // isn't the same when the schematic or layout editor netlists the
   // module for circuit simulation

        controller_disp       cont_disp(.ph1, .ph2, .reset, .zero, .hit, .stand, .betamount, // inputs
                              .alusrcb0, .alusrcb, .alucontrol, .ra2src, .pord, .ra1, .ra2, .wasrc, .we, .wdsrc, .wa, // control signals
                                                   .wd, .wdadr, .balancedisp, .carddispselect, .carddisp); // display signals and datapath
connection

        datapath    dp(.ph1, .ph2, .zero, .reset,
                    .alusrcb0, .alusrcb, .alucontrol, .ra2src, .pord, .ra1, .ra2, .wasrc, .we, .wdsrc, .wa, .wd , .wdadr);


endmodule


// Wrapper Module for Controller and Display
module controller_disp( input logic ph1, ph2, reset, stand, betamount, zero, hit,
                                          input logic [3:0] wdadr,
                                          input logic [7:0] wd,
                                          output logic pord, ra2src, wasrc, we, alusrcb0,
                                          output logic [3:0] alusrcb, ra1, ra2, wa,
                                          output logic [1:0] wdsrc,
                                          output logic [1:0] alucontrol,
                                          output logic [12:0] balancedisp,      // Balance Display
                                          output logic [11:0] carddisp,         // Card Display
                                          output logic [2:0] carddispselect); // Card Display Select

        logic whosturn;

        controller  cont(.ph1, .ph2, .reset, .zero,
                    .alusrcb0, .alusrcb, .alucontrol, .ra2src, .pord, .ra1, .ra2, .wasrc, .we, .wdsrc, .wa, .hit, .stand, .betamount, .whosturn);

        display        disp(.ph1, .reset, .ph2, .we, .wd, .wdadr, .balancedisp, .carddispselect, .carddisp, .whosturn);

endmodule


module display(input logic we, ph1, ph2, reset, whosturn,
                              input wire [7:0] wd,
```

```
                                input logic [3:0] wdadr,
                                output logic [12:0] balancedisp,
                                output logic [11:0] carddisp,
                                output logic [2:0] carddispselect);

logic [3:0] pcard1, pcard2, pcard3, pcard4, pcard5, dcard1, dcard2, dcard3, dcard4, dcard5;
logic [7:0] balance;
wire [7:0] balancetemp;
wire [3:0] thecard;
logic wepc1, wepc2, wepc3, wepc4, wepc5, wedc1, wedc2, wedc3, wedc4, wedc5, web;

        flopen #(4) pcard1flop(ph1,ph2,we&wepc1,wd[3:0],pcard1);
        flopen #(4) pcard2flop(ph1,ph2,we&wepc2,wd[3:0],pcard2);
        flopen #(4) pcard3flop(ph1,ph2,we&wepc3,wd[3:0],pcard3);
        flopen #(4) pcard4flop(ph1,ph2,we&wepc4,wd[3:0],pcard4);
        flopen #(4) pcard5flop(ph1,ph2,we&wepc5,wd[3:0],pcard5);
        flopen #(4) dcard1flop(ph1,ph2,we&wedc1,wd[3:0],dcard1);
        flopen #(4) dcard2flop(ph1,ph2,we&wedc2,wd[3:0],dcard2);
        flopen #(4) dcard3flop(ph1,ph2,we&wedc3,wd[3:0],dcard3);
        flopen #(4) dcard4flop(ph1,ph2,we&wedc4,wd[3:0],dcard4);
        flopen #(4) dcard5flop(ph1,ph2,we&wedc5,wd[3:0],dcard5);
        flopen #(8) balanceflop23(ph1,ph2,we&web,wd,balance);

dispstatelist dispstate, nextdispstate;
logic [2:0] nds, dstate_logic;
// resetable state register with inital value of CARD0

mux2 #(3) dstateresetmux(nextdispstate, CARD0, reset, nds);
flop #(3) dstatereg(ph1, ph2, nds, dstate_logic);
assign dispstate = dispstatelist'(dstate_logic);

//assign balancetemp = balance;
always_comb
begin
//get values
        wepc1 = 1'b0;
        wepc2 = 1'b0;
        wepc3 = 1'b0;
        wepc4 = 1'b0;
        wepc5 = 1'b0;
        wedc1 = 1'b0;
        wedc2 = 1'b0;
        wedc3 = 1'b0;
        wedc4 = 1'b0;
        wedc5 = 1'b0;
        web = 1'b0;
        case(wdadr)

        4'b0000:  web = 1'b1;

        4'b0001:  wepc1 = 1'b1;
        4'b0010:  wepc2 = 1'b1;
        4'b0011:  wepc3 = 1'b1;
        4'b0100:  wepc4 = 1'b1;
        4'b0101:  wepc5 = 1'b1;

        4'b1001:  wedc1 = 1'b1;
        4'b1010:  wedc2 = 1'b1;
        4'b1011:  wedc3 = 1'b1;
```

```
        4'b1100:  wedc4 = 1'b1;
        4'b1101:  wedc5 = 1'b1;
        default: ;
endcase

end

always_comb begin
        case(dispstate)
        CARD0:  nextdispstate = CARD1;
        CARD1:  nextdispstate = CARD2;
        CARD2:  nextdispstate = CARD3;
        CARD3:  nextdispstate = CARD4;
        CARD4:  nextdispstate = CARD5;
        CARD5:  nextdispstate = CARD0;
        default:       nextdispstate = CARD0;
        endcase
end

always_comb begin
        case(dispstate)
        CARD0:  carddispselect = 3'b000;
        CARD1:  carddispselect = 3'b001;
        CARD2:  carddispselect = 3'b010;
        CARD3:  carddispselect = 3'b011;
        CARD4:  carddispselect = 3'b100;
        CARD5:  carddispselect = 3'b101;
        default:       carddispselect = 3'b000;
        endcase
end

//balance

//assign balancedisp[5:0] = balance[0] ? 6'b111011 : 6'b000111;
mux14 displaymux(dcard1, pcard1, pcard2, pcard3, pcard4, pcard5, pcard1, pcard1,pcard1,dcard1, dcard2, dcard3, dcard4,
dcard5,{whosturn,carddispselect},thecard);

always_comb begin
        case (balance[0])
        1'b1:   balancedisp[5:0] = 6'b000111;
        1'b0:   balancedisp[5:0] = 6'b111011;
        default:balancedisp[5:0] = 6'b000000;
        endcase

    case (balance[6:1])
      6'b000000: balancedisp[12:6] = 7'b0111111;
      6'b000110: balancedisp[12:6] = 7'b0000110;  //12
      6'b001011: balancedisp[12:6] = 7'b1011011; //22
      6'b010000: balancedisp[12:6] = 7'b1001111; //3
      6'b010101: balancedisp[12:6] = 7'b1100110;//4
      6'b011011: balancedisp[12:6] = 7'b1101101;//5
      6'b011111: balancedisp[12:6] = 7'b1111101;
      6'b100100: balancedisp[12:6] = 7'b0000111;//7
      6'b101001: balancedisp[12:6] = 7'b1111111;
      6'b101110: balancedisp[12:6] = 7'b1100111;//9

      6'b000011: balancedisp[12:6] = 7'b0111111;
      6'b001000: balancedisp[12:6] = 7'b0000110;  //17
      6'b001101: balancedisp[12:6] = 7'b1011011; //27
```

```
      6'b010010: balancedisp[12:6] = 7'b1001111; //3
      6'b010111: balancedisp[12:6] = 7'b1100110;//4
      6'b011100: balancedisp[12:6] = 7'b1101101;//5
      6'b100001: balancedisp[12:6] = 7'b1111101;
      6'b100110: balancedisp[12:6] = 7'b0000111;//77
      6'b101011: balancedisp[12:6] = 7'b1111111;
      6'b110000: balancedisp[12:6] = 7'b1100111;//97
               default: balancedisp[12:6] = 7'b0000000;
       endcase


       case (thecard)     // mlkjihggfedcba  bits 3 and 6 deleted (from left)
       4'h0: carddisp = 12'b000011110111;  //A
       4'h1: carddisp = 12'b000011011011;  //2
       4'h2: carddisp = 12'b000011001111;  //3
       4'h3: carddisp = 12'b000011100110;  //4
       4'h4: carddisp = 12'b100001101001;  //5
       4'h5: carddisp = 12'b000011111101;  //6
       4'h6: carddisp = 12'b000000000111;  //7    ///TO BE FINISHED
       4'h7: carddisp = 12'b000011111111;  //8
       4'h8: carddisp = 12'b000011101111;  //9
       4'h9: carddisp = 12'b010100000001;  //T
       4'hA: carddisp = 12'b000000011110;  //J
       4'hB: carddisp = 12'b100000111111;  //Q
       4'hC: carddisp = 12'b101001110000;  //K


       default: carddisp = 12'b0;
endcase

//cards
end
endmodule



module controller(     input logic  zero, reset, ph1, ph2, hit, stand, betamount,
                                    output logic alusrcb0, pord, ra2src, wasrc, we, whosturn,
                                    output logic [1:0]     alucontrol,
                                    output logic [1:0]     wdsrc,
                                    output logic [3:0]     ra1, ra2, wa, alusrcb);

statelist state, nextstate;
substatelist substate, nextsubstate;


logic userchoice;
wire keypress, userchoicetemp;

       logic [3:0] ns, state_logic;
       logic [5:0] ns_s, substate_logic;

  // resetable state register with inital value of FETCH1
       mux2 #(4) stateresetmux(nextstate, FULLSRESET, reset, ns);
       flop #(4) statereg(ph1, ph2, ns, state_logic);
       assign state = statelist'(state_logic);

       mux2 #(6) substateresetmux(nextsubstate, START, reset, ns_s);
       flop #(6) substatereg(ph1, ph2, ns_s, substate_logic);
```

```
        assign substate = substatelist'(substate_logic);

//nothing to see here
assign keypress = hit | stand;

flop #(1) userchoiceflop1(ph1, ph2,  hit&(~stand), userchoicetemp);
flop #(1) userchoiceflop2(ph1, ph2,  userchoicetemp, userchoice);

always comb
    begin
    case (state)
        FULLSRESET:    begin
                               nextstate = FULLSRESET;
                               case(substate)
                                       START:                 nextsubstate = BALANCECLEAR;
                                       BALANCECLEAR:  nextsubstate = FINISH;
                                       FINISH:                begin
                                                                      nextstate = SRESET;
                                                                      nextsubstate = START;
                                                                      end
                                       default:               nextsubstate = START;
                               endcase
                               end

        SRESET:        begin
                               nextstate = SRESET;
                               case(substate)
                                       START:                 nextsubstate = ACECOUNTCLEAR;
                                       ACECOUNTCLEAR: nextsubstate = PSUMCLEAR;
                                       PSUMCLEAR:             nextsubstate = DSUMCLEAR;
                                       DSUMCLEAR:             nextsubstate = PCARD1CLEAR;
                                       PCARD1CLEAR:   nextsubstate = PCARD2CLEAR;
                                       PCARD2CLEAR:   nextsubstate = PCARD3CLEAR;
                                       PCARD3CLEAR:   nextsubstate = PCARD4CLEAR;
                                       PCARD4CLEAR:   nextsubstate = PCARD5CLEAR;
                                       PCARD5CLEAR:   nextsubstate = DCARD1CLEAR;
                                       DCARD1CLEAR:   nextsubstate = DCARD2CLEAR;
                                       DCARD2CLEAR:   nextsubstate = DCARD3CLEAR;
                                       DCARD3CLEAR:   nextsubstate = DCARD4CLEAR;
                                       DCARD4CLEAR:   nextsubstate = DCARD5CLEAR;
                                       DCARD5CLEAR:   nextsubstate = PNUMCARDSCLEAR;
                                       PNUMCARDSCLEAR:        nextsubstate = DNUMCARDSCLEAR;
                                       DNUMCARDSCLEAR:        nextsubstate = NUMACESCLEAR;
                                       NUMACESCLEAR:  nextsubstate = WAIT1;
                                       WAIT1:  case(keypress)
                                                                      1'b1:   begin
                                                                                     nextsubstate = WAIT2;
                                                                                     nextstate = SRESET; end

                                                                      1'b0:   begin
                                                                                     nextsubstate = WAIT1;
                                                                                     nextstate = SRESET; end
                                                                      default:nextsubstate = WAIT1;
                                                               endcase
                                       WAIT2:  case(keypress)
                                                                      1'b1:   begin
                                                                                     nextsubstate = WAIT2;
                                                                                     nextstate = SRESET; end
```

```
                                        1'b0:   begin
                                                nextsubstate = FINISH;
                                                nextstate = SRESET; end
                                endcase
                        FINISH:                 begin
                                                nextstate = DDEALI;
                                                nextsubstate = START;
                                                end
                        default:                nextsubstate = START;
                endcase
                end

PBUST:          begin
                nextstate = PBUST;
                case(substate)
                        START:                  nextsubstate = GETPSUM;
                        GETPSUM:                nextsubstate = SLT22;
                        SLT22:                  case(zero)
                                                        1'b1:           begin
                                                                        nextsubstate = GETACECOUNT;
                                                                        nextstate = PBUST; end
                                                        1'b0:           begin
                                                                        nextstate = HITSTAND;
                                                                        nextsubstate = START; end
                                                        default:        nextsubstate = START;           // Should never happen
                                                        endcase
                        GETACECOUNT:            nextsubstate = ADD0;
                        ADD0:                   case(zero)
                                                        1'b1:           begin
                                                                        nextstate = SUBBET;
                                                                        nextsubstate = START; end
                                                        1'b0:           begin
                                                                        nextsubstate = PCOLLAPSEACE;
                                                                        nextstate = PBUST; end
                                                        default:        nextsubstate = START;
                                                        endcase
                        PCOLLAPSEACE:   nextsubstate = SUB10;
                        SUB10:                  nextsubstate = STOREPSUM;
                        STOREPSUM:              nextsubstate = GETACECOUNT2;
                        GETACECOUNT2:   nextsubstate = SUB1;
                        SUB1:                   nextsubstate = STOREACECOUNT;
                        STOREACECOUNT:          nextsubstate = START;
                        default:                nextsubstate = START;                           // Should never happen
                endcase
                end

DBUST:          begin
                nextstate = DBUST;
                case(substate)
                        START:                  nextsubstate = GETDSUM;
                        GETDSUM:                nextsubstate = SLT22;
                        SLT22:                  case(zero)
                                                        1'b1:           begin
                                                                        nextsubstate = GETACECOUNT;
                                                                        nextstate = DBUST; end
                                                        1'b0:           begin
                                                                        nextstate = DHIT;
                                                                        nextsubstate = START; end
                                                        default:        nextsubstate = START;           // Should never happen
```

```
                                        endcase
                GETACECOUNT:            nextsubstate = ADD0;
                ADD0:                   case(zero)
                                                1'b1:           begin
                                                                        nextstate = ADDBET;
                                                                        nextsubstate = START; end
                                                1'b0:           begin
                                                                        nextsubstate = DCOLLAPSEACE;
                                                                        nextstate = DBUST; end
                                                default:        nextsubstate = START;
                                                endcase
                DCOLLAPSEACE:   nextsubstate = SUB10;
                SUB10:                  nextsubstate = STOREDSUM;
                STOREDSUM:              nextsubstate = GETACECOUNT2;
                GETACECOUNT2:   nextsubstate = SUB1;
                SUB1:                   nextsubstate = STOREACECOUNT;
                STOREACECOUNT:          nextsubstate = START;
                default:                nextsubstate = START;                           // Should never happen
        endcase
        end

DHIT:           begin
                nextstate = DHIT;
                case(substate)
                        START:                  nextsubstate = GETDNUMCARDS;
                        GETDNUMCARDS:   nextsubstate = SUB5;
                        SUB5:   case(zero)
                                                1'b1:   begin
                                                                nextsubstate = START;
                                                                nextstate = SUBBET; end

                                                1'b0:   begin
                                                                nextsubstate = GETDSUM;
                                                                nextstate = DHIT; end
                                                default:nextsubstate = START;
                                        endcase


                        GETDSUM:                nextsubstate = SLT17;
                        SLT17:                  case(zero)
                                                1'b1:           begin
                                                                        nextstate = COMPARESCORES1;
                                                                        nextsubstate = START; end
                                                1'b0:           begin
                                                                        nextstate = DDEAL;
                                                                        nextsubstate = START; end
                                                default:        nextsubstate = START;
                                                endcase
                        default:                nextsubstate = START;           // Should never happen.EVER.
                endcase
                end

PDEALI:         begin
                nextstate = PDEALI;
                case(substate)
                        START:                  nextsubstate = GETPNUMCARDS;
                        GETPNUMCARDS:   nextsubstate = ADD1;
                        ADD1:                   nextsubstate = STOREPNUMCARDS;
                        STOREPNUMCARDS:         nextsubstate = PGETCARDDEST;
```

```
                        PGETCARDDEST:  nextsubstate = GETRANDOM;
                        GETRANDOM:             nextsubstate = PGETCARDDEST3;
                        PGETCARDDEST3: nextsubstate = GETPSUM;
                        GETPSUM:               nextsubstate = ADDCARD;
                        ADDCARD:               nextsubstate = STOREPSUM;
                        STOREPSUM:             nextsubstate = PGETCARDDEST2;
                        PGETCARDDEST2: nextsubstate = ISACE;
                        ISACE:                 nextsubstate = ADDACE;
                        ADDACE:                nextsubstate = STOREACECOUNT;
                        STOREACECOUNT:         nextsubstate = FINISH;
                        FINISH:                begin
                                                       nextstate = PDEAL;
                                                       nextsubstate = START;
                                                       end
                        default:               nextsubstate = START;
                endcase
                end

DDEALI:         begin
                nextstate = DDEALI;
                case(substate)
                        START:                 nextsubstate = GETDNUMCARDS;
                        GETDNUMCARDS:  nextsubstate = ADD1;
                        ADD1:                  nextsubstate = STOREDNUMCARDS;
                        STOREDNUMCARDS:        nextsubstate = DGETCARDDEST;
                        DGETCARDDEST:  nextsubstate = GETRANDOM;
                        GETRANDOM:             nextsubstate = DGETCARDDEST2;
                        DGETCARDDEST2: nextsubstate = GETDSUM;
                        GETDSUM:               nextsubstate = ADDCARD;
                        ADDCARD:               nextsubstate = STOREDSUM;
                        STOREDSUM:             nextsubstate = FINISH;
                        FINISH:                begin
                                                       nextstate = PDEALI;
                                                       nextsubstate = START;
                                                       end
                        default:               nextsubstate = START;
                endcase
                end

PDEAL:          begin
                nextstate = PDEAL;
                case(substate)
                        START:                 nextsubstate = GETPNUMCARDS;
                        GETPNUMCARDS:  nextsubstate = ADD1;
                        ADD1:                  nextsubstate = STOREPNUMCARDS;
                        STOREPNUMCARDS:        nextsubstate = PGETCARDDEST;
                        PGETCARDDEST:  nextsubstate = GETRANDOM;
                        GETRANDOM:             nextsubstate = PGETCARDDEST3;
                        PGETCARDDEST3: nextsubstate = GETPSUM;
                        GETPSUM:               nextsubstate = ADDCARD;
                        ADDCARD:               nextsubstate = STOREPSUM;
                        STOREPSUM:             nextsubstate = PGETCARDDEST2;
                        PGETCARDDEST2:  nextsubstate = ISACE;
                        ISACE:                 nextsubstate = ADDACE;
                        ADDACE:                nextsubstate = STOREACECOUNT;
                        STOREACECOUNT:         nextsubstate = FINISH;
                        FINISH:                begin
                                                       nextstate = PBUST;
                                                       nextsubstate = START;
```

```
                                                end
                        default:                nextsubstate = START;
                endcase
                end

DCHECKFIRSTACE:
                begin
                nextstate = DCHECKFIRSTACE;
                case(substate)
                        START:                  nextsubstate = ACECOUNTCLEAR;
                        ACECOUNTCLEAR: nextsubstate = DGETCARDDEST;
                        DGETCARDDEST:  nextsubstate = ISACE;
                        ISACE:                  nextsubstate = ADDACE;
                        ADDACE:                 nextsubstate = STOREACECOUNT;
                        STOREACECOUNT: nextsubstate = FINISH;
                        FINISH:                 begin
                                                        nextstate = DBUST;
                                                        nextsubstate = START;
                                                        end
                        default:                nextsubstate = START;
                endcase
                end

DDEAL:          begin
                nextstate = DDEAL;
                case(substate)
                        START:                  nextsubstate = GETDNUMCARDS;
                        GETDNUMCARDS:  nextsubstate = ADD1;
                        ADD1:                   nextsubstate = STOREDNUMCARDS;
                        STOREDNUMCARDS:         nextsubstate = DGETCARDDEST;
                        DGETCARDDEST:  nextsubstate = GETRANDOM;
                        GETRANDOM:              nextsubstate = DGETCARDDEST3;
                        DGETCARDDEST3: nextsubstate = GETDSUM;
                        GETDSUM:                nextsubstate = ADDCARD;
                        ADDCARD:                nextsubstate = STOREDSUM;
                        STOREDSUM:              nextsubstate = DGETCARDDEST2;
                        DGETCARDDEST2:    nextsubstate = ISACE;
                        ISACE:                  nextsubstate = ADDACE;
                        ADDACE:                 nextsubstate = STOREACECOUNT;
                        STOREACECOUNT: nextsubstate = WAIT1;
                        WAIT1:  case(keypress)
                                                1'b1:   begin
                                                                nextsubstate = WAIT2;
                                                                nextstate = DDEAL; end

                                                1'b0:   begin
                                                                nextsubstate = WAIT1;
                                                                nextstate = DDEAL; end
                                                default:nextsubstate = WAIT1;
                                        endcase
                        WAIT2:  case(keypress)
                                                1'b1:   begin
                                                                nextsubstate = WAIT2;
                                                                nextstate = DDEAL; end

                                                1'b0:   begin
                                                                nextsubstate = FINISH;
                                                                nextstate = DDEAL; end
                                        endcase
```

```
                        FINISH:                 begin
                                                        nextstate = DBUST;
                                                        nextsubstate = START;
                                                        end
                        default:                nextsubstate = START;
                endcase
                end

HITSTAND:       begin
                nextstate = HITSTAND;
                case(substate)
                        START:                  nextsubstate = GETPNUMCARDS;
                        GETPNUMCARDS:   nextsubstate = SUB5;
                        SUB5:   case(zero)
                                                1'b1:   begin
                                                                nextsubstate = START;
                                                                nextstate = ADDBET; end

                                                1'b0:   begin
                                                                nextsubstate = WAIT1;
                                                                nextstate = HITSTAND; end
                                                default:nextsubstate = WAIT1;
                                        endcase

                        WAIT1:  case(keypress)
                                                1'b1:   begin
                                                                nextsubstate = WAIT2;
                                                                nextstate = HITSTAND; end

                                                1'b0:   begin
                                                                nextsubstate = WAIT1;
                                                                nextstate = HITSTAND; end
                                                default:nextsubstate = WAIT1;
                                        endcase
                        WAIT2:  case(keypress)
                                                1'b1:   begin
                                                                nextsubstate = WAIT2;
                                                                nextstate = HITSTAND; end

                                                1'b0:   begin
                                                                nextsubstate = FINISH;
                                                                nextstate = HITSTAND; end
                                        endcase
                        FINISH: case(userchoice)
                                                        1'b1:   begin
                                                                        nextstate = PDEAL;
                                                                        nextsubstate = START; end
                                                    1'b0:       begin
                                                                        nextstate = DCHECKFIRSTACE;
                                                                        nextsubstate = START; end
                                                        default:nextsubstate = START;
                                        endcase
                        default:                nextsubstate = START;
                        endcase
                        end

ADDBET:                 begin
                        nextstate = ADDBET;
                        case(substate)
```

```
                        START:                  nextsubstate = GETBALANCE;
                        GETBALANCE:             nextsubstate = ADDTOBALANCE;
                        ADDTOBALANCE:   nextsubstate = STOREBALANCE;
                        STOREBALANCE:   nextsubstate = WAIT1;
                        WAIT1:  case(keypress)
                                                1'b1:   begin
                                                                nextsubstate = WAIT2;
                                                                nextstate = ADDBET; end

                                                1'b0:   begin
                                                                nextsubstate = WAIT1;
                                                                nextstate = ADDBET; end
                                                default:nextsubstate = WAIT1;
                                        endcase
                        WAIT2:  case(keypress)
                                                1'b1:   begin
                                                                nextsubstate = WAIT2;
                                                                nextstate = ADDBET; end

                                                1'b0:   begin
                                                                nextsubstate = FINISH;
                                                                nextstate = ADDBET; end
                                        endcase
                        FINISH:                 begin
                                                        nextstate = SRESET;
                                                        nextsubstate = START; end
                        default:                nextsubstate = START;
                        endcase
                        end

SUBBET:                 begin
                        nextstate = SUBBET;
                        case(substate)
                        START:                  nextsubstate = GETBALANCE;
                        GETBALANCE:             nextsubstate = SUBTOBALANCE;
                        SUBTOBALANCE:   nextsubstate = STOREBALANCE;
                        STOREBALANCE:   nextsubstate = WAIT1;
                        WAIT1:  case(keypress)
                                                1'b1:   begin
                                                                nextsubstate = WAIT2;
                                                                nextstate = SUBBET; end

                                                1'b0:   begin
                                                                nextsubstate = WAIT1;
                                                                nextstate = SUBBET; end
                                                default:nextsubstate = WAIT1;
                                        endcase
                        WAIT2:  case(keypress)
                                                1'b1:   begin
                                                                nextsubstate = WAIT2;
                                                                nextstate = SUBBET; end

                                                1'b0:   begin
                                                                nextsubstate = FINISH;
                                                                nextstate = SUBBET; end
                                        endcase
                        FINISH:                 begin
                                                        nextstate = SRESET;
                                                        nextsubstate = START; end
```

```
                                default:                nextsubstate = START;
                                endcase
                                end

        COMPARESCORES1:         begin
                                nextstate = COMPARESCORES1;
                                case(substate)
                                START:                  nextsubstate = GETTOTALS;
                                GETTOTALS:              nextsubstate = SLT;
                                SLT:                    case(zero)
                                                                1'b1:   begin
                                                                        nextstate = COMPARESCORES2;
                                                                        nextsubstate = GETTOTALS; end
                                                        1'b0:       begin
                                                                        nextstate = SUBBET;
                                                                        nextsubstate = START; end
                                                                default:nextsubstate = START;
                                                        endcase
                                default:                nextsubstate = START;
                                endcase
                                end

        COMPARESCORES2:         begin
                                nextstate = COMPARESCORES2;
                                case(substate)
                                GETTOTALS:              nextsubstate = SUB;
                                SUB:                    case(zero)
                                                                1'b1:   begin
                                                                        nextstate = SRESET;
                                                                        nextsubstate = START; end
                                                                1'b0:   begin
                                                                        nextstate = ADDBET;
                                                                        nextsubstate = START; end
                                                                default:nextsubstate = GETTOTALS;
                                                        endcase
                                default:                nextsubstate = GETTOTALS;
                                endcase
                                end

        default:                begin
                                nextsubstate = START;
                                nextstate = SRESET;
                                end                                             // should never happen
        endcase
        end

always_comb
        begin

        case(state)
        FULLSRESET:     whosturn = 1'b0;
        SRESET:         whosturn = 1'b0;
        PBUST:          whosturn = 1'b0;
        DBUST:          whosturn = 1'b1;
        DHIT:           whosturn = 1'b1;
        PDEALI:         whosturn = 1'b0;
        DDEALI:         whosturn = 1'b1;
        PDEAL:          whosturn = 1'b0;
        DDEAL:          whosturn = 1'b1;
```

```
        HITSTAND:       whosturn = 1'b0;
        ADDBET:         whosturn = 1'b0;
        SUBBET:         whosturn = 1'b0;
        COMPARESCORES1:         whosturn = 1'b0;
        COMPARESCORES2:         whosturn = 1'b0;
        DCHECKFIRSTACE:         whosturn = 1'b0;
        default: whosturn = 1'b0;

endcase
end
always_comb

        begin
                alusrcb0        = 1'b0;
                alusrcb         = 4'b0000;
                alucontrol      = 2'b00;
                pord            = 1'b0;
                ra2src          = 1'b0;
                wasrc           = 1'b0;
                we                      = 1'b0;
                wdsrc           = 2'b00;
                ra1                     = 4'b0000;
                ra2             = 4'b0000;
                wa                      = 4'b0000;
        case(substate)
        ADD0:   begin
                        alucontrol = 2'b00;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b0000;
                        we = 1'b0;



                        end

        ADD1:   begin
                        alucontrol =2'b00;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b0001;
                        we = 1'b0;


                        end

        ADDACE: begin
                        alucontrol = 2'b00;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b0110;
                        we = 1'b0;


                        end

        ADDCARD:begin
                        alucontrol = 2'b00;
                        alusrcb0 = 1'b1;
                        we = 1'b0;
```

```
                end

ADDTOBALANCE: begin
                case(betamount)
                1'b0:
                        begin
                        alucontrol = 2'b00;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b1001;
                        we = 1'b0; end
                1'b1:
                        begin
                        alucontrol = 2'b00;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b1000;
                        we = 1'b0; end
                default:
                        begin
                        alucontrol = 2'b00;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b1001;
                        we = 1'b0; end
                endcase


                end

BALANCECLEAR:   begin
                        we = 1'b1;
                        wa = 4'b0000;
                        wasrc = 1'b0;
                        wdsrc = 2'b10;


                end

ACECOUNTCLEAR: begin
                        we = 1'b1;
                        wa = 4'b1000;
                        wasrc = 1'b0;
                        wdsrc = 2'b00;


                end

DCARD1CLEAR:    begin
                        we = 1'b1;
                        wa = 4'b1001;
                        wasrc = 1'b0;
                        wdsrc = 2'b10;


                end

DCARD2CLEAR:    begin
                        we = 1'b1;
                        wa = 4'b1010;
                        wasrc = 1'b0;
                        wdsrc = 2'b10;
```

```
                end

DCARD3CLEAR:    begin
                        we = 1'b1;
                        wa = 4'b1011;
                        wasrc = 1'b0;
                        wdsrc = 2'b10;


                end

DCARD4CLEAR:    begin
                        we = 1'b1;
                        wa = 4'b1100;
                        wasrc = 1'b0;
                        wdsrc = 2'b10;


                end

DCARD5CLEAR:    begin
                        we = 1'b1;
                        wa = 4'b1101;
                        wasrc = 1'b0;
                        wdsrc = 2'b10;


                end

GETDSUM:        begin
                        we = 1'b0;
                        ra1 = 4'b1110;
                        ra2src          = 1'b1;



                        end

DGETCARDDEST:   begin
                        we = 1'b0;
                        pord = 1'b1;
                        ra2 = 4'b1111;
                        ra2src          = 1'b0;



                        end

DGETCARDDEST2:  begin
                        we = 1'b0;
                        pord = 1'b1;
                        ra2 = 4'b1111;
                        ra2src          = 1'b0;



                        end
```

```
DGETCARDDEST3: begin
                        we = 1'b0;
                        pord = 1'b1;
                        ra2 = 4'b1111;
                        ra2src          = 1'b0;



                        end

GETACECOUNT:    begin
                        we = 1'b0;
                        ra1 = 4'b1000;


                        end

GETACECOUNT2:   begin
                        we = 1'b0;
                        ra1 = 4'b1000;



                        end

ISACE:          begin
                        we = 1'b0;
                        ra2src = 1'b1;
                        ra1 = 4'b1000;



                        end

NUMACESCLEAR:   begin
                        we = 1'b1;
                        wa = 4'b1000;
                        wasrc = 1'b0;
                        wdsrc = 2'b00;



                        end

DNUMCARDSCLEAR:         begin
                        we = 1'b1;
                        wa = 4'b1111;
                        wasrc = 1'b0;
                        wdsrc = 2'b00;



                        end

DSUMCLEAR:      begin
                        we = 1'b1;
                        wa = 4'b1110;
                        wasrc = 1'b0;
```

```
                        wdsrc = 2'b00;


                        end
FINISH:         begin
                        we = 1'b0;


                        end
GETBALANCE:     begin
                        we = 1'b0;
                        ra1 = 4'b0000;


                        end
GETDNUMCARDS:   begin
                        we = 1'b0;
                        ra1 = 4'b1111;


                        end
GETPNUMCARDS:   begin
                        we = 1'b0;
                        ra1 = 4'b0111;


                        end
GETPSUM:        begin
                        we = 1'b0;
                        ra1 = 4'b0110;
                        //ra2 = 4'b
                        ra2src          = 1'b1;


                        end
GETRANDOM:      begin
                        we = 1'b1;
                        wasrc = 1'b1;
                        wdsrc = 2'b01;


                        end
GETTOTALS:      begin
                        we = 1'b0;
                        ra1 = 4'b0110;
                        wasrc = 1'b0;
```

```
                        ra2 = 4'b1110;
                        ra2src          = 1'b0;


                        end

PCARD1CLEAR:    begin
                        we = 1'b1;
                        wa = 4'b0001;
                        wasrc = 1'b0;
                        wdsrc = 2'b10;


                        end

PCARD2CLEAR:    begin
                        we = 1'b1;
                        wa = 4'b0010;
                        wasrc = 1'b0;
                        wdsrc = 2'b10;


                        end

PCARD3CLEAR:    begin
                        we = 1'b1;
                        wa = 4'b0011;
                        wasrc = 1'b0;
                        wdsrc = 2'b10;


                        end

PCARD4CLEAR:    begin
                        we = 1'b1;
                        wa = 4'b0100;
                        wasrc = 1'b0;
                        wdsrc = 2'b10;


                        end

PCARD5CLEAR:    begin
                        we = 1'b1;
                        wa = 4'b0101;
                        wasrc = 1'b0;
                        wdsrc = 2'b10;


                        end

PCOLLAPSEACE:   begin
                        we = 1'b0;
```

```
                        ra1 = 4'b0110;


                        end

DCOLLAPSEACE:   begin
                        we = 1'b0;
                        ra1 = 4'b1110;


                        end

PGETCARDDEST:   begin
                        we = 1'b0;
                        pord = 1'b0;
                        ra2 = 4'b0111;
                        ra2src = 1'b0;


                        end

PGETCARDDEST2: begin
                        we = 1'b0;
                        pord = 1'b0;
                        ra2 = 4'b0111;
                        ra2src = 1'b0;


                        end

PGETCARDDEST3: begin
                        we = 1'b0;
                        pord = 1'b0;
                        ra2 = 4'b0111;
                        ra2src = 1'b0;


                        end

PSUMCLEAR:      begin
                        we = 1'b1;
                        wa = 4'b0110;
                        wasrc = 1'b0;
                        wdsrc = 2'b00;


                        end

PNUMCARDSCLEAR:         begin
                        we = 1'b1;
                        wa = 4'b0111;
                        wasrc = 1'b0;
                        wdsrc = 2'b00;
```

```
                        end

SLT:            begin
                        alucontrol = 2'b11;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b0111;
                        we = 1'b0;



                        end

SLT17:          begin
                        alucontrol = 2'b11;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b0011;
                        we = 1'b0;



                        end

SLT22:          begin
                        alucontrol = 2'b11;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b0101;
                        we = 1'b0;



                        end

START:          begin
                        we = 1'b0;


                        end

STOREBALANCE:   begin
                        we = 1'b1;
                        wasrc = 1'b0;
                        wdsrc = 2'b11;
                        wa = 4'b0000;
                        ra1 = 4'b1010;
                        ra2 = 4'b1010;



                        end

STOREDNUMCARDS:         begin
                        we = 1'b1;
                        wasrc = 1'b0;
                        wdsrc = 2'b11;
                        wa = 4'b1111;
```

```
                        end

STOREDSUM:      begin
                        we = 1'b1;
                        wasrc = 1'b0;
                        wdsrc = 2'b11;
                        wa = 4'b1110;



                        end

STOREACECOUNT: begin
                        we = 1'b1;
                        wasrc = 1'b0;
                        wdsrc = 2'b11;
                        wa = 4'b1000;



                        end

STOREPNUMCARDS:         begin
                        we = 1'b1;
                        wasrc = 1'b0;
                        wdsrc = 2'b11;
                        wa = 4'b0111;



                        end

STOREPSUM:      begin
                        we = 1'b1;
                        wasrc = 1'b0;
                        wdsrc = 2'b11;
                        wa = 4'b0110;

                        end

SUB:            begin
                        alucontrol = 2'b10;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b0111;
                        we = 1'b0;



                        end

SUB1:           begin
                        alucontrol = 2'b10;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b0001;
```

```
                        we = 1'b0;


                        end

SUB10:          begin
                        alucontrol = 2'b10;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b1000;
                        we = 1'b0;


                        end

SUBTOBALANCE: begin
                case(betamount)
                1'b0:
                        begin
                        alucontrol = 2'b10;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b1001;
                        we = 1'b0; end
                1'b1:
                        begin
                        alucontrol = 2'b10;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b1000;
                        we = 1'b0; end
                default:
                        begin
                        alucontrol = 2'b10;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b1001;
                        we = 1'b0; end
                endcase


                end

WAIT1:  begin
                we = 1'b0;


                end

SUB5:           begin
                        alucontrol = 2'b10;
                        alusrcb0 = 1'b0;
                        alusrcb = 4'b1001;
                        we = 1'b0;


                        end

WAIT2:  begin
                we = 1'b0;
```

```
                        end

        default: begin
                        we = 1'b0;



                        end

endcase
end
endmodule



module datapath  (input  logic             ph1, ph2, ra2src, wasrc, pord, alusrcb0, we, reset,
                  input  logic [1:0]       wdsrc, //inputs and outputs
                  input  logic [3:0]       alusrcb, ra1, ra2, wa,
                  input  logic [1:0]       alucontrol,
                  output logic             zero,
                               output logic [7:0] wd,
                               output wire [3:0] wdadr);



  wire [7:0]  rd1, rd2, a, b, srcb, aluresult, constantbank, cardvalue, aluout,  wasace, rcard; //logic
  wire [3:0]  a2, carddest;

        logic [7:0] C0, C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11,
                            C17, C22, C42;

        assign C0 =  8'b00000000;
        assign C1 =  8'b00000001; //constants
        assign C10 = 8'b00001010;
        assign C2 =  8'b00000010;
        assign C22 = 8'b00010110;
        assign C17 = 8'b00010001;
        assign C4 =  8'b00000100;
        assign C42 = 8'b00101111;
        assign C3 =  8'b00000011;
        assign C5 =  8'b00000101;
        assign C6 =  8'b00000110;
        assign C7 =  8'b00000111;
        assign C8 =  8'b00001000;
        assign C9 =  8'b00001001;
        assign C11 = 8'b00001011;
        assign wasace = {7'b0000000, (1'b1 & ~srcb[0]) & (1'b1 & ~srcb[1]) & (1'b1 & ~srcb[2]) & (1'b1 & ~srcb[3]) & (1'b1 & ~srcb[4]) & (1'b1 &
~srcb[5]) & (1'b1 & ~srcb[6]) & (1'b1 & ~srcb[7])} ;
  //meaningless comment
  //cardgen    cg(ph1,rcard); //fake random
  cardgen      cg(reset,ph1,ph2,rcard); //real random jk

  mux2       #(4) a2mux(ra2, carddest, ra2src, a2);
  mux2       #(4) wdadrmux(wa, carddest, wasrc, wdadr);
  mux2       bmux(constantbank, cardvalue, alusrcb0, b);
  mux4          #(8) wdmux(C0,rcard,C42,aluout,wdsrc,wd);
```

```
  mux10         constantbankmux(C0,C1,C2,C17,C4,C22,wasace,srcb,C10,C5,alusrcb,constantbank);
  mux13         cardvaluemux(C11,C2,C3,C4,C5,C6,C7,C8,C9,C10,C10,C10,C10,srcb,cardvalue);

  //flop       #(8) rcardreg(ph1, ph2, {4'b0000, rcard}, rcardin);
  flop       #(4) carddestreg(ph1, ph2, {pord, rd2[2:0]}, carddest);
//assign carddest = {pord,rd1[2:0]};
  flop       #(8) areg(ph1, ph2, rd1, a);
  flop       #(8) breg(ph1, ph2, rd2, srcb);
  flop       #(8) alureg(ph1, ph2, aluresult, aluout);

  fullreg    rf(.ph1, .ph2, .we, .ra1, .a2, .wdadr, .wd, .rd1, .rd2);

  alu        #(8) alunit(a, b, alucontrol, aluresult, zero);
endmodule


module alu #(parameter WIDTH = 8)
            (input  logic [WIDTH-1:0] a, b,
             input  logic [1:0]       alucontrol,
             output logic [WIDTH-1:0] result,
             output logic             zero);

  logic [WIDTH-1:0] b2, andresult, orresult, sumresult, sltresult;

 // andN    andblock(a, b, andresult);
  //orN     orblock(a, b, orresult);
  condinv binv(b, alucontrol[1], b2);
  adder   addblock(a, b2, alucontrol[1], sumresult);
  // slt should be 1 if most significant bit of sum is 1
  assign sltresult = sumresult[WIDTH-1];

  mux2 resultmux(sumresult, sltresult, alucontrol[0], result);
  zerodetect #(WIDTH) zd(result, zero);
endmodule

module regfile #(parameter WIDTH = 8, REGBITS = 3)
                (input  logic              ph1, ph2,
                 input  logic              regwrite,
                 input  logic [REGBITS-1:0] ra1, a2, wa,
                 input  logic [WIDTH-1:0]   wd,
                 output logic [WIDTH-1:0]   rd1, rd2);

   logic [WIDTH-1:0] RAM [2**REGBITS-1:0];

  // three ported register file
  // read two ports combinationally
  // write third port during phase2 (second half-cycle)
  // register 0 NOT hardwired to 0
  always @(ph2)
    if (regwrite) RAM[wa] <= wd;

  assign rd1 = RAM[ra1] ;
  assign rd2 = RAM[a2] ;
endmodule

module fullreg (input wire ph1, ph2, //was logic
                            input wire we,
                            input wire [3:0] ra1, a2, wdadr,
                            input wire [7:0] wd, //was logic
```

```
                            output wire [7:0] rd1, rd2); //was logic
        wire wed, wep;
    wire [7:0] rd1d, rd2d, rd1p, rd2p;
        assign wed = we & wdadr[3];
        assign wep = we & ~wdadr[3];
        regfile    rfd(ph1, ph2, wed, ra1[2:0], a2[2:0], wdadr[2:0], wd, rd1d, rd2d);
        regfile    rfp(ph1, ph2, wep, ra1[2:0], a2[2:0], wdadr[2:0], wd, rd1p, rd2p);
        mux2    rd1mux(rd1p,rd1d,ra1[3],rd1);
        mux2    rd2mux(rd2p,rd2d,a2[3],rd2);


endmodule

module zerodetect #(parameter WIDTH = 8)
                   (input  logic [WIDTH-1:0] a,
                    output logic             y);

   assign y = (a==0);
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;

  latch #(WIDTH) master(ph2, d, mid);
  latch #(WIDTH) slave(ph1, mid, q);
endmodule

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

  logic [WIDTH-1:0] d2;

  mux2 #(WIDTH) enmux(q, d, en, d2);
  flop #(WIDTH) f(ph1, ph2, d2, q);
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 latch #(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 mux2 #(parameter WIDTH = 8)
             (input  logic [WIDTH-1:0] d0, d1,
              input  logic             s,
              output logic [WIDTH-1:0] y);

  assign y = s ? d1 : d0;
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

module mux4 #(parameter WIDTH = 8)
             (input  logic [WIDTH-1:0] d0, d1, d2, d3,
              input  logic [1:0]       s,
              output logic [WIDTH-1:0] y);

  always comb
    case (s)
      2'b00: y = d0;
      2'b01: y = d1;
      2'b10: y = d2;
      2'b11: y = d3;
    endcase
endmodule

module andN #(parameter WIDTH = 8)
             (input  logic [WIDTH-1:0] a, b,
              output logic [WIDTH-1:0] y);

  assign y = a & b;
endmodule

module orN #(parameter WIDTH = 8)
            (input  logic [WIDTH-1:0] a, b,
             output logic [WIDTH-1:0] y);

  assign y = a | b;
endmodule

module inv #(parameter WIDTH = 8)
            (input  logic [WIDTH-1:0] a,
             output logic [WIDTH-1:0] y);

  assign y = ~a;
```

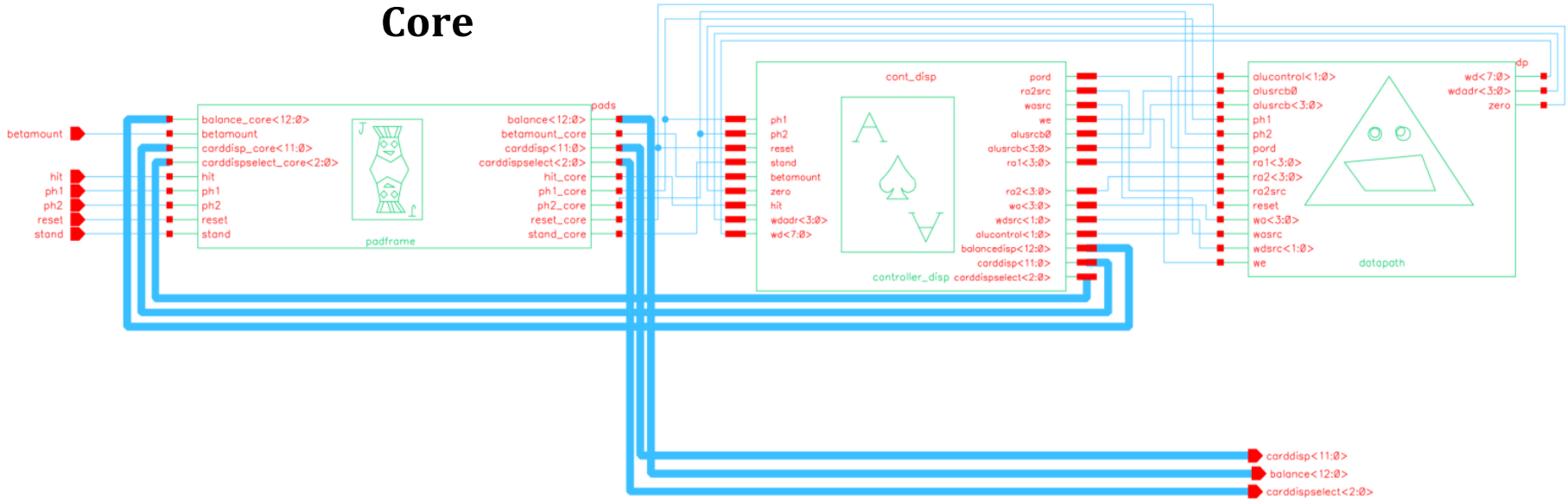
```
endmodule

module condinv #(parameter WIDTH = 8)
                (input  logic [WIDTH-1:0] a,
                 input  logic             invert,
                 output logic [WIDTH-1:0] y);

  logic [WIDTH-1:0] ab;

  inv  inverter(a, ab);
  mux2 invmux(a, ab, invert, y);
endmodule

module adder #(parameter WIDTH = 8)
              (input  logic [WIDTH-1:0] a, b,
               input  logic             cin,
               output logic [WIDTH-1:0] y);

  assign y = a + b + cin;
endmodule


module cardgen (input logic reset, ph1, ph2,
                            output logic [7:0] rcard);

wire en, bit4, bit3, bit2, bit1;
wire [3:0] rcard0;

ring5   rng4(reset, ph1, ph2, bit4);
ring7   rng3(reset, ph1, ph2, bit3);
ring11  rng2(reset, ph1, ph2, bit2);
ring13  rng1(reset, ph1, ph2, bit1);



assign en = (~bit4 | ~bit3 | (~bit2 & ~bit1));

//assign rcard = {bit4, bit3, bit2, bit1};
assign rcard0 = {bit4, bit3, bit2, bit1};
flopen rcardflopen(ph1, ph2, en, {4'b0000, rcard0}, rcard);
//rlatch       rcardreg(en, rcard0, rcard);
endmodule

module ring7 (input logic reset,ph1,ph2,
                            output logic bit3);
wire t1,t2 ,t3 ,t4,t5,t6,t7;
flop       #(1) r7f1(ph1, ph2, (~t1)|reset, t2);
flop       #(1) r7f2(ph1, ph2, t2, t3);
flop       #(1) r7f3(ph1, ph2, t3, t4);
flop       #(1) r7f4(ph1, ph2, t4, t5);
flop       #(1) r7f5(ph1, ph2, t5, t6);
flop       #(1) r7f6(ph1, ph2, t6, t7);
flop       #(1) r7f7(ph1, ph2, t7, t1);
assign bit3 = t1;

endmodule

module ring5 (input logic reset,ph1,ph2,
                            output logic bit4);
```

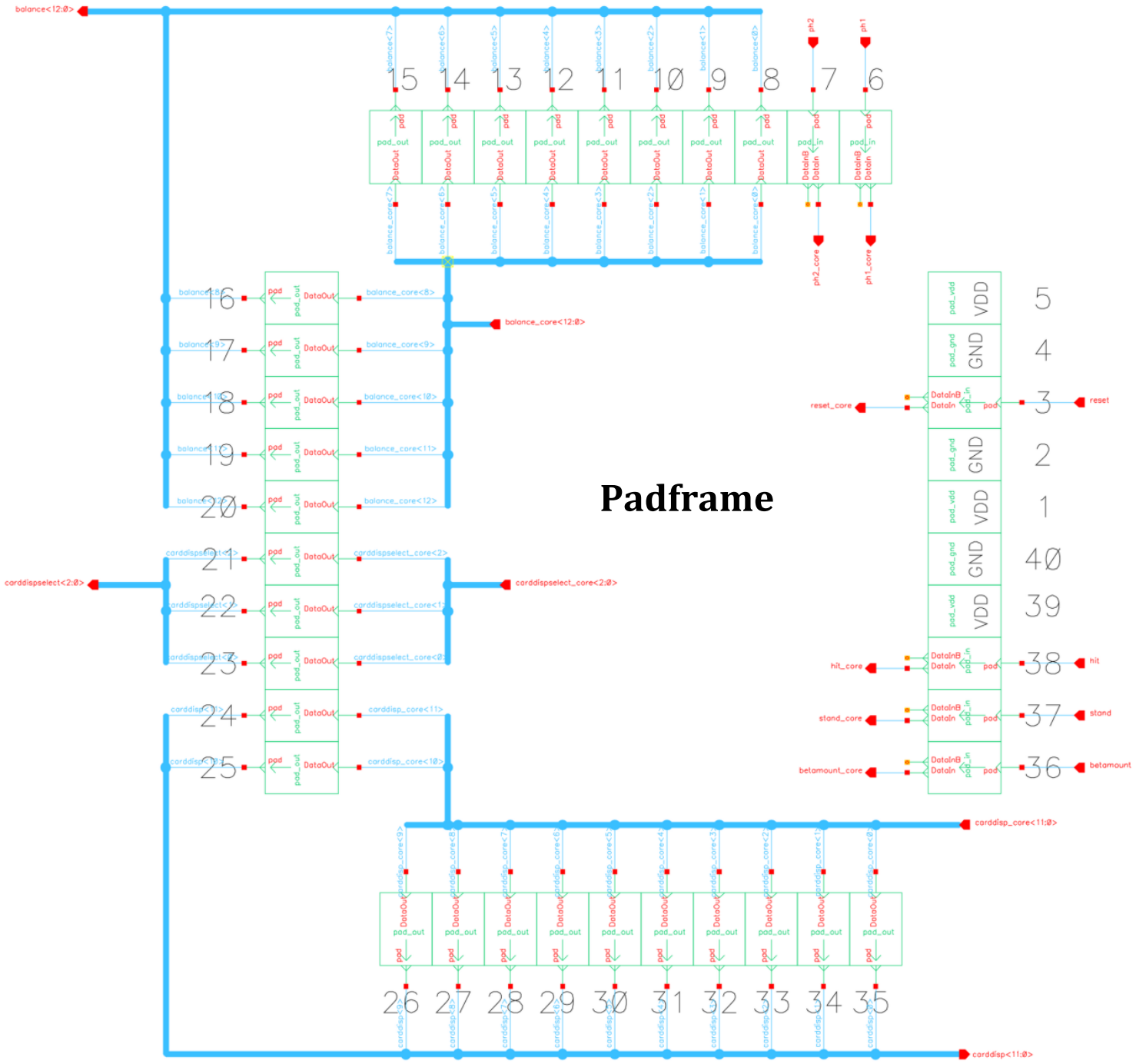
```
wire t1,t2 ,t3 ,t4,t5;
flop       #(1) r5f1(ph1, ph2, (~t1)|reset, t2);
flop       #(1) r5f2(ph1, ph2, t2, t3);
flop       #(1) r5f3(ph1, ph2, t3, t4);
flop       #(1) r5f4(ph1, ph2, t4, t5);
flop       #(1) r5f5(ph1, ph2, t5, t1);

assign bit4 = t1;

endmodule

module ring11 (input logic reset,ph1,ph2,
                              output logic bit2);
wire t1,t2 ,t3 ,t4,t5,t6,t7,t8,t9,t10,t11;
flop       #(1) r11f1(ph1, ph2, (~t1)|reset, t2);
flop       #(1) r11f2(ph1, ph2, t2, t3);
flop       #(1) r11f3(ph1, ph2, t3, t4);
flop       #(1) r11f4(ph1, ph2, t4, t5);
flop       #(1) r11f5(ph1, ph2, t5, t6);
flop       #(1) r11f6(ph1, ph2, t6, t7);
flop       #(1) r11f7(ph1, ph2, t7, t8);
flop       #(1) r11f8(ph1, ph2, t8, t9);
flop       #(1) r11f9(ph1, ph2, t9, t10);
flop       #(1) r11f10(ph1, ph2, t10, t11);
flop       #(1) r11f11(ph1, ph2, t11, t1);
assign bit2 = t1;

endmodule

module ring13 (input logic reset,ph1,ph2,
                              output logic bit1);
wire t1,t2 ,t3 ,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13;
flop       #(1) r13f1(ph1, ph2, (~t1)|reset, t2);
flop       #(1) r13f2(ph1, ph2, t2, t3);
flop       #(1) r13f3(ph1, ph2, t3, t4);
flop       #(1) r13f4(ph1, ph2, t4, t5);
flop       #(1) r13f5(ph1, ph2, t5, t6);
flop       #(1) r13f6(ph1, ph2, t6, t7);
flop       #(1) r13f7(ph1, ph2, t7, t8);
flop       #(1) r13f8(ph1, ph2, t8, t9);
flop       #(1) r13f9(ph1, ph2, t9, t10);
flop       #(1) r13f10(ph1, ph2, t10, t11);
flop       #(1) r13f11(ph1, ph2, t11, t12);
flop       #(1) r13f12(ph1, ph2, t12, t13);
flop       #(1) r13f13(ph1, ph2, t13, t1);
assign bit1 = t1;

endmodule


module mux10 (input logic [7:0] d0, d1, d2 ,d3 ,d4, d5 ,d6 ,d7, d8, d9,
                      input logic [3:0] alusrcb,
                      output logic [7:0] constantbank);
always_comb begin
case(alusrcb)
4'b0000:  constantbank <= d0;
4'b0001:  constantbank <= d1;
4'b0010:  constantbank <= d2;
4'b0011:  constantbank <= d3;
```

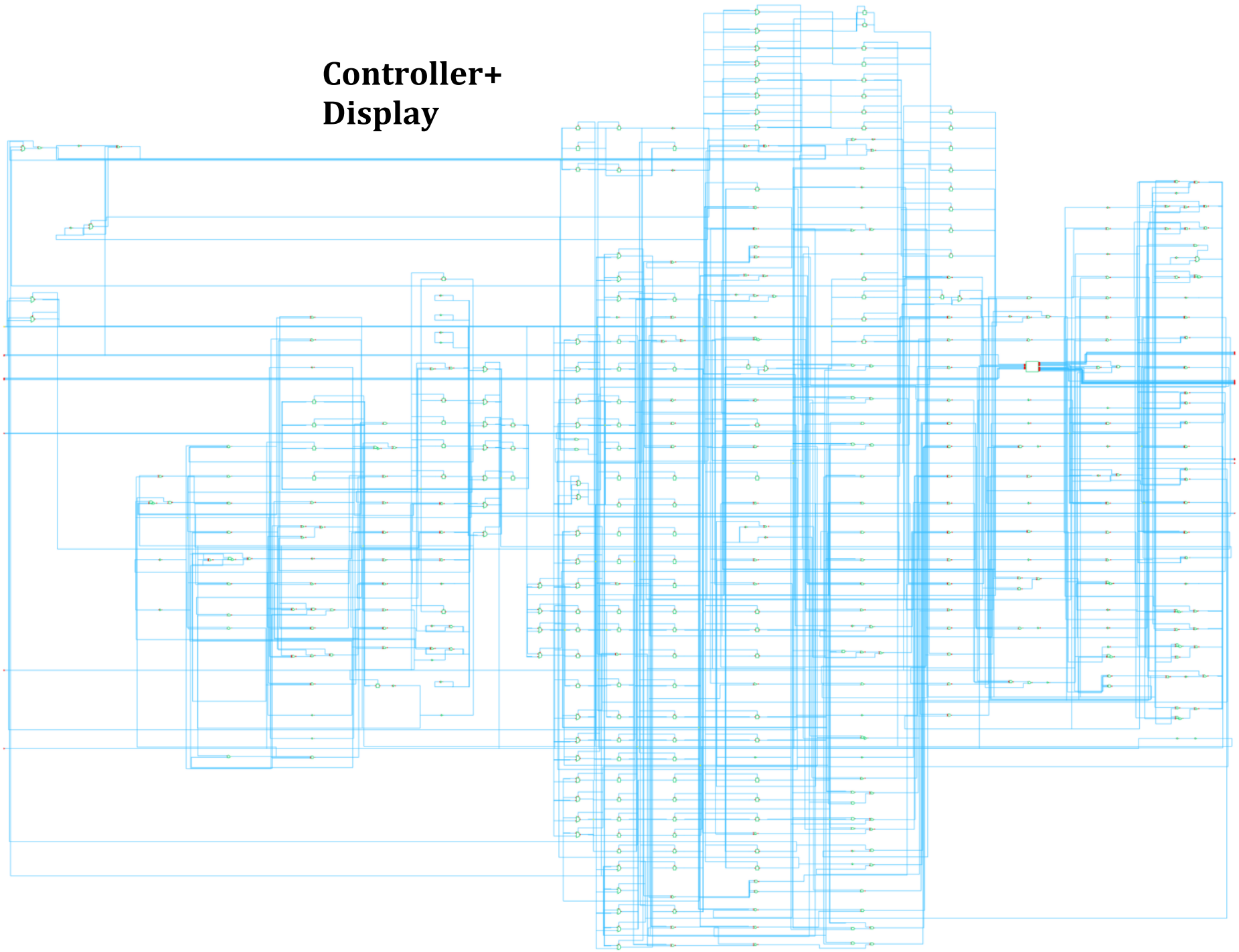
```
4'b0100:  constantbank <= d4;
4'b0101:  constantbank <= d5;
4'b0110:  constantbank <= d6;
4'b0111:  constantbank <= d7;
4'b1000:  constantbank <= d8;
4'b1001:  constantbank <= d9;
default:  constantbank <= d0;
endcase
end
endmodule


module mux13 (input logic [7:0] d0, d1, d2 ,d3 ,d4, d5 ,d6 ,d7, d8, d9, d10, d11, d12,
                      input logic [7:0] srcb,
                      output logic [7:0] cardvalue);
always_comb begin
case(srcb)
8'b00000000:  cardvalue <= d0;
8'b00000001:  cardvalue <= d1;
8'b00000010:  cardvalue <= d2;
8'b00000011:  cardvalue <= d3;
8'b00000100:  cardvalue <= d4;
8'b00000101:  cardvalue <= d5;
8'b00000110:  cardvalue <= d6;
8'b00000111:  cardvalue <= d7;
8'b00001000:  cardvalue <= d8;
8'b00001001:  cardvalue <= d9;
8'b00001010:  cardvalue <= d10;
8'b00001011:  cardvalue <= d11;
8'b00001100:  cardvalue <= d12;
default:  cardvalue <= d0;
endcase
end
endmodule

module mux14 #(parameter WIDTH = 4) (input logic [WIDTH-1:0] d0, d1, d2 ,d3 ,d4, d5 ,d6 ,d7, d8, d9, d10, d11, d12, d13,
                      input logic [3:0] srcb,
                      output logic [WIDTH-1:0] cardvalue);
always_comb begin
case(srcb)
4'b0000:  cardvalue <= d0;
4'b0001:  cardvalue <= d1;
4'b0010:  cardvalue <= d2;
4'b0011:  cardvalue <= d3;
4'b0100:  cardvalue <= d4;
4'b0101:  cardvalue <= d5;
4'b0110:  cardvalue <= d6;
4'b0111:  cardvalue <= d7;
4'b1000:  cardvalue <= d8;
4'b1001:  cardvalue <= d9;
4'b1010:  cardvalue <= d10;
4'b1011:  cardvalue <= d11;
4'b1100:  cardvalue <= d12;
4'b1101:  cardvalue <= d13;
default:  cardvalue <= d0;
endcase
end
endmodule
```

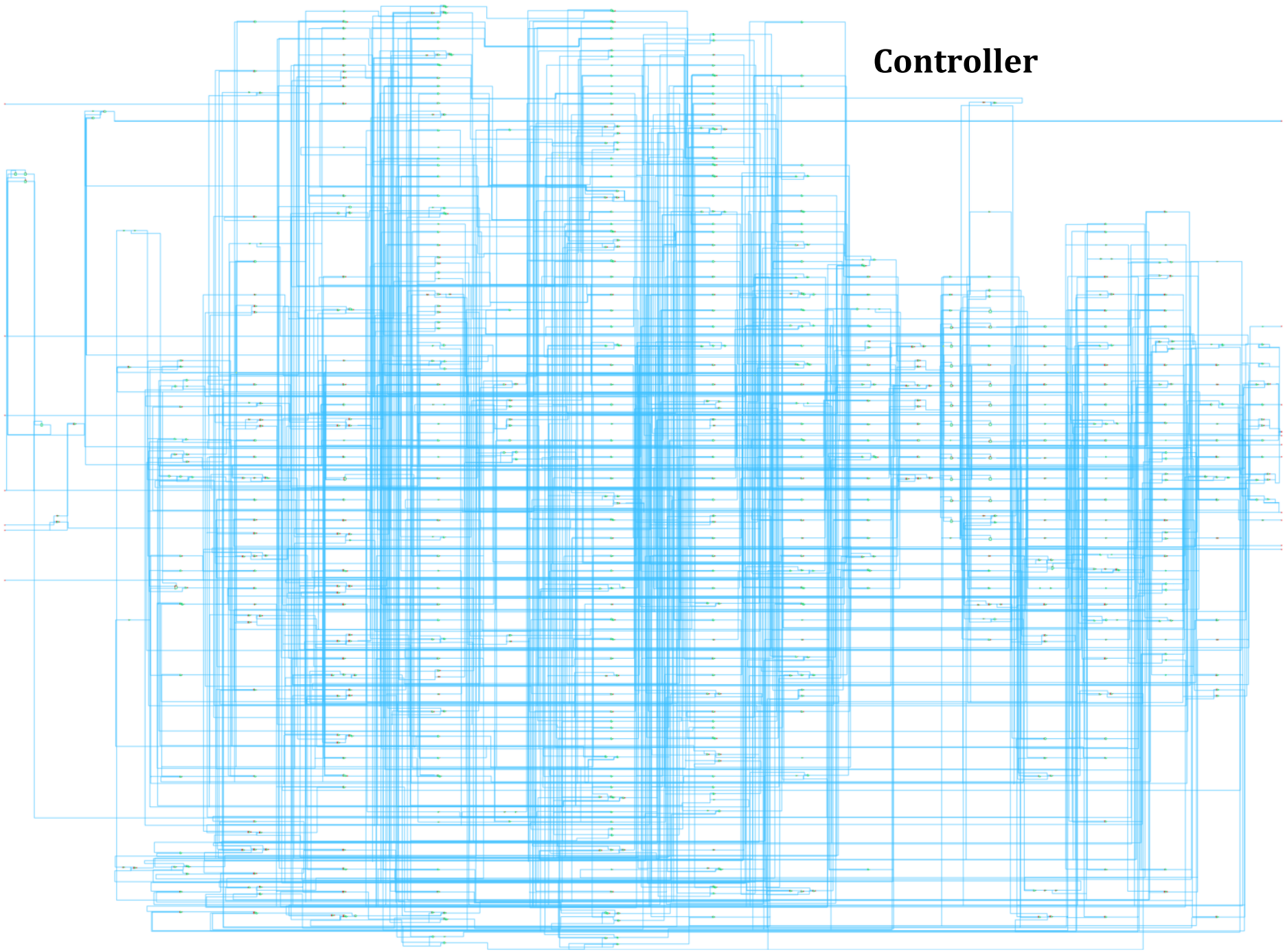
Appendix B

Schematics

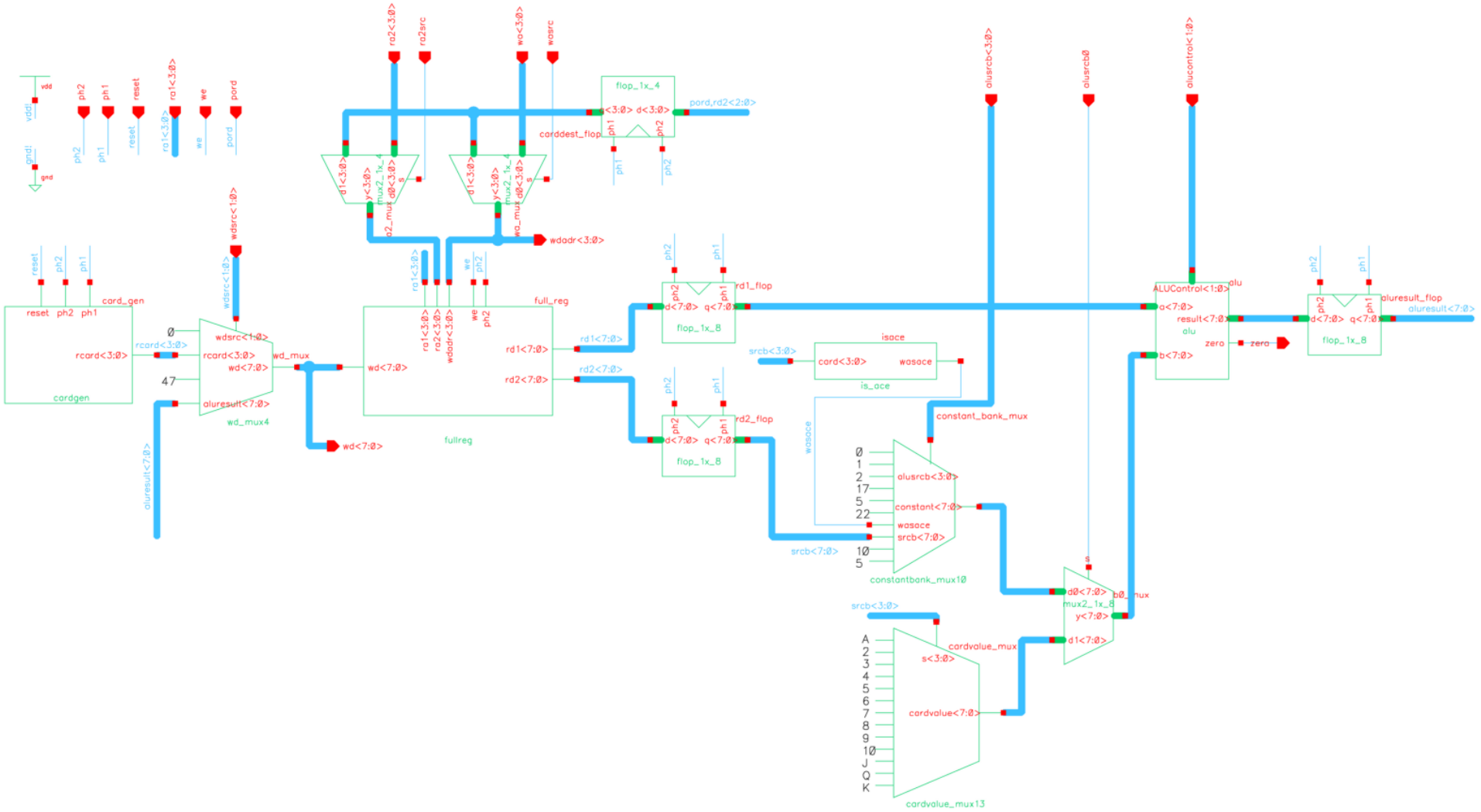



Padframe
balance<12:0>
balance_core<12:0>
corddispselect<2:0>
corddispselect_core<2:0>
corddisp_core<11:0>
corddisp<11:0>
reset_core
reset
hit_core
hit
stand_core
stand
betamount_core
betamount
ph2_core
ph1_core
VDD
GND
pad_out
DataOut
DataInB
DataIn
pad

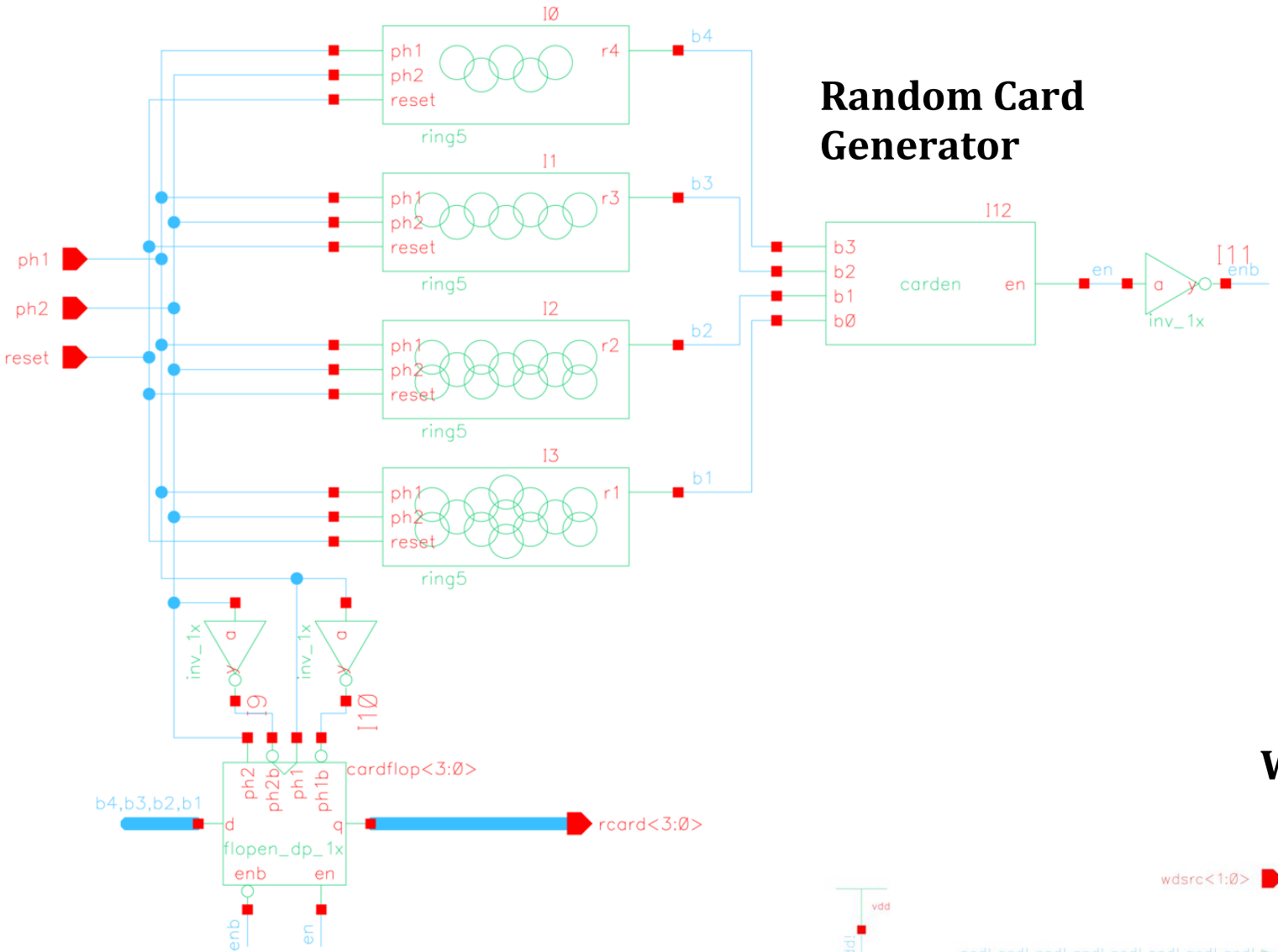

Controller+
Display


Controller

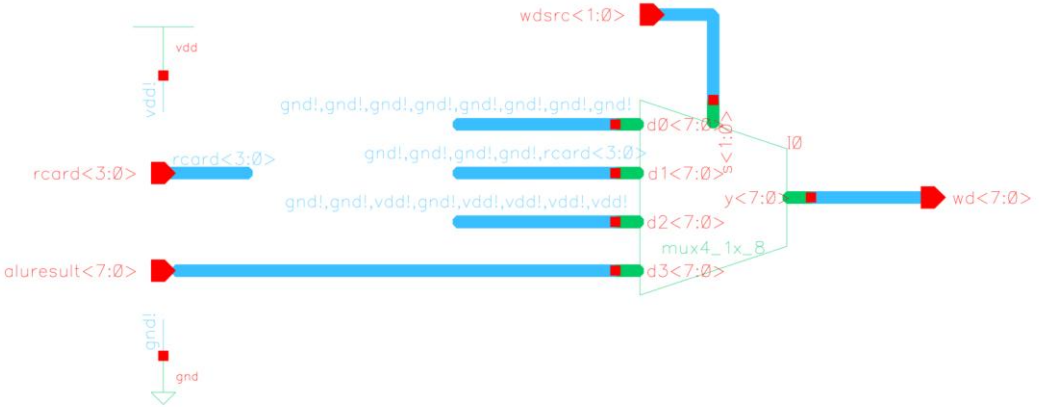
Datapath


vdd!
gnd!
ph2
ph1
reset
ra1<3:0>
we
pord
ra2<3:0>
ra2src
wa<3:0>
wasrc
flop_1x_4
d<3:0>
q<3:0>
pord,rd2<2:0>
carddest_flop
mux2_1x_4
y<3:0>
wa_mux<3:0>
wdadr<3:0>
wdsrc<1:0>
card_gen
cardgen
rcard<3:0>
wd_mux4
wd_mux
wd<7:0>
aluresult<7:0>
full_reg
fullreg
rd1<7:0>
rd2<7:0>
rd1_flop
rd2_flop
flop_1x_8
alusrcb<3:0>
alusrcb0
alucontrol<1:0>
ALUControl<1:0>
alu
a<7:0>
b<7:0>
result<7:0>
zero
aluresult_flop
isace
is_ace
card<3:0>
wasace
srcb<3:0>
srcb<7:0>
constant_bank_mux
constantbank_mux10
constant<7:0>
cardvalue_mux
cardvalue_mux13
cardvalue<7:0>
s<3:0>
mux2_1x_8
d0<7:0>
d1<7:0>
y<7:0>
s

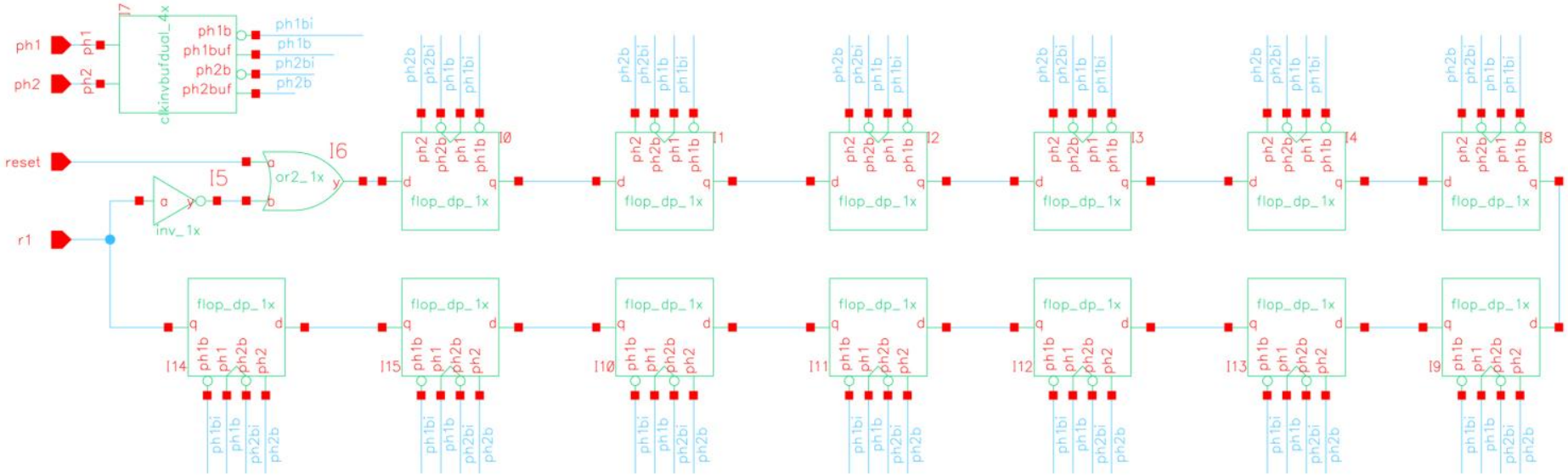
Random Card Generator



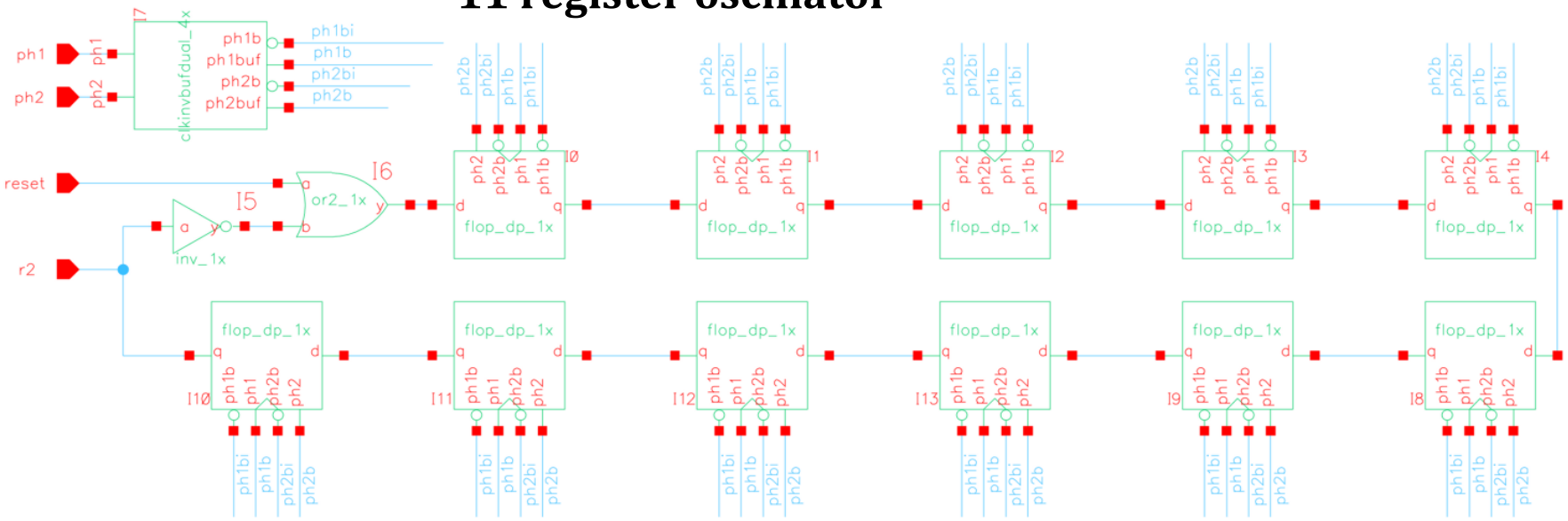
WriteData Mux

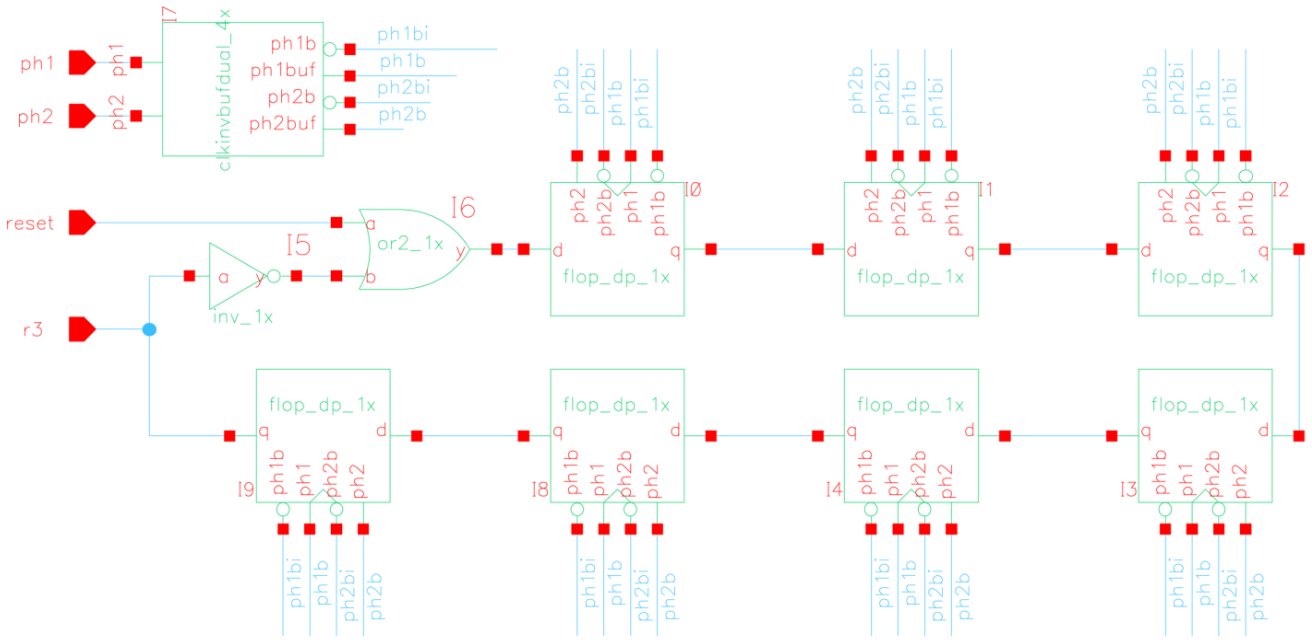


13 register oscillator

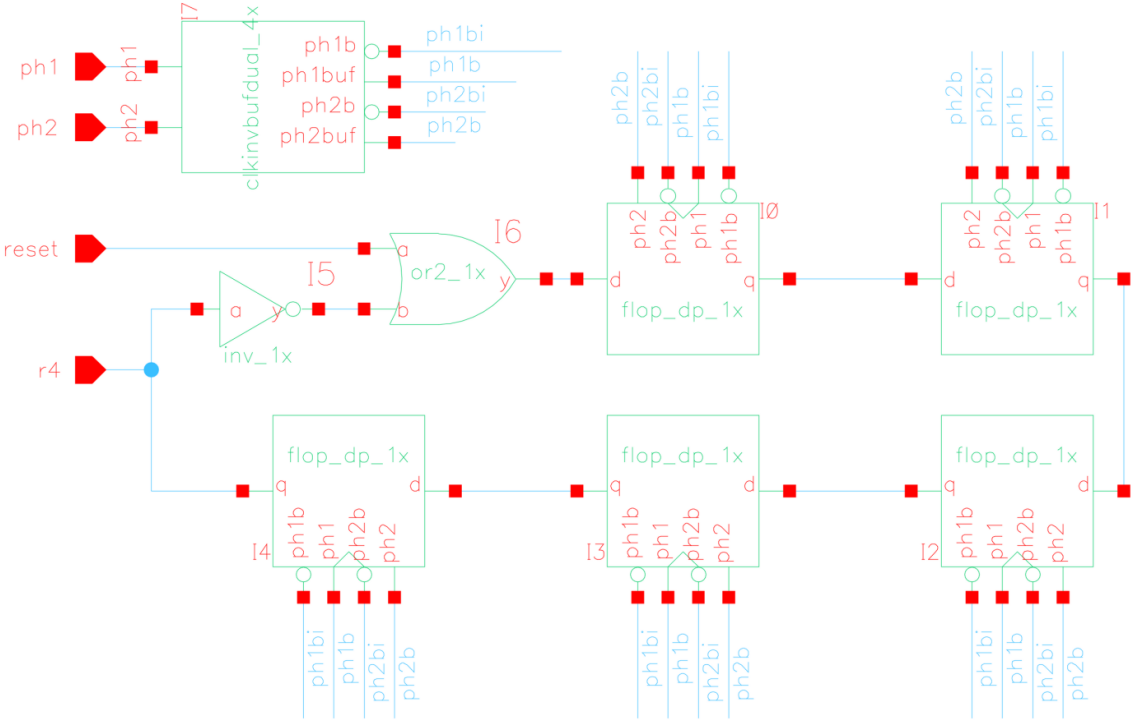


11 register oscillator



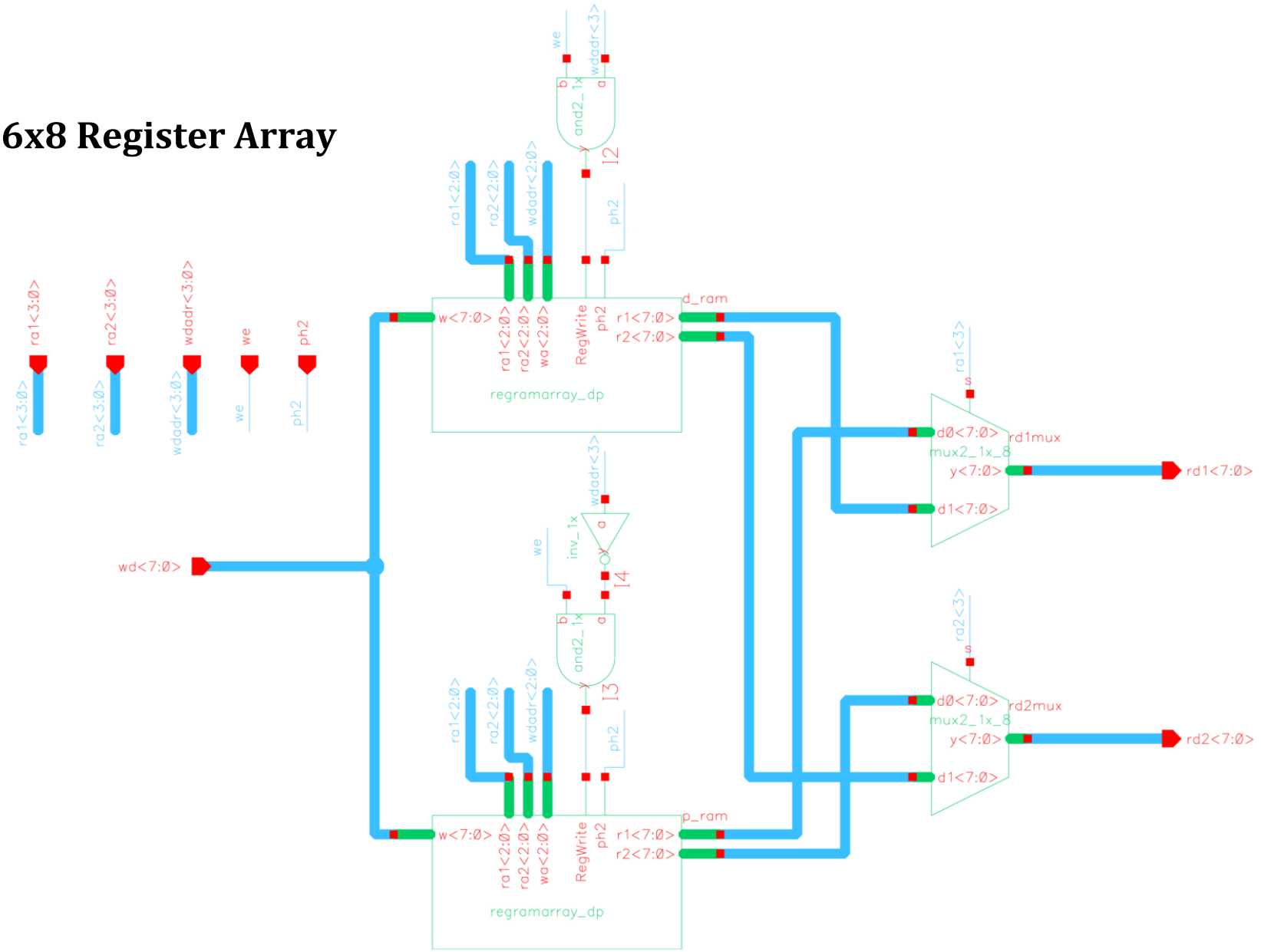


7 register oscillator

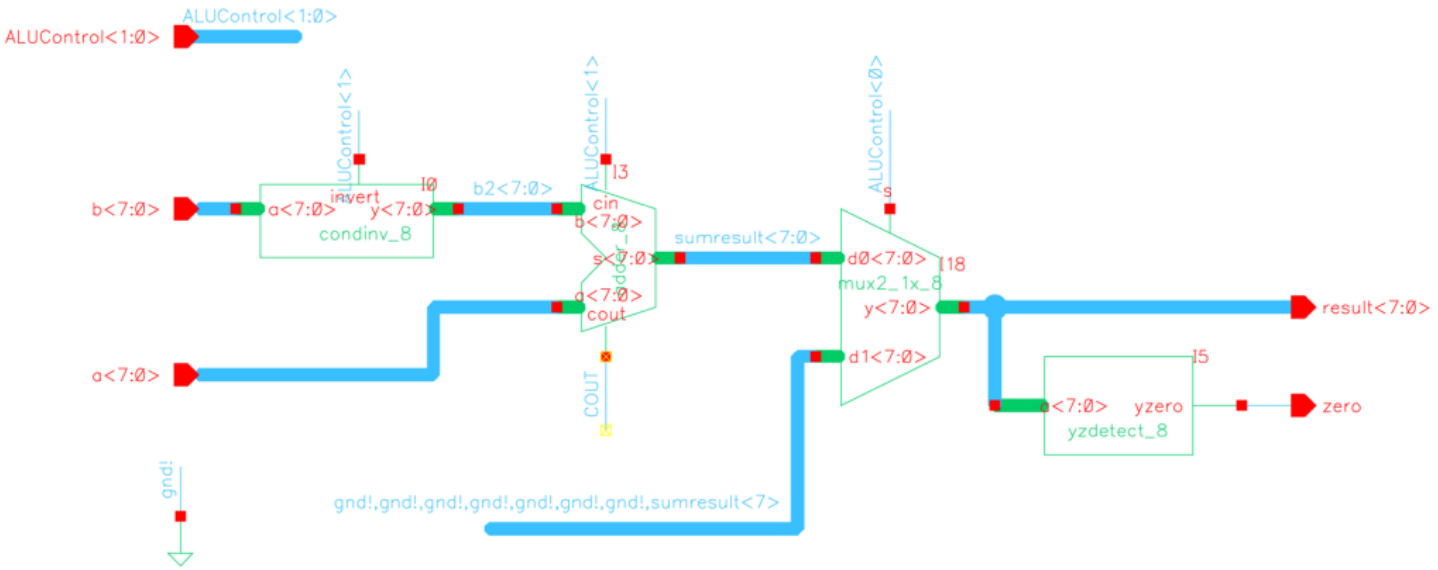


5 register oscillator

16x8 Register Array



ALU with add, sub, and SLT capability



Harvey Mudd College

UPDATED
Apr 15 21:51:33 2010

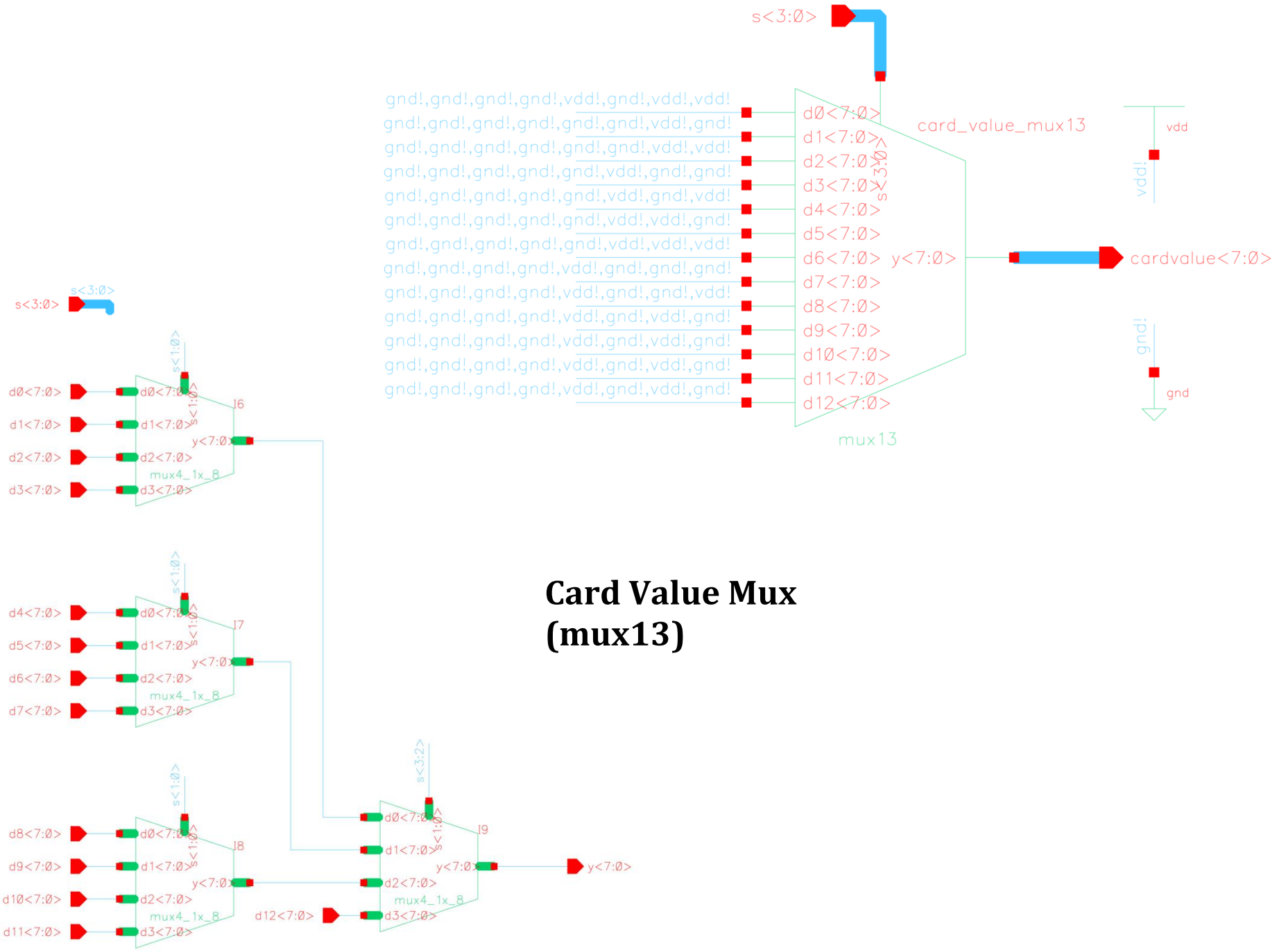
BY
zflom

alu

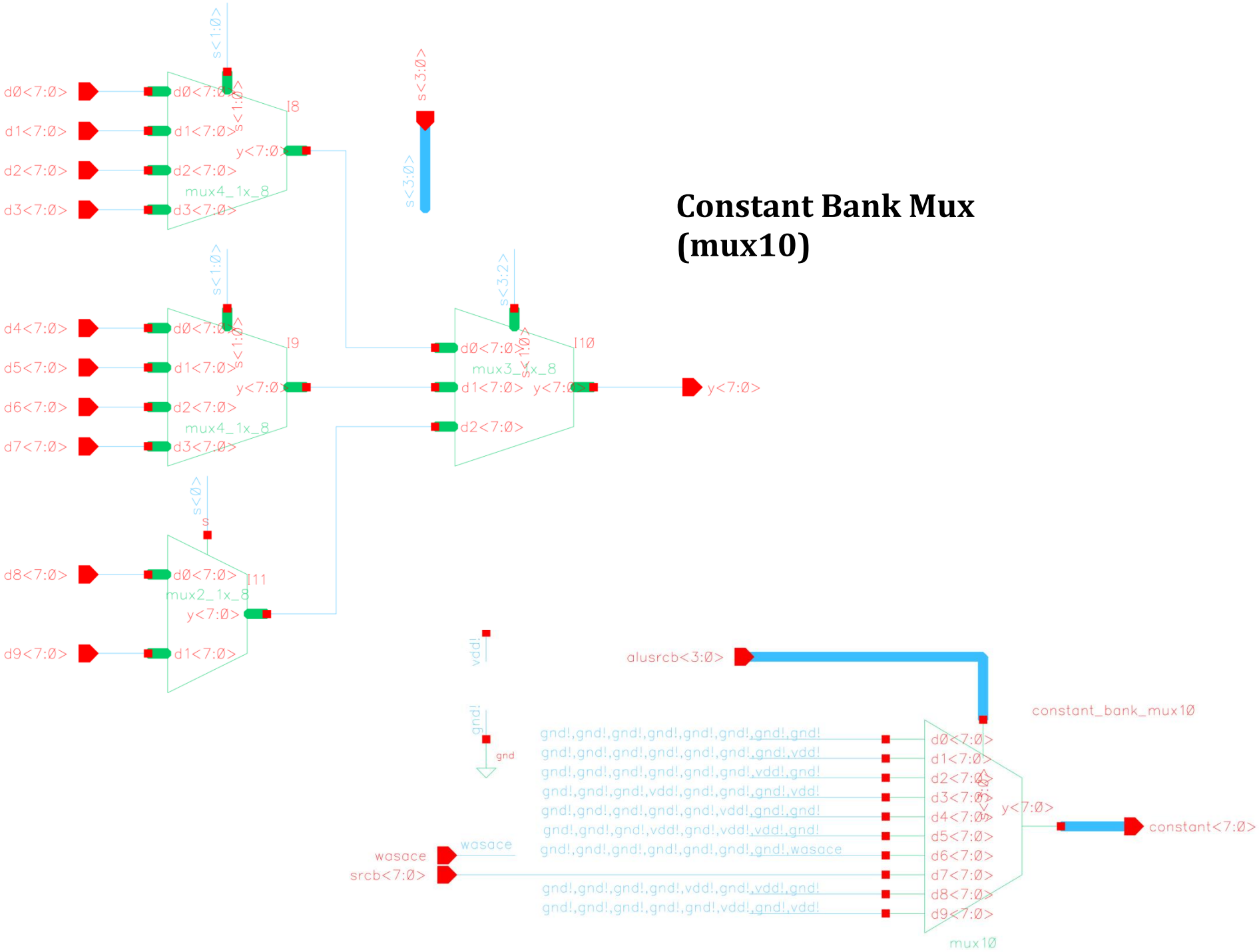
SIZE
A

REV

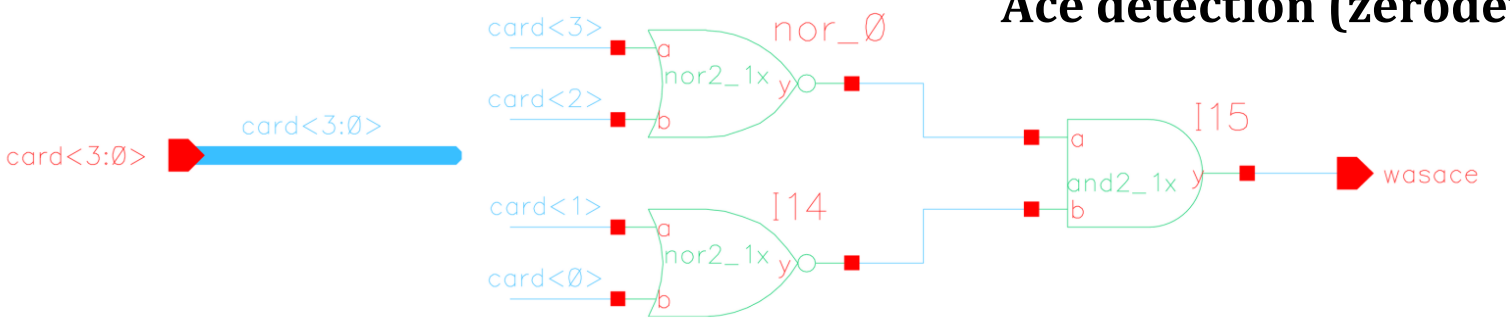
SHEET


s<3:0>
gnd!,gnd!,gnd!,gnd!,vdd!,gnd!,vdd!,vdd!
gnd!,gnd!,gnd!,gnd!,gnd!,gnd!,vdd!,gnd!
gnd!,gnd!,gnd!,gnd!,gnd!,gnd!,vdd!,vdd!
gnd!,gnd!,gnd!,gnd!,gnd!,vdd!,gnd!,gnd!
gnd!,gnd!,gnd!,gnd!,gnd!,vdd!,gnd!,vdd!
gnd!,gnd!,gnd!,gnd!,gnd!,vdd!,vdd!,gnd!
gnd!,gnd!,gnd!,gnd!,gnd!,vdd!,vdd!,vdd!
gnd!,gnd!,gnd!,gnd!,vdd!,gnd!,gnd!,gnd!
gnd!,gnd!,gnd!,gnd!,vdd!,gnd!,gnd!,vdd!
gnd!,gnd!,gnd!,gnd!,vdd!,gnd!,vdd!,gnd!
gnd!,gnd!,gnd!,gnd!,vdd!,gnd!,vdd!,gnd!
gnd!,gnd!,gnd!,gnd!,vdd!,gnd!,vdd!,gnd!
d0<7:0>
d1<7:0>
d2<7:0>
d3<7:0>
d4<7:0>
d5<7:0>
d6<7:0>
d7<7:0>
d8<7:0>
d9<7:0>
d10<7:0>
d11<7:0>
d12<7:0>
s<3:2>
y<7:0>
card_value_mux13
mux13
vdd
vdd!
gnd!
gnd
cardvalue<7:0>
s<3:0>
s<1:0>
I6
I7
I8
I9
mux4_1x_8
d0<7:0>
d1<7:0>
d2<7:0>
d3<7:0>
d4<7:0>
d5<7:0>
d6<7:0>
d7<7:0>
d8<7:0>
d9<7:0>
d10<7:0>
d11<7:0>
d12<7:0>
y<7:0>
y<7:0>
Card Value Mux
(mux13)

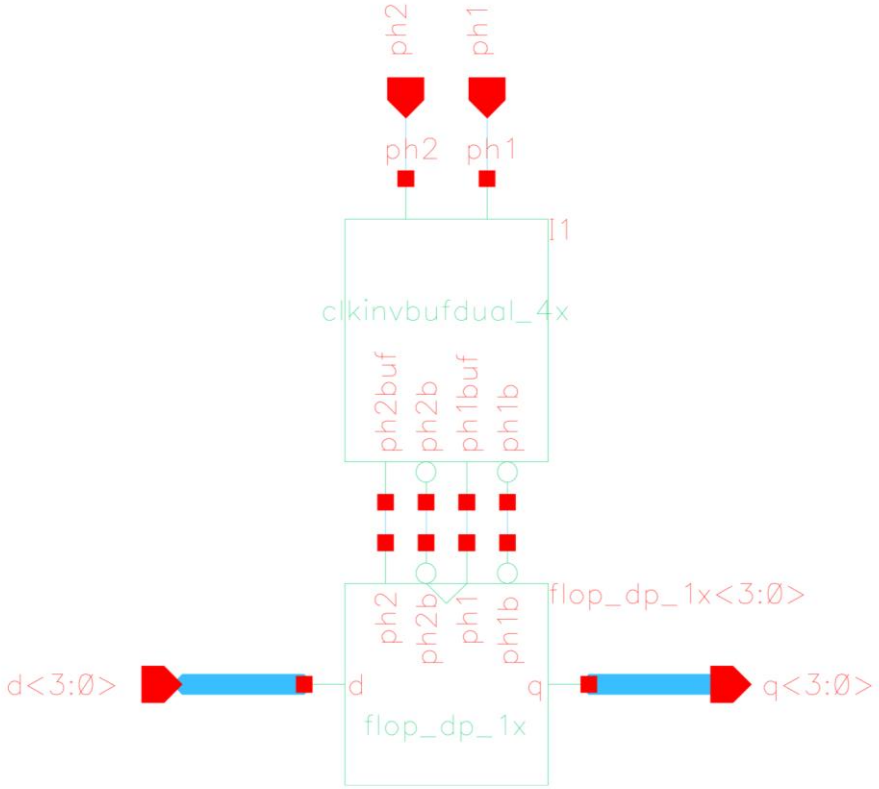
Constant Bank Mux (mux10)



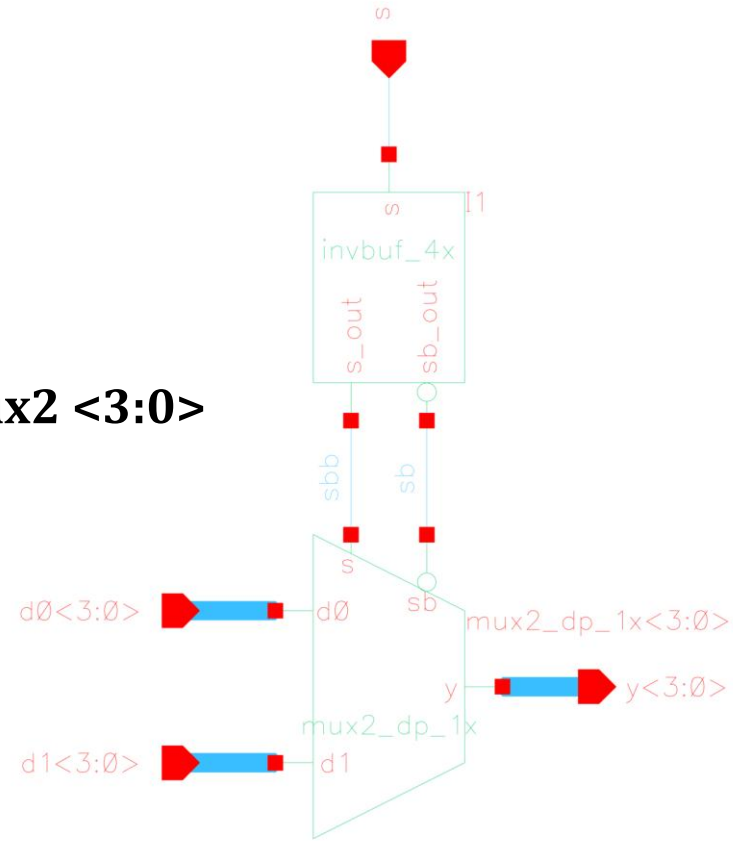
Ace detection (zerodetect)

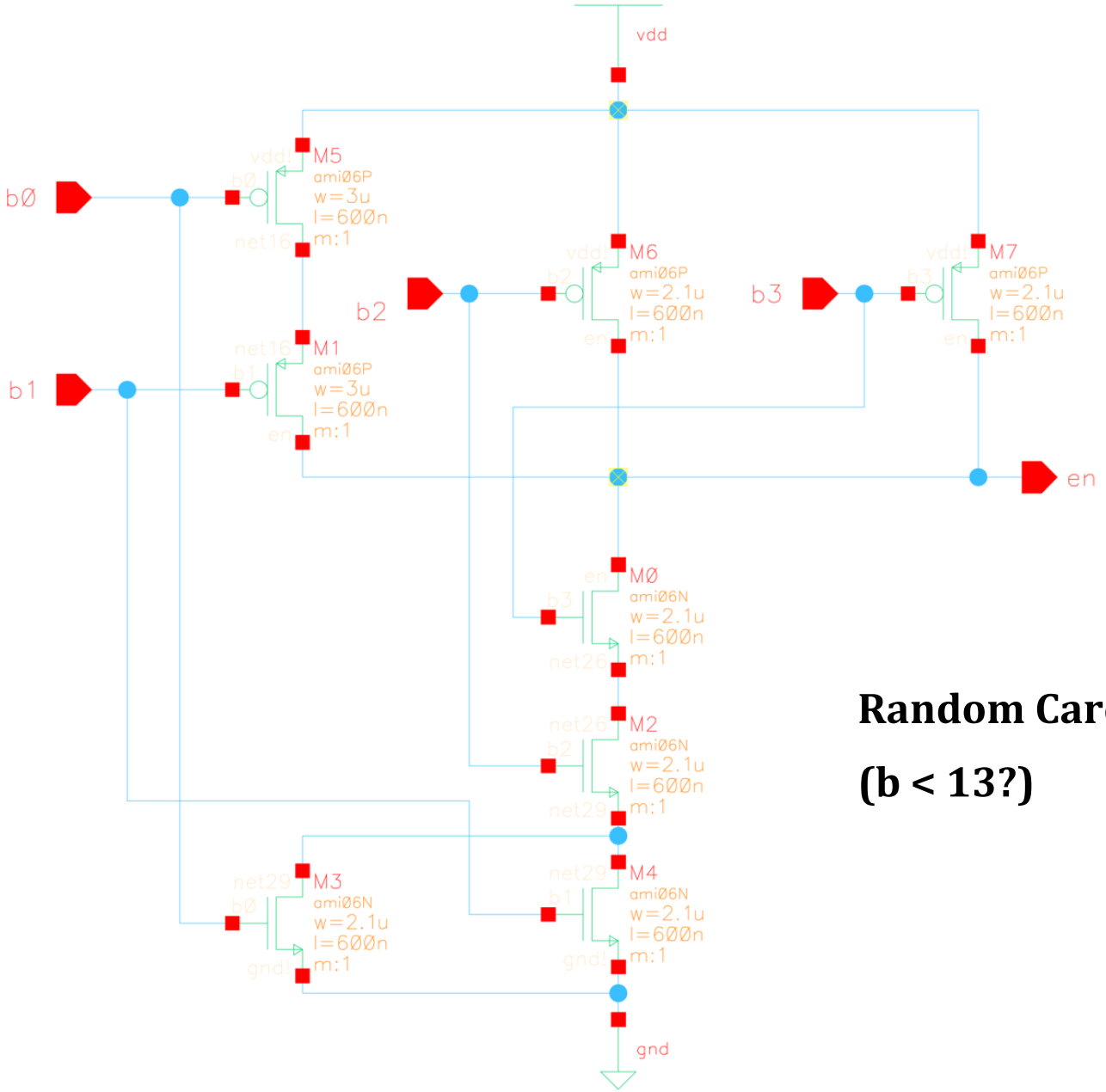


Feedback Flops (read data into read address)



Mux2 <3:0>



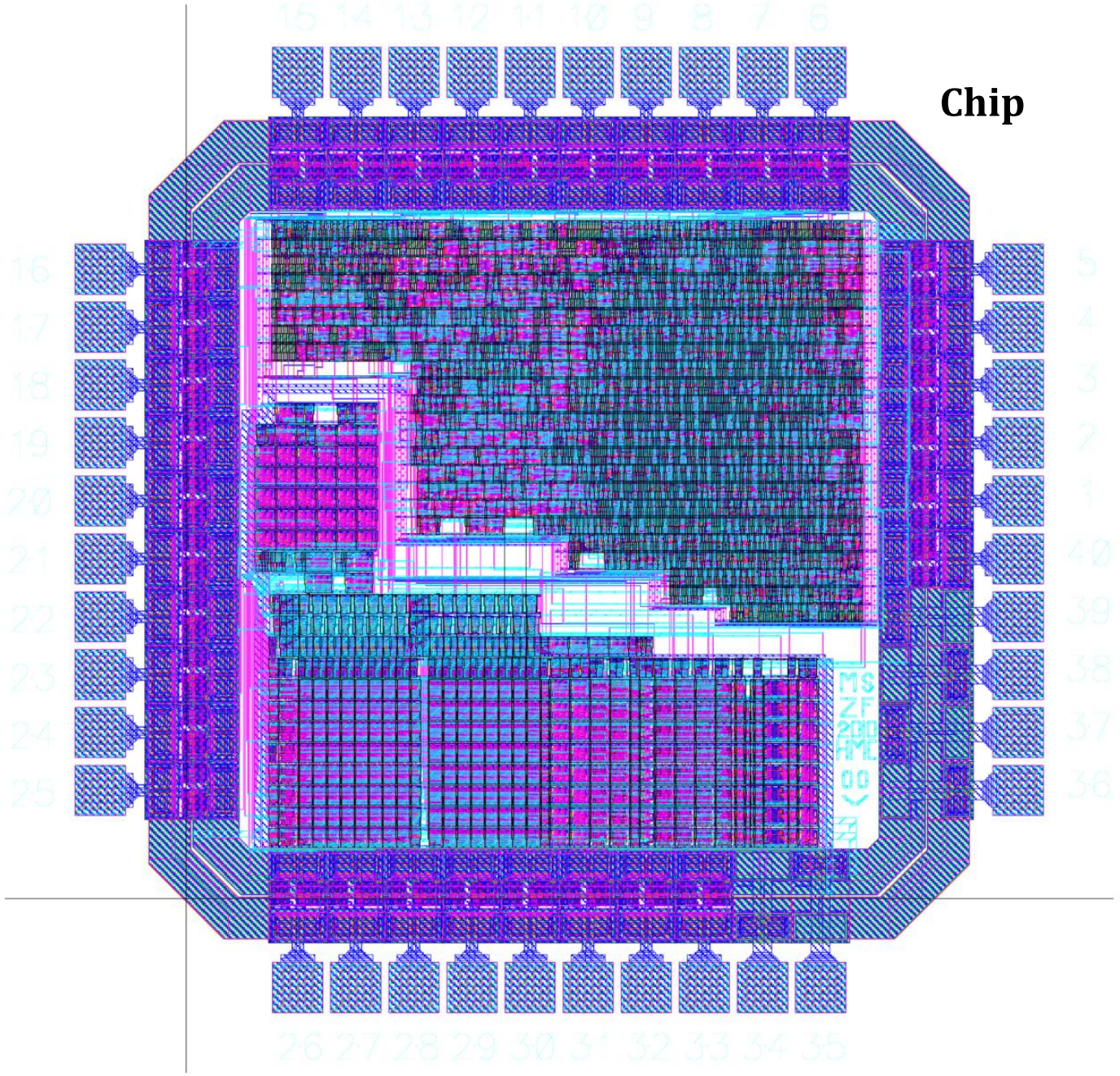


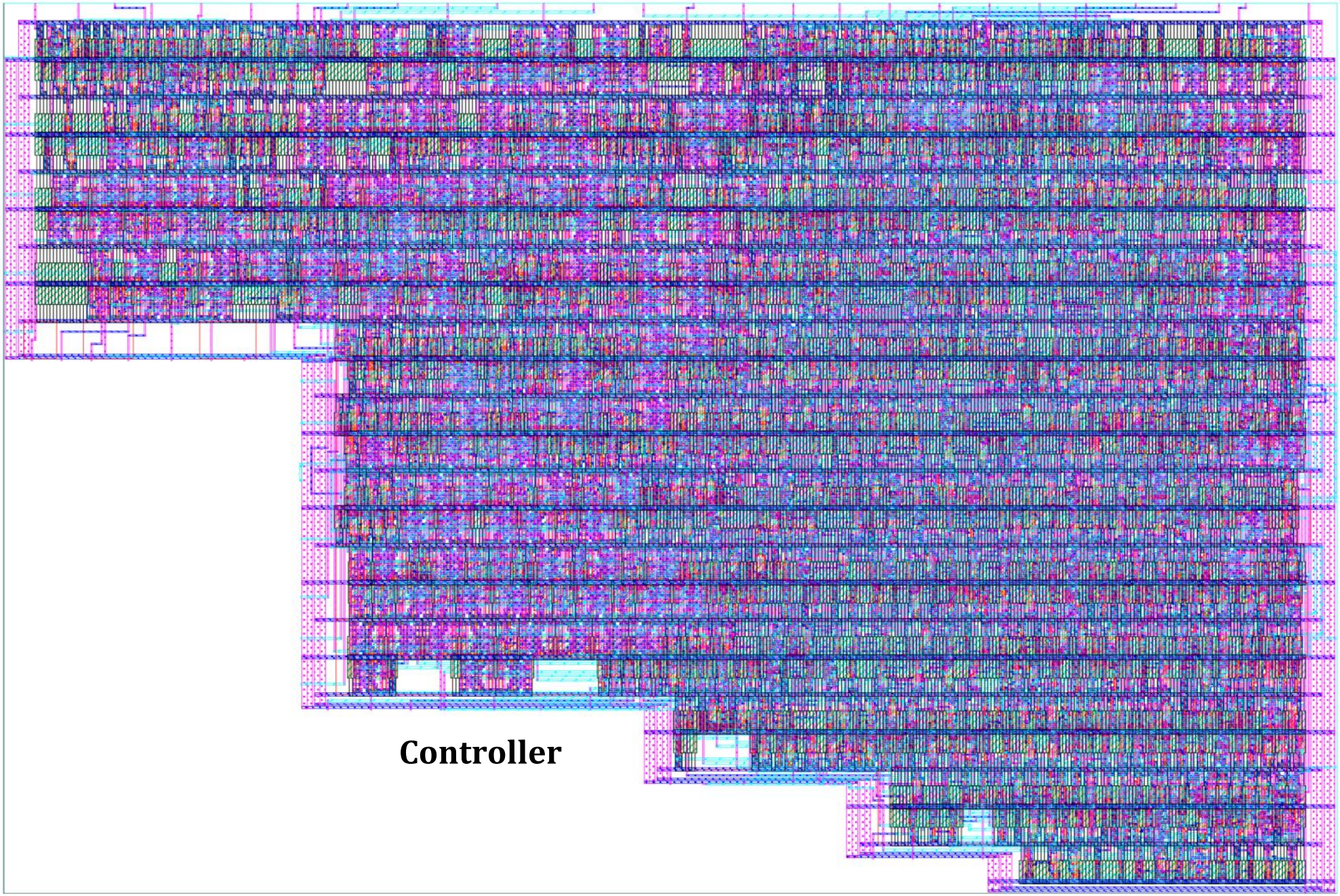
Random Card Enable

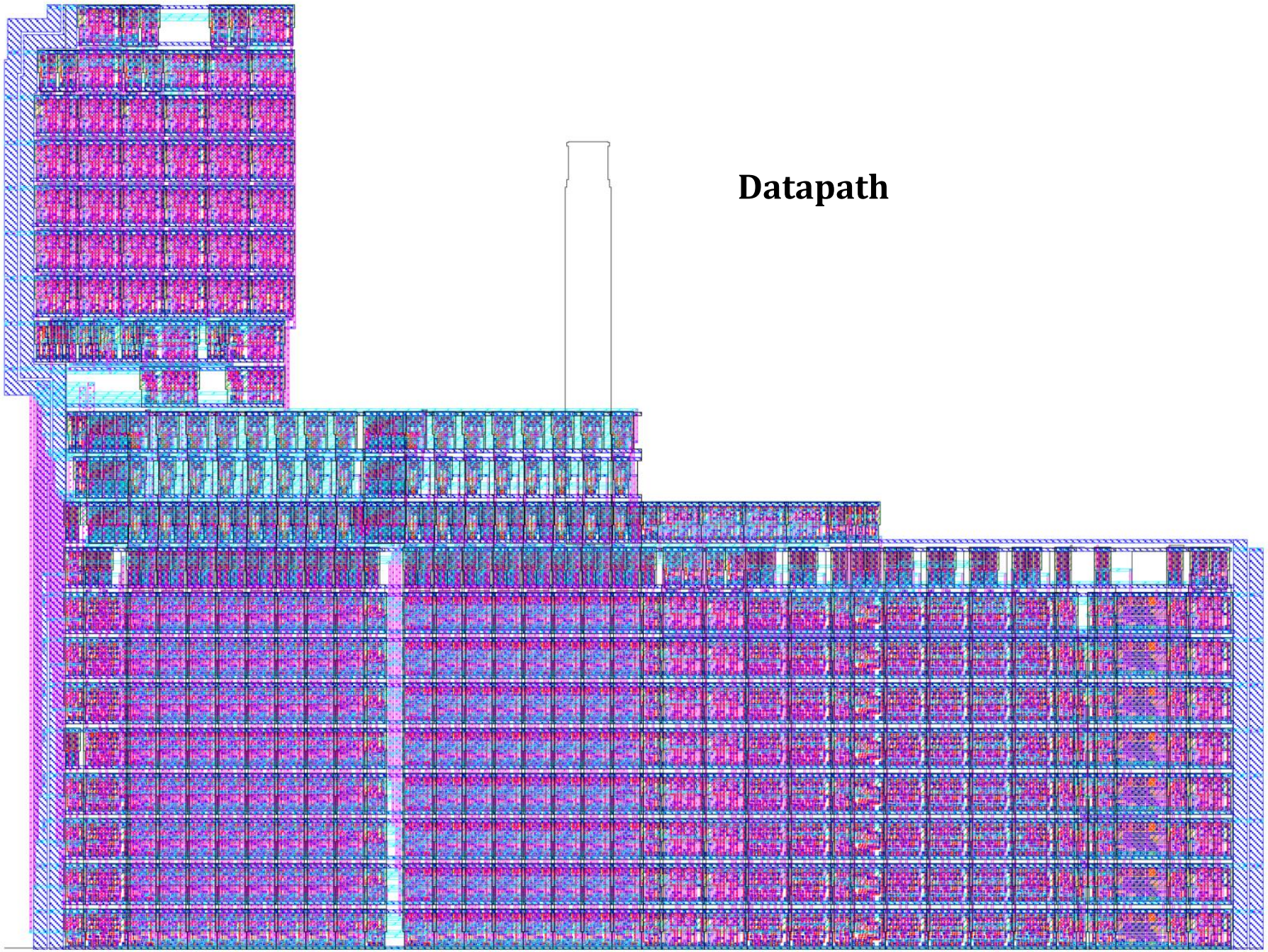
(b < 13?)

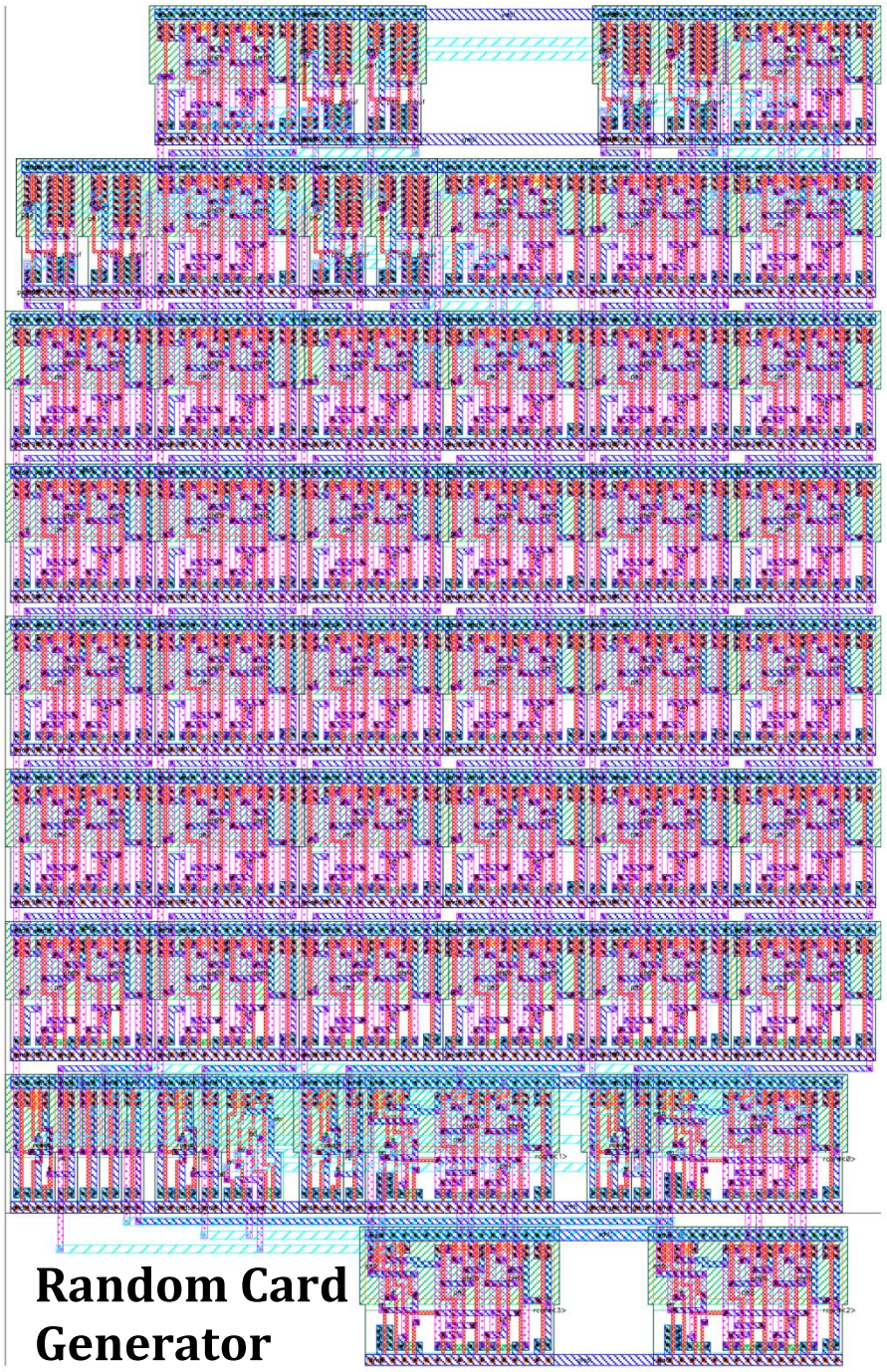
Appendix C

Layout



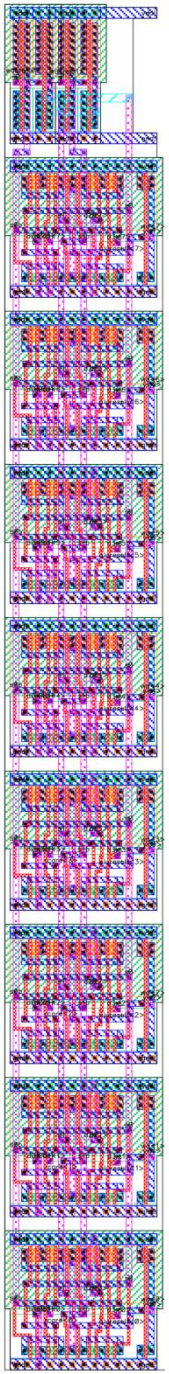

Controller

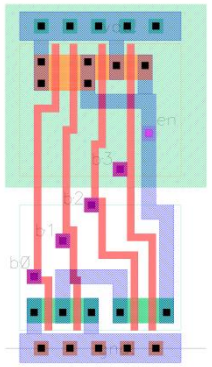

Datapath



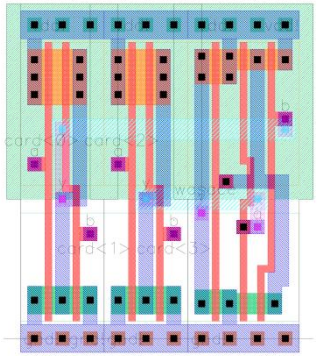
Random Card Generator

Write Data Mux4



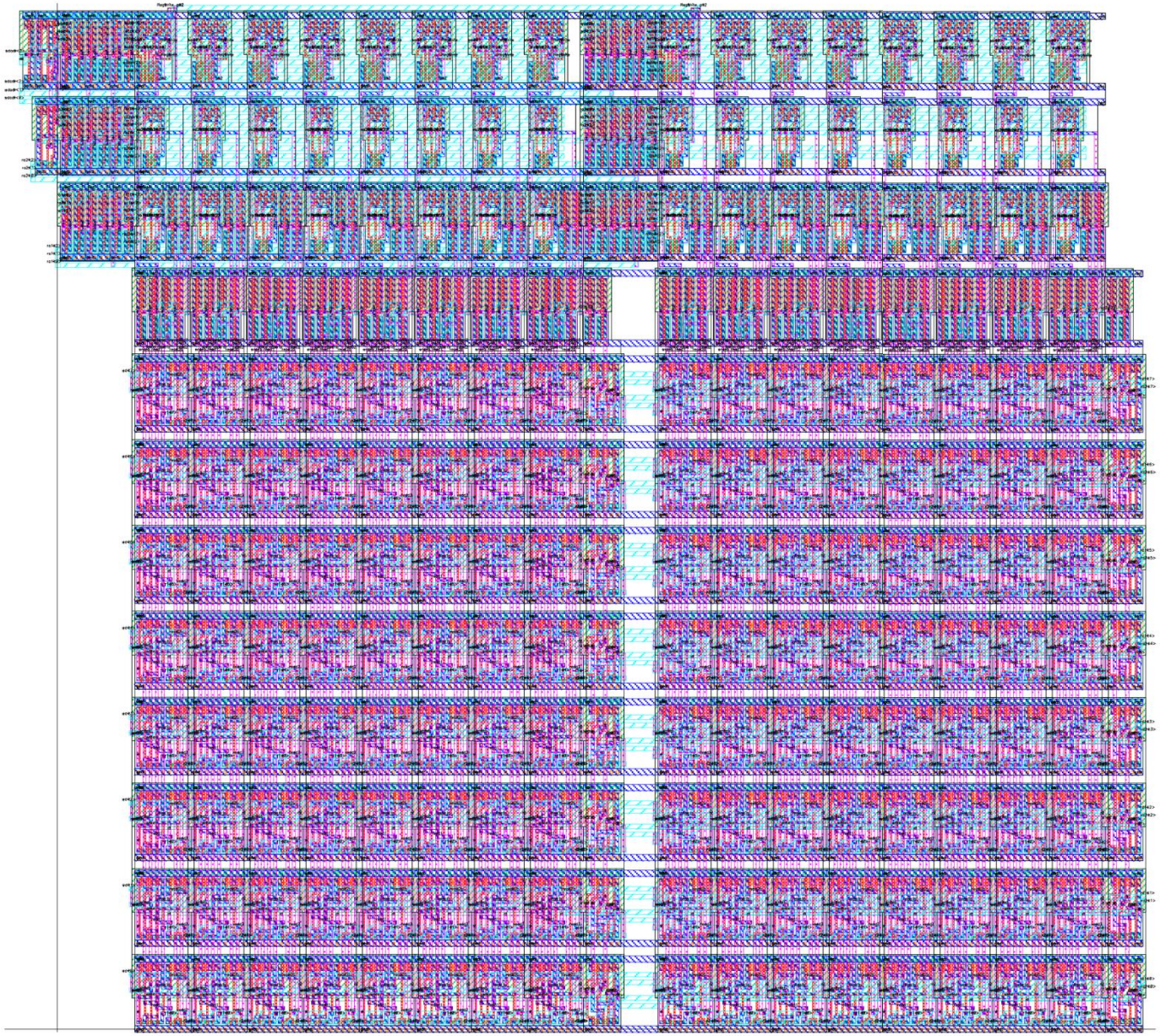


Random Card Enable (b < 13?)

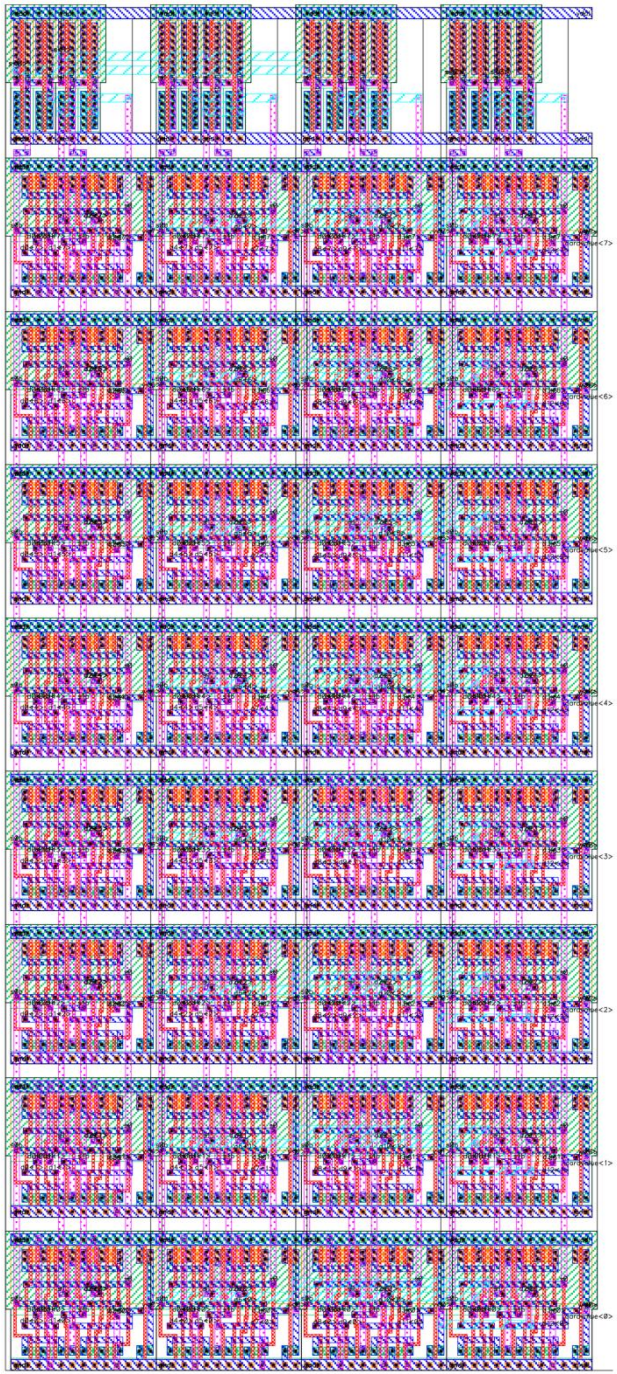


Ace Detector

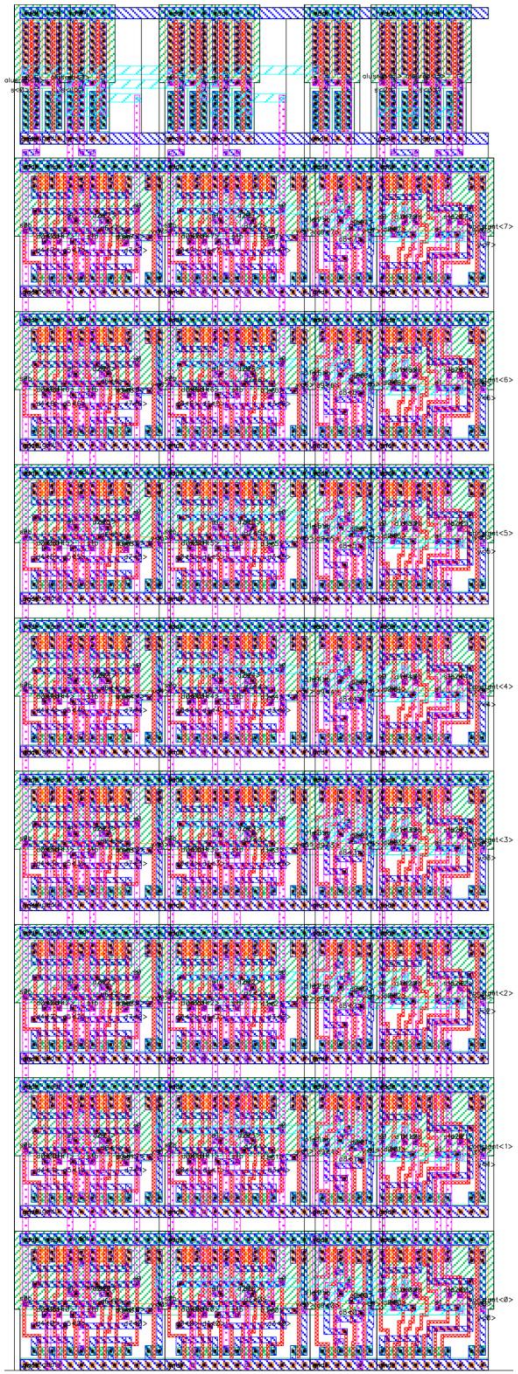
Full 16x8 Register Array



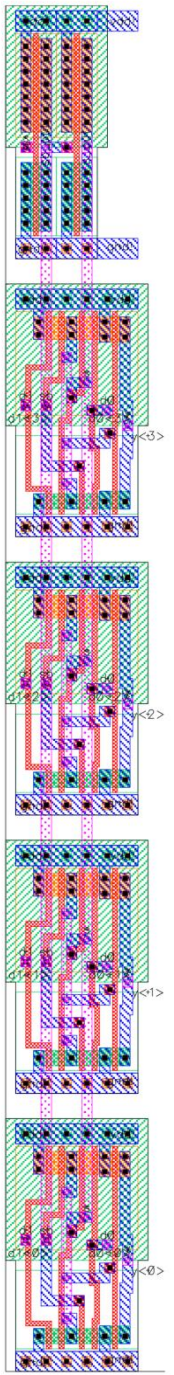
Mux 13 (card to point value)



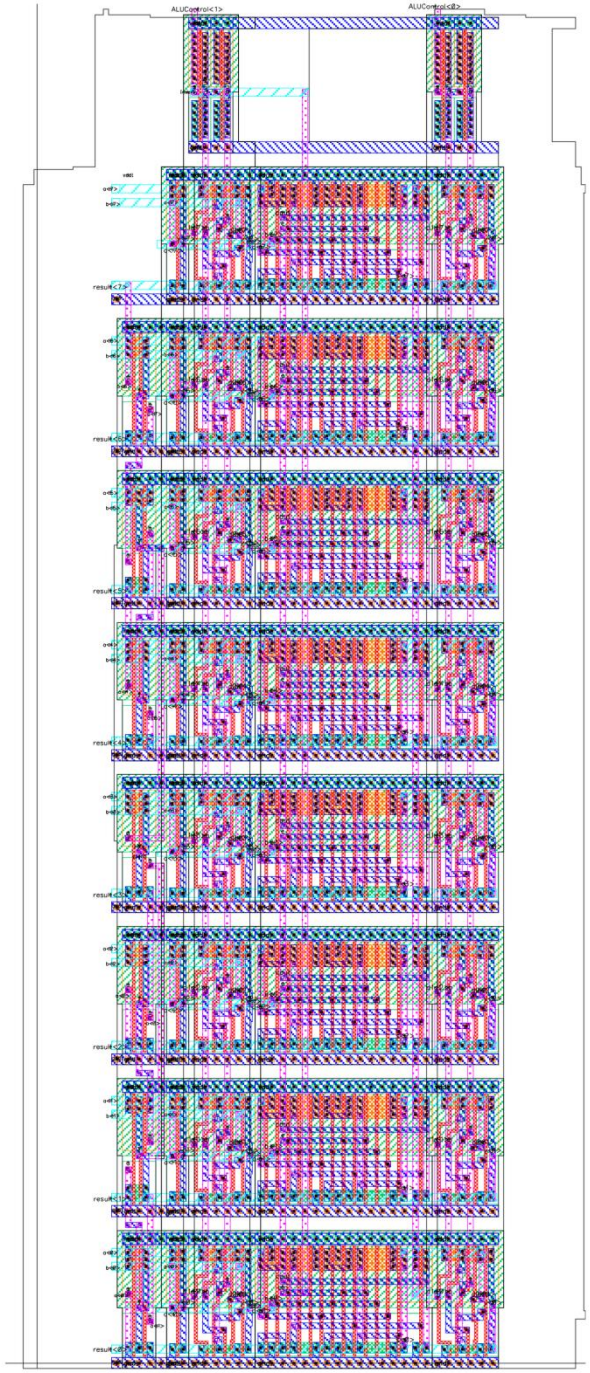
Mux 10



Mux2_1x_4



Simplified ALU



Flop_1x_4